Системы управления базами данных (СУБД)

Олег Анатольевич Тарлапан
Сергей Вячеславович Морозов
Институт системного программирования РАН
снс, кфмн

Основной учебник

учебное пособие «Базы данных» автор С.Д.Кузнецов (изд.центр Академия 2012)
учебное пособие «Основы баз данных» автор С.Д.Кузнецов (изд. Бином 2007)
учебное пособие «Базы данных. Модели и языки» автор С.Д.Кузнецов (изд. 2008)

Интернет
Интерактивный курс С.Д. Кузнецова «Введение в реляционные базы данных»
Университет Информационных технологий **www.intuit.ru** (текстовый **www.citforum.ru**)

Дополнительная литература:
К. Дж. Дейт. Введение в системы баз данных, 8-е издание. М., Вильямс, 2005
К. Дж. Дейт, Хью Дарвен. Основы будущих баз данных. Третий манифест. М., Янус-К, 2004
Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию. Учебник. (Исправл. изд.). - М., "Финансы и статистика", 2006.

Просьба старостам групп оставить контактные телефоны. Фамилия, имя, отчество, группа и телефон (для форс-мажорных случаев)

Информационная система
Файловая система
Система Управления Базами Данных (СУБД)

Информационная система - программно-аппаратный комплекс, выполняющий три основные функции:

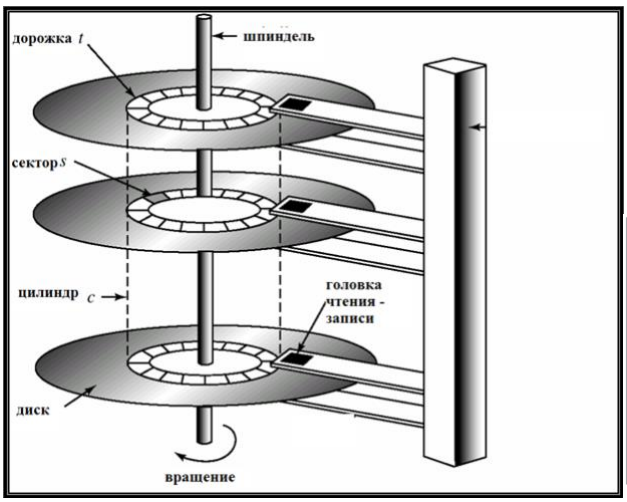
- поддержка надежного хранения информации,
- выполнение специфических для данной системы преобразований и вычислений,
- предоставление пользователю удобного интуитивного интерфейса.

Файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

Функции системы управления файлами:

- распределение внешней памяти,
- отображение имен файлов в соответствующие адреса во внешней памяти,
- обеспечение доступа к данным.

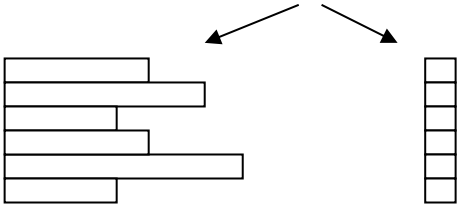
Структура файлов



В настоящее время возможность обмениваться с магнитными дисками порциями меньше объема блока не используется в файловых системах.
Во всех файловых системах явно или неявно выделяется **базовый уровень**, обеспечивающий работу с файлами, представляющими набор прямо адресуемых в адресном пространстве файла блоков. Размер логических блоков равен или кратен размеру физического блока диска и размеру страницы виртуальной памяти.

Обычно в файловых системах базовый уровень не доступен пользователю, и закрывается интерфейсом более высокого уровня.

Пользовательские интерфейсы файловых систем



1. представление **файла как последовательность записей**.
Каждая запись - это последовательность байтов постоянного или переменного размера. Записи можно читать или записывать последовательно или позиционировать файл на запись с указанным номером.
2. представление **файла как последовательность байтов**.
Из файла можно прочитать указанное число байтов либо начиная с его начала, либо предварительно произведя его позиционирование на байт с указанным номером. Аналогично, можно записать указанное число байтов в конец файла, либо предварительно произведя позиционирование файла.

Именование файлов. КаталОг.
Классификация файловых систем по способу именования файлов

Современные файловые системы поддерживают многоуровневое именование файлов за счет поддержания во внешней памяти дополнительных файлов со специальной структурой - **каталОгов**.

Полное (квалифицированное) имя файла состоит из цепочки имен всех родительских каталогов начиная с корневого уровня и заканчивая собственным именем файла.

Изолированные файловые системы
Каждый архив файлов (полное дерево каталогов) целиком располагался на одном логическом диске, т.е. устройстве представляемом с помощью средств операционной системы как отдельный диск. Полное имя файла начинается с имени дискового устройства.

Полностью централизованные файловые системы
Вся совокупность каталогов и файлов представляется как единое дерево, физически распределенное по всем внешним запоминающим устройствам (операционная система Multics).

Компромиссное решение применено в файловых системах ОС UNIX.
На базовом уровне в этих файловых системах поддерживаются изолированные архивы файлов. Один из этих архивов объявляется корневой файловой системой. После запуска системы можно "смонтировать" корневую файловую систему и остальные изолированные файловые архивы в одну общую файловую систему.

Защита файлов от несанкционированного доступа (авторизация доступа)

Мандатный способ защиты
По отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю.
Каждый пользователь имеет отдельный **мандат** для работы с каждым файлом.

Дискреционный способ защиты
Каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: собственный идентификатор и идентификатор группы, к которой относится пользователь.
Этими же идентификаторами снабжается каждый процесс, запущенный от имени данного пользователя и имеющий возможность обращаться к файловой системе.
Каждый файл системы хранит полный идентификатор пользователя, который создал этот файл, и информацию о том, какие действия с файлом может производить он сам, какие действия с файлом доступны для остальных пользователей той же группы и что могут делать с файлом пользователи других групп.
Для каждого файла контролируется возможность выполнения трех действий: чтение, запись и выполнение.

Достоинства
- компактность (два целых числа для представления идентификаторов и шкала из 9 бит для характеристики возможных действий),
- скорость проверки (требуется небольшое количество действий)

Недостатки
Предопределенный ограниченный набор проверяемых действий

Обеспечение многопользовательского доступа

Контроль совместимости режима при открытии файла

Области разумного применения файлов

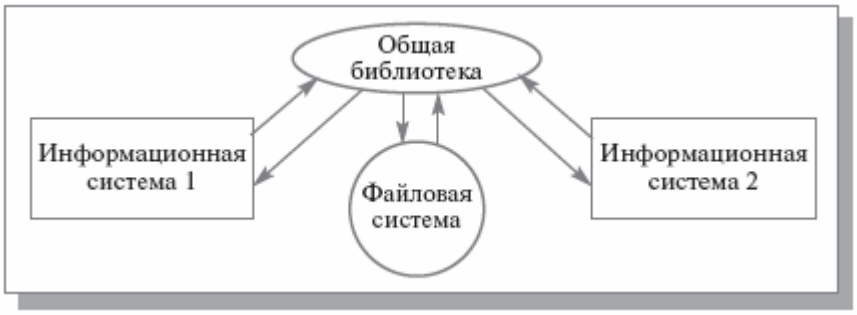
Цепочка связей между программными компонентами при написании программы

Файловые системы обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам.

Потребности информационных систем

Структура типовой информационной системы на начальном этапе использования вычислительной техники

Возможная схема интеграции сложных информационных систем.
Попытка создания первых прототипов СУБД
(две информационные системы с общей библиотекой работы с файловой системой)



Система учета служащих некоторой организации

Система должна поддерживать возможность получения следующих данных:
Для «служащего»:

- номера удостоверения по полному имени служащего (для простоты мы полагаем, что имена всех служащих различны);
- полного имени по номеру удостоверения;
- информации о соответствии занимаемой должности, и размере зарплаты.

Для «отдела»:

- имени руководителя отдела;
- общей численности отдела;
- общей суммы зарплаты служащих отдела, среднего размера зарплаты и т. д.

Для «организации»:

- списка служащих по отделам;

Кроме того, система обеспечивать средства поддержки:

- перевода служащего из одного отдела в другой;
- приема на работу новых и увольнения работающих служащих.

Необходимые структуры данных

Структура файла СЛУЖАЩИЕ (случай одного файла)

Пример файла СЛУЖАЩИЕ

Иванов	33	Да	100000	10	Астахов
Астахов	22	Да	200000	10	Астахов
Яковлев	11	Нет	30 000	20	Яковлев

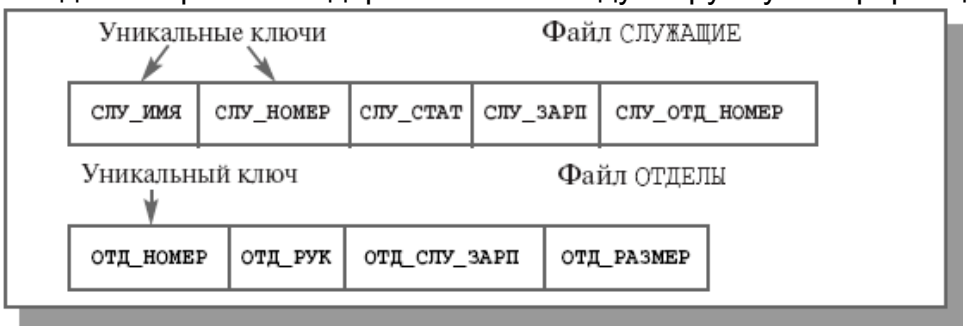
Вспомогательные **индексы** ускоряющие доступ

СЛУ_ИМЯ	СЛУ_НОМЕР	СЛУ_ОТД_НОМЕР
1	2	0
0	1	1
2	0	2

Сложности:

- требуется создание достаточно сложной надстройки для многоключевого доступа к файлам;
- возникает существенная избыточность данных (для каждого служащего повторяется имя руководителя его отдела);
- требуется выполнение массовой выборки и вычислений для получения суммарной информации об отделах.

Структура файла СЛУЖАЩИЕ и ОТДЕЛЫ (случай двух файлов)
Каждый из файлов содержит только не дублируемую информацию.



Минимизированы динамические вычислениях суммарной информации по отделам.
Сложности:

- система управления данными должна знать об их структуре,
- система управления данными должна поддерживать согласованность данных.

Обеспечение Согласованности(Целостности) данных

1. Если в файле СЛУЖАЩИЕ содержится запись со значением поля СЛУ_ОТД_НОМЕР, равным N, то и в файле ОТДЕЛЫ должна содержаться запись со значением поля ОТД_НОМЕР, также равным N
$\forall$ tr $\in$ Bt(СЛУЖАЩИЕ) $\Rightarrow \exists$ ts $\in$ Bs(ОТДЕЛЫ) : tr.СЛУ_ОТД_НОМЕР = ts.ОТД_НОМЕР

2. Если в файле ОТДЕЛЫ содержится запись со значением поля ОТД_РУК, равным M, то и в файле СЛУЖАЩИЕ должна содержаться запись со значением поля СЛУ_НОМЕР, также равным M
$\forall$ ts $\in$ Bs(ОТДЕЛЫ) $\Rightarrow \exists$ tr $\in$ Br(СЛУЖАЩИЕ) : ts.ОТД_РУК = tr.СЛУ_НОМЕР

(1) и (2) **ограничения ссылочной целостности**.

3. При любом корректном состоянии информационной системы значение поля ОТД_РАЗМЕР любой записи отд_k файла ОТДЕЛЫ должно быть равно числу всех тех записей файла СЛУЖАЩИЕ, в которых значение поля СЛУ_ОТД_НОМЕР совпадает со значением поля ОТД_НОМЕР записи отд_k

4. При любом корректном состоянии информационной системы значение поля ОТД_СЛУ_ЗАРП любой записи отд_k файла ОТДЕЛЫ должно быть равно сумме значений поля СЛУ_ЗАРП всех тех записей файла СЛУЖАЩИЕ, в которых значение поля СЛУ_ОТД_НОМЕР совпадает со значением поля ОТД_НОМЕР записи отд_k.

(3) и (4) **ограничения целостности общего вида**

Поддержка согласованности данных – **основной признак СУБД**.
Информацию, описывающую структуру хранимых данных и накладываемые на них ограничения целостности принято называть **метаданными**.

Формулировка запросов к системе

Наиболее естественный способ получения информации – формулировка запроса на некотором языке максимально близком пользователям.
Такие языки называются **языками запросов к базам данных**.
Получения общей численности отдела, в котором работает Иванов Иван Иванович

```
SELECT ОТД_РАЗМЕР
  FROM СЛУЖАЩИЕ, ОТДЕЛЫ
  WHERE СЛУ_ИМЯ = 'ИВАНОВ ИВАН ИВАНОВИЧ' AND
    СЛУ_ОТД_НОМЕР = ОТД_НОМЕР;
```

Получение списка служащих, не соответствующих занимаемой должности

```
SELECT СЛУ_ИМЯ, СЛУ_НОМЕР
  FROM СЛУЖАЩИЕ
  WHERE СЛУ_СТАТ = "НЕТ";
```

Запросы носят **декларативный** характер

Поддержка надежности хранения данных (восстановление после сбоев)

В **файловой системе** сбой при добавлении служащего может вызвать рассогласованное состояние файлов СЛУЖАЩИЕ и ОТДЕЛЫ
Будет нарушено правило (3) (численность), вероятно правило (4) (зарплата), и возможно правило (1) (номер отдела)).

В процессе восстановления система должна это обнаружить и исправить.
Системы управления файлами для этого мало пригодны.

СУБД берут заботу о поддержании целостности данных при различных сбоях системы на себя, поддерживая **транзакционное управление и журнализацию изменений базы данных**.

Поддержка параллельной многопользовательской работы

В **файловых системах** любая операция одного из пользователей по изменению данных будет приводить к полной блокировки чтения информации из базы данных всеми пользователями.

В **СУБД** реализуются механизмы, обеспечивающие гораздо более тонкую синхронизацию параллельного доступа к данным, на уровне отдельных записей.

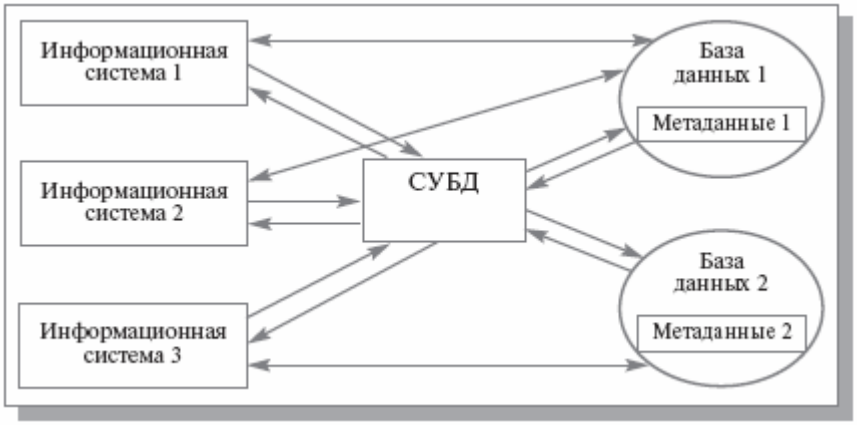
СУБД как независимый системный компонент

СУБД в составе информационной системы

Дефекты:

1. СУБД должны располагаться вне информационной системы.
2. Метаданные должны располагаться вместе с данными вне информационной системы.

Типовая структура использования современной СУБД



СУБД

Потребности информационных систем, которые не покрываются возможностями систем управления файлами:

- поддержание логически согласованного набора файлов;
- обеспечение языка описания и манипулирования данными;
- восстановление информации после разного рода сбоев;
- эффективная параллельная работа нескольких пользователей.

СУБД – система управления данными выполняющая следующий набор функций:

- управление **логически согласованными данными** в долговременной памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД.

Основные свойства транзакций **АСИД(ACID)**:

- **Атомарность/atomicity**. Транзакция выполняется как атомарная операция - либо выполняется вся транзакция целиком, либо она целиком не выполняется.
- **Согласованность/consistency**. Транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние. Внутри транзакции согласованность базы данных может нарушаться.
- **Изоляция/isolation**. Транзакции разных пользователей не должны мешать друг другу (например, как если бы они выполнялись строго по очереди).
- **Долговечность/durability**. Если транзакция выполнена, то результаты ее работы должны сохраниться в базе данных, даже если в следующий момент произойдет сбой системы.

Сериализация транзакций (сериальный план выполнения смеси транзакций) - такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения.

Надежность СУБД

Под **надежностью** СУБД понимают ее способность восстановить последнее согласованное состояние БД после любого сбоя.

Классификация сбоев по типу происхождения:
- **аппаратные**,
- **программные**.

Классификация сбоев по степени результирующего воздействия на БД:
- **мягкие**,
- **жесткие**.

Поддержание надежности хранения данных в БД достигается **избыточностью** хранимых данных за счет **ведения журнала изменений БД**.

Журнализация

Журнал - особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в который поступают записи обо всех изменениях основной части БД.

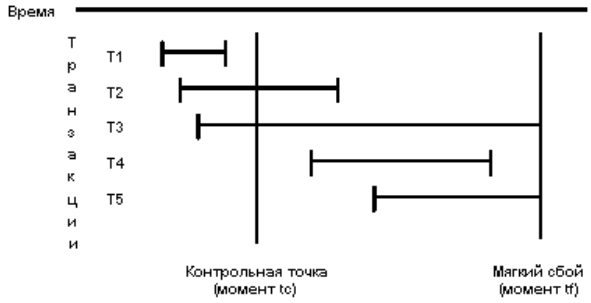
Уровни журнализация БД:

- **логический**, когда запись в журнале соответствует некоторой логической операции изменения БД
  (например, операции удаления строки из таблицы реляционной БД),
- **физический**, когда запись в журнале соответствует минимальной внутренней операции модификации страницы внешней памяти.
  (например, изменение сегмента внешней памяти в таком то интервале адресов)

Основная идея журнализации это использование стратегии "**упреждающей**" записи в журнал (протокола **WAL (Write Ahead Log)** – первичная запись лога).

Способы использования журнала в СУБД:

- индивидуальный откат транзакции (ROLLBACK),
- восстановление БД после мягкого сбоя,
- восстановление БД после жесткого сбоя.



Поддержка языков БД

SDL - Schema Definition Language
DML - Data Manipulation Language
SQL - Structured Query Language
Единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных.

Типовая организация современной СУБД

Современная реляционная СУБД включает следующие компоненты:

- внутреннюю часть - ядро СУБД (часто его называют Data Base Engine),
- компилятор языка БД (обычно SQL),
- подсистему поддержки времени выполнения,
- набор утилит.

Ранние подходы к организации БД

Общие характеристик ранних систем:

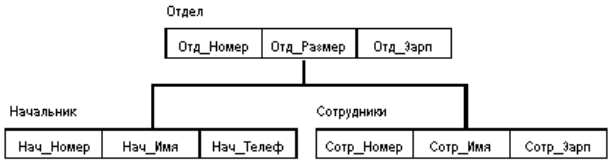
- Все ранние системы не основывались на каких-либо абстрактных моделях и не имели какой-либо математической модели.
- В ранних системах доступ к БД производился на уровне записей.
- Вся оптимизация доступа к БД осуществлялась пользователем.
- Большинство ранних систем было впоследствии оснащено "реляционными" интерфейсами.

Иерархические системы

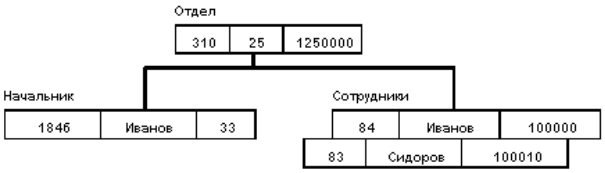
Структура иерархической БД

Упорядоченный набор нескольких экземпляров одного типа дерева.
Тип дерева - один "корневой" тип записи и упорядоченный набор из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева).

Пример схемы БД:



Пример базы данных:



Узел дерева - несколько упорядоченных записей, каждая запись может являться корнем своего собственного поддерева.

Манипулирование данными

- Найти указанное дерево БД (например, отдел 310);
- Перейти от одного дерева к другому;
- Перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику);
- Перейти от одной записи к другой в порядке обхода иерархии;
- Вставить новую запись в указанную позицию;
- Удалить/Изменить текущую запись.

Ограничения целостности

Никакой потомок не может существовать без своего родителя.

Сетевые системы

Структура сетевой БД

Основана на произвольных ориентированных графах и включает набора экземпляров каждого типа записи и набора экземпляров каждого типа связи.

Пример схемы БД:

Манипулирование данными

- Поиск конкретной записи в наборе однотипных записей;
- Перейти от предка к первому потомку по некоторой связи;
- Перейти к следующему потомку в некоторой связи;
- Перейти от потомка к предку по некоторой связи;
- Создать/Уничтожить/Модифицировать запись;
- Включить в связь/Исключить из связи/Переставить в другую связь

Ограничения целостности

в некоторых реализациях поддерживалась целостность по ссылкам.

Системы, основанные на инвертированных списках

Структуры данных

Двумерные таблицы, состоящие из строк – записей и столбцов – атрибутов.
Строки упорядочены системой в некоторой физической последовательности.
Пример БД

№	СЛУ_ИМЯ	СЛУ_НОМЕР	СЛУ_СТАТУС	СЛУ_ЗАРЛАТА	СЛУ_ОТД_N	СЛУ_РУК
0	Иванов	33	Да	100000	10	Астахов
1	Астахов	22	Да	200000	10	Астахов
2	Яковлев	11	Нет	300000	20	Яковлев
3	Петров	50	Да	200000	20	Яковлев

Индексы ускоряющие доступ

СЛУ_ИМЯ	СЛУ_НОМЕР	СЛУ_ОТД_НОМЕР
1	2	0
0	1	1
3	0	3
2	3	2

Манипулирование данными

- прямые поисковые операторы;
- операторы, находящие запись в терминах относительной позиции от предыдущей записи.
- операторы над адресуемыми записями

Ограничения целостности

В некоторых системах поддерживались ограничения уникальности значений некоторых атрибутов.

Достоинства и недостатки ранних дореляционных СУБД

Достоинства

- развитые средства управления данными во внешней памяти на низком уровне, и как следствие возможность разработки высокоэффективных прикладных систем.

Недостатки

- необходимость знания прикладной системой физической организации БД;
- отсутствие или сильная ограниченность правил определения целостности БД.

Реляционная модель данных

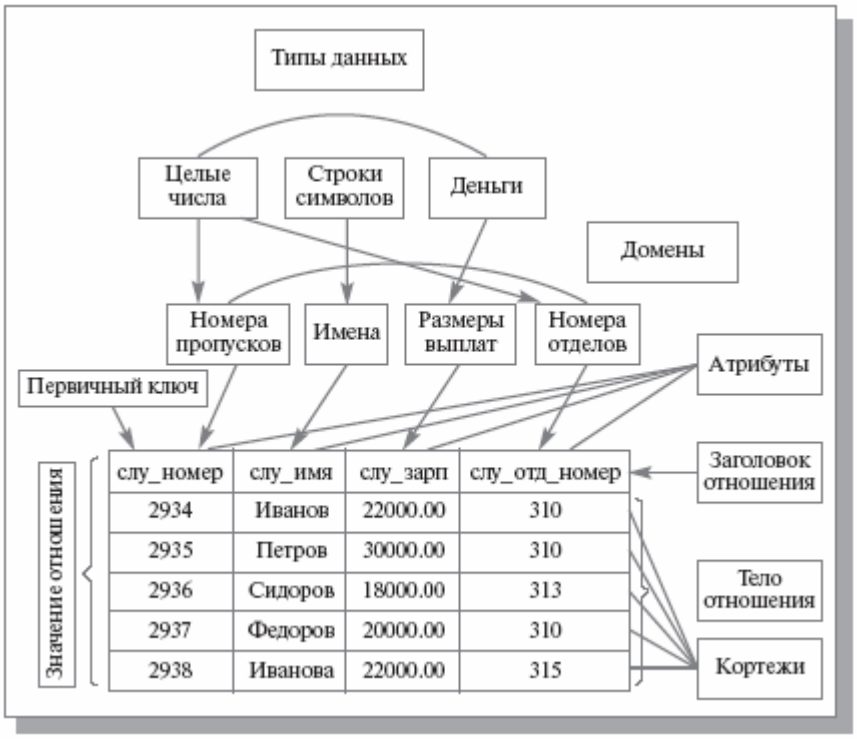
1969 г. Эдгар Кодд
«Реляционная модель данных для больших совместно используемых банков данных» [*http://citforum.ru*]

Достоинства реляционного подхода:

- основывается на небольшом числе интуитивно понятных абстракций, имеющих четкие формальные определения;
- позволяет моделировать данные из различных предметных областей;
- теоретическим базисом является аппарат теории множеств и математической логики;
- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации базы данных.

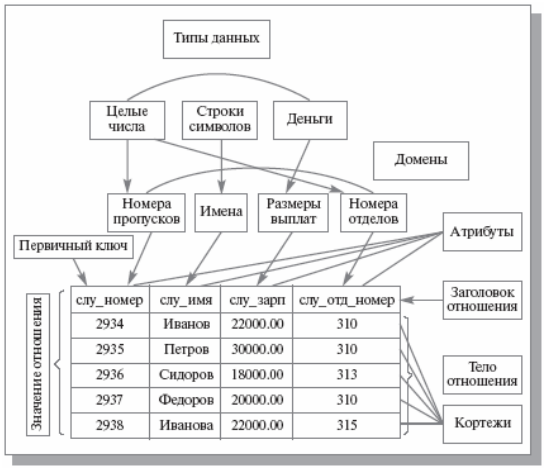
Базовые понятия реляционной модели данных

Тип данных, домен, атрибут, кортеж, отношение, первичный ключ



Тип данных

- определение множества значений данного типа;
- определение набора операций, применимых к значениям типа;
- определение способа внешнего представления значений типа (литералов).



Стандартные типы данных современных реляционных БД включают:

- символьные,
- числовые (точные и приблизительные),
- специальные «числовые» (деньги),
- специальные «темпоральные» (дата, время, временной интервал).
- битовые строки,

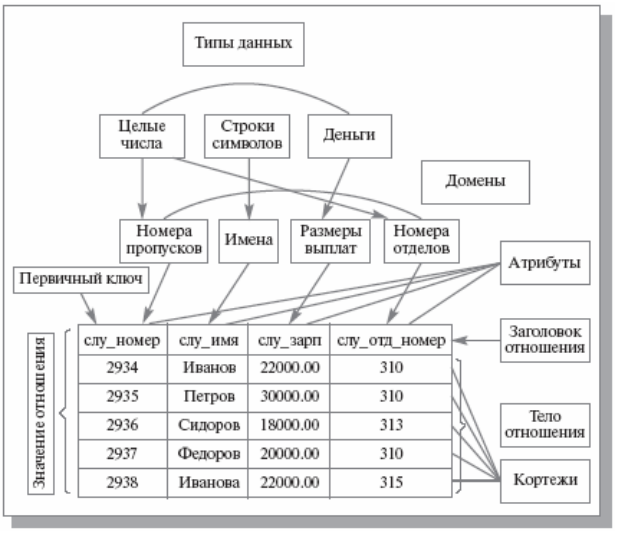
\+ возможность определять пользовательские типы данных

Домен

Базовый тип данных
Логическое выражение (ограничение домена).
Уникальное имя

Схема отношения
Hr = { <ИМЯ_АТРИБУТА, ИМЯ_ДОМЕНА_ИЛИ_БАЗОВОГО_ТИПА>}

Отношение СЛУЖАЩИЕ :
<слу_номер, номера_пропусков>,
<слу_имя, имена>,
<слу_зарп, размеры_выплат>,
<слу_отд_номер, номера_отделов>



Кортеж
tr(Hr) = {<ИМЯ_АТРИБУТА, ЗНАЧЕНИЕ>}

Тело отношения
{ tr(Hr) }

Отношение
A(Hr) = Hr, { tr(Hr) }

Реляционная база данных – набор отношений

Фундаментальные свойства отношений
Отсутствие кортежей-дубликатов

Первичный ключ - такое подмножество S1 из множества атрибутов схемы отношения, что в любое время значение первичного ключа в любом кортеже тела отношения отличается от значения первичного ключа в любом другом кортеже тела этого отношения, а никакое собственное подмножество S2 из множества S1 этим свойством не обладает.

Первичный ключ **один** и задается проектировщиком базы данных.

Остальные минимальные наборы атрибутов, обладающие свойством уникальности, принято называются **возможными ключами**.

Составной ключ – ключ состоящий более чем из одного атрибута

Фундаментальные свойства отношений
Отсутствие упорядоченности кортежей

Положительные аспекты:

- определение тела отношения как «*множества*» кортежей, облегчает построение механизма манипулирования реляционными данными – реляционной алгебры и исчисления;
- хранение кортежей в упорядоченном виде в условиях интенсивно обновляемой базы данных гораздо сложнее технически и поддержка упорядоченности влечет за собой дополнительные накладные расходы.

Фундаментальные свойства отношений
Отсутствие упорядоченности атрибутов

```
struct  {
        int a;
        char b;
        int c;  } d;
```

способы доступа к атрибуту «b»

1. *(&d + sizeof(integer))
2. d.b

Положительные аспекты:

- СУБД сама принимает решение о том, в каком физическом порядке следует хранить значения атрибутов кортежей;
- облегчается выполнение операции модификации схем существующих отношений при эволюции схем баз данных.

Фундаментальные свойства отношений
Атомарность значений атрибутов, первая нормальная форма отношения

Атомарность значения всех атрибутов отношения следует из определения домена как подмножества значений простого типа данных (среди значений домена не могут содержаться множества или отношения).
Принято говорить, что в реляционных базах данных допускаются только нормализованные отношения, или отношения, представленные в **первой нормальной форме**, то есть удовлетворяют свойству атомарности.

Ненормализованное отношение

НОМЕР_ОТДЕЛА	ОТДЕЛ		
	СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
310	2934	Иванов	22000.00
	2935	Петров	30000.00
	2937	Федоров	20000.00
313	2936	Сидоров	18000.00
315	2938	Иванова	22000.00

Нормализованный вариант

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

Добавление кортежа:
{<СЛУ_ИМЯ,Иванов>, <СЛУ_НОМЕР,3000>, <СЛУ_ЗАРП,25000.00><СЛУ_ОТД_НОМЕР,320>}
{<СЛУ_ИМЯ,Иванов>, <СЛУ_НОМЕР,3000>, <СЛУ_ЗАРП,25000.00><СЛУ_ОТД_НОМЕР,310>}

Реляционная модель данных

Реляционная модель состоит из трех частей:

- структурной;
- манипуляционной;
- целостной.

Целостная часть реляционной модели

Первое требование - Требование целостности сущности (**entity integrity**)
У любого отношения должен существовать первичный ключ, и никакое значение первичного ключа в кортежах отношения не должно содержать неопределенных значений.

Неопределенное значение (**NULL**).
Не принадлежит никакому типу данных и может присутствовать среди значений любого атрибута, определенного на любом типе данных (если это явно не запрещено при определении атрибута).

a	op	NULL	= NULL
NULL	op	a	= NULL
a	lop	NULL	= unknown
NULL	lop	a	= unknown

a – это значение некоторого типа данных или NULL,
op –двуместная «арифметическая» операция этого типа данных (+-/…),
lop – «логическая» операция сравнения значений этого типа (<>=!=…),

«**unknown**» – это третье значение логического, или булевского, типа
NOT unknown = unknown

false	AND	unknown	= false
true	OR	unknown	= true
true	AND	unknown	= unknown
false	OR	unknown	= unknown

Целостная часть реляционной модели

Внешний ключ (foreign key) – атрибут некоторого отношения, значения которого однозначно характеризуют сущности, представленные кортежами некоторого другого отношения, т. е. задают значения их первичного ключа.

Второе требование - **Требование ссылочной целостности** (**referential integrity**)
(Требования целостности внешнего ключа)
Для каждого значения внешнего ключа, появляющегося в ссылающемся отношения, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть полностью неопределенным, т.е. ни на что не указывать.

Способы поддержания ссылочной целостности

- ограничить (restrict);
- установить в NULL (set NULL) [(set default)];
- каскадное удаление (cascade).

Некоторые отличия реляционной и SQL моделей данных

- в SQL есть возможность обращения к атрибутам отношения по номерам;
- в классической реляционной модели в возможных ключах не допускаются неопределенные значения (в SQL допускаются);
- в языке SQL допускается несколько вариантов определения внешнего ключа, из которых только один полностью соответствует классическому подходу;
- в языке SQL допускается определение таблиц без первичного и возможных ключей.

Набор операций реляционной алгебры Кодда:

Теоретико-множественные операции:
- объединения отношений (UNION)(1);
- пересечения отношений (INTERSECT)(2);
- взятия разности отношений (MINUS)(1);
- взятия декартова произведения отношений (TIMES)(2).

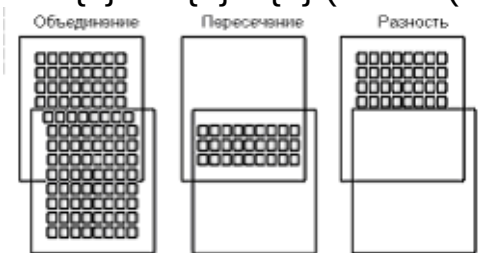
Специальные реляционные операции включают:
- ограничение отношения (WHERE)(3);
- проекцию отношения (PROJECT)(3);
- соединение отношений (JOIN)(2);
- деление отношений (DIVIDE BY)(2).

Дополнительные операции:
- операция присваивания (:=);
- операция переименования атрибутов (RENAME)(4).

Приоритет операций:
RENAME >=
WHERE = PROJECT >=
TIMES = JOIN = INTERSECT = DIVIDE BY >=
UNION = MINUS

Теоретико-множественные операции над множествами A{a} и B{b}

- **UNION (ОБЪЕДИНЕНИЕ)**
  C{c} = A{a}∪B{b} (∀c =>(∃a:a=c || ∃b:b=c), ∀a => ∃c:c=a, ∀b => ∃c:c=b)
- **INTERSECT (ПЕРЕСЕЧЕНИЕ)**
  C{c} = A{a}∩B{b} (∀c =>(∃a:a=c && ∃b:b=c), ∀a,b:a=b =>∃c:c=a=b)
- **MINUS (РАЗНОСТЬ)**
  C{c} = A{a}-B{b} (∀c =>(∃a:a=c && ~∃b:b=c), ∀a:~∃b:a=b => ∃c:c=a)



Отношения совместимые по объединению:

Hr(A)==Hr(b).

Отношения «почти» совместимые по объединению

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
37	Федоров	20000	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 UNION СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313
37	Федоров	20000	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 INTERSECT СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 MINUS СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
36	Сидоров	18000	313

UNION == служащих, участвующих в каком-либо проекте.
INTERSECT == служащих, одновременно участвующих в двух проектах.
MINUS == служащих, участвующих только в первом проекте.
Избыточность теоретико-множественных операций алгебры Кодда
A INTERSECT B = A MINUS (A MINUS B) = B MINUS (B MINUS A)

Операция взятия декартова произведения TIMES

C{<c1,c2>}= A{a} TIMES B{b}

(∀c1€A,c2€B=>∃<c1,c2>€C, ∀<c1,c2>€C=>(∃c1€A,c2€B))

Совместимость по взятию расширенного декартова

Hr(A)∩ Hr(B) ==∅

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

ПРОЕКТЫ

ПР_НАЗВАНИЕ	ПР_РУК
ПРОЕКТ1	Иванов
ПРОЕКТ2	Федоров

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES ПРОЕКТЫ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N	ПР_НАЗВАНИЕ	ПР_РУК
34	Иванов	22000	310	ПРОЕКТ1	Иванов
35	Петров	30000	310	ПРОЕКТ1	Иванов
36	Сидоров	18000	313	ПРОЕКТ1	Иванов
34	Иванов	22000	310	ПРОЕКТ2	Федоров
35	Петров	30000	310	ПРОЕКТ2	Федоров
36	Сидоров	18000	313	ПРОЕКТ2	Федоров

Все операции, кроме взятия разности, являются коммутативными b ассоциативными

A op B = B op A, (A op B) op C = A op (B op C),

Специальные реляционные операции

Операция ограничения (WHERE)

A WHERE comp, где

A – ограничиваемое отношение,

comp – условие, составленное из «простых условий» и операций AND, OR, NOT.

Простое условие:

- (a comp-op b),
- (a comp-op const),

где

a и b – имена атрибутов ограничиваемого отношения, определенных на одном и том же домене или базовом типе данных, для которого определена операция сравнения comp_op;

const – литерально заданная константа того же домена или типа, что и атрибут a;

comp-op – операции «=», «!=», «>», «>=», «<», «<=».

Способы раскрытия логических условий:

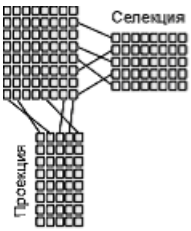
A WHERE (comp1 **AND** comp2) == (A WHERE comp1) **INTERSECT** (A WHERE comp2);
A WHERE (comp1 **OR** comp2) == (A WHERE comp1) **UNION** (A WHERE comp2);
A WHERE **NOT** comp1 == A **MINUS** (A WHERE comp1).

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 WHERE (СЛУ_ЗАРП > 20000)

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310



Операция взятия проекции (PROJECT A {a1, a2, ..., an})
PROJECT A{a1,.,an} = A'{a'} :
∀a{<a1:v1>,.<am,vm>}€A => ∃a'{<a1:v1>,.<an,vn>}€A',
∀a'{<a1:v1>,.<an,vn>}€A' => ∃a{<a1:v1>,.<am,vm>}€A, {a1,.an} € {a1,.am}

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

PROJECT СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 {СЛУ_ОТД_N}

СЛУ_ОТД_N
310
313

Операция соединения отношений (JOIN)

A JOINT B WHERE comp == (A TIMES B) WHERE comp

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРОЕКТЫ

ПРО_N	ПРО_РУК	ПРО_ЗАРП
1	Иванов	22000
2	Федоров	22000

СЛУЖАЩИЕ JOIN (ПРОЕКТЫ RENAME (ПРО_N,ПРО_N1))WHERE(СЛУ_ЗАРП>ПРО_ЗАРП)

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_N1	ПРО_РУК	ПРО_ЗАРП
35	Петров	30000	1	1	Иванов	22000
35	Петров	30000	2	1	Иванов	22000
35	Петров	30000	1	2	Федоров	22000
35	Петров	30000	2	2	Федоров	22000

Действительное соединение если условие comp может быть преобразована к виду **comp1 AND (a comp-op b)**, где a и b – имена атрибутов разных отношений-операндов.

Эквисоединение (**EQUIJOIN**) соединение с условием вида (a = b), где a и b – атрибуты разных операндов соединения.

Естественное соединение (**NATURAL JOIN**)
Пусть заданы отношения A{ac} и B{cb}.
A **NATURAL JOIN** B == PROJECT (A JOIN (B RENAME (c,c1)) WHERE (c==c1)) {abc}

СЛУЖАЩИЕ

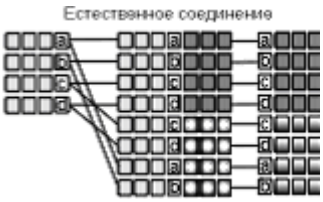
СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРОЕКТЫ

ПРО_N	ПРО_РУК	ПРО_ЗАРП
1	Иванов	22000
2	Федоров	22000

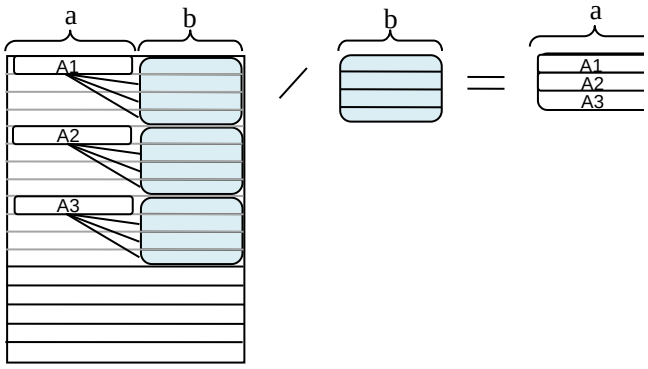
СЛУЖАЩИЕ NATURAL JOIN ПРОЕКТЫ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК	ПРО_ЗАРП
34	Иванов	22000	1	Иванов	22000
35	Петров	30000	1	Иванов	22000
36	Сидоров	18000	1	Иванов	22000
34	Иванов	22000	2	Федоров	22000
35	Петров	30000	2	Федоров	22000
37	Федоров	20000	2	Федоров	22000



Операция деления отношений
(A DIVIDE BY B)

A {a1, a2, ..., an, b1, b2, ..., bm}, B {b1, b2, ..., bm}
{aj} == a, {bj} == b.



СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

НОМЕРА_ПРОЕКТОВ

ПРО_N
1
2

СЛУЖАЩИЕ DIVIDE BY НОМЕРА_ПРОЕКТОВ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП
34	Иванов	22000
35	Петров	30000

Алгебра Кодда - итог:

Теоретико-множественные операции:
- UNION, INTERSECT, MINUS, TIMES.

Специальные реляционные операции включают:
- WHERE, PROJECT, JOIN, DIVIDE BY.

Дополнительные операции:
- операция присваивания (:=) и переименования RENAME.

- рассмотренная версия алгебры – модификация Кристофера Дейта с RENAME,
  (оригинальная с точечной нотацией ОТНОШЕНИЕ.ИМЯ_АТРИБУТА)
- избыточна
- практична (отражается в операторах SQL)

Алгебра A

в конце 90-х годов предложена Кристофером Дейтом и Хью Дарвеном
настоящая алгебра в математическом смысле – отсутствуют ограничения на операнды.

Алгебра А Дейта и Дарвена

Итак, пусть
r – отношение,
A – имя атрибута отношения r,
T – имя соответствующего типа (т. е. типа или домена атрибута A),
v – значение типа T.
Тогда:

- **заголовком Hr** отношения r называется множество **атрибутов**, пар упорядоченного вида <A, T>. По определению никакие два атрибута в этом множестве не могут содержать одно и то же имя атрибута A;
- **кортеж tr**, соответствующий заголовку Hr, – это множество триплетов упорядоченного вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr;
- **тело Br** отношения r – это множество кортежей tr. При этом, как вы понимаете, могут существовать такие кортежи tr, которые соответствуют Hr, но не входят в Br.

Базовый (НЕ ОКОНЧАТЕЛЬНЫЙ) набор операций алгебры А:

- операции реляционного дополнения **<NOT>**,
- удаления атрибута **<REMOVE>**,
- переименования атрибута <**RENAME**>,
- реляционной конъюнкции **<AND>** и
- реляционной дизъюнкции <**OR**>.

Операция реляционного дополнения

Пусть **s** обозначает результат операции **<NOT> r**.
Тогда:
Hs = Hr;
Bs = {ts : exists tr (tr $\notin$ Br and ts = tr) }

НОМЕРА_ПРОЕКТОВ

N_ПРОЕКТА	ИМЯ_РУК
1	Иванов
2	Петров

Домен Номера Проектов: {1,2,3}
Домен Имена Руководителей : {Иванов,Петров}

<NOT> НОМЕРА_ПРОЕКТОВ

N_ПРОЕКТА	ИМЯ_РУК
1	Петров
2	Иванов
3	Иванов
3	Петров

Операция удаления атрибута

s == **r <REMOVE> A**.

∃ T : <A, T> ∈ Hr

Hs = Hr minus {<A, T>};

Bs = {ts : exists tr exists v (tr ∈ Br and v ∈ T and <A,T,v> ∈ tr and ts = tr minus {<A,T,v>})};

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

(СЛУЖАЩИЕ REMOVE ПРО_НОМ) REMOVE СЛУ_ЗАРП

СЛУ_N	СЛУ_ИМЯ
34	Иванов
35	Петров
36	Сидоров
37	Федоров

Операция переименования

s == **r <RENAME> (A, B)**.

$\exists$ T : <A, T> $\in$ Hr, и чтобы <B, T> $\notin$ Hr.
Тогда:
Hs = (Hr minus {<A, T>}) union {<B, T>};
Bs = {ts : exists tr exists v (tr $\in$ Br and v $\in$ T and <A, T, v> $\in$ tr and
ts = (tr minus {<A,T,v}) union {<B,T,v>})};

Операция реляционной конъюнкции s == r1 <AND> r2

$\forall$ A : <A, T1> $\in$ Hr1 AND <A, T2> $\in$ Hr2 : T1=T2

одноименные атрибуты должны быть определены на одном домене

Hs = Hr1 union Hr2;

Bs = { ts : $\exists$ tr1(Hr1), $\exists$ tr2(Hr2) : (tr1 $\in$ Br1 and tr2 $\in$ Br2) => ts = (tr1 union tr2) };

Теоретико-множественная операция **union** для кортежа

1. {<A1,T1,V1>} union {<A2,T2,V2>} = {<A1,T1,V1>,<A2,T2,V2>}, (A1!=A2)
2. {<A,T,V>} union {<A,T,V>} = {<A,T,V>} (V1 == V2)
3. {<A,T,V1>} union {<A,T,V2>} = $\varnothing$ (V1 != V2)

Поведение **<AND>** определяется схемами ее операндов:

1. Hr1 $\cap$ Hr2 = $\varnothing$ (пример 1.) <AND> = **TIMES**
2. Hr1 = Hr2 (пример 2. и 3.) <AND> = **INTERSECT**
3. Hr1 $\cap$ Hr2 $\neq$ $\varnothing$ (пример 1. 2. 3.) <AND> = **NATURAL JOIN**

<AND>
случай Hr1 ∩ Hr2 = ∅
Hs = Hr1 union Hr2;
Bs = { ts : ∃ tr1(Hr1), ∃ tr2(Hr2) : (tr1 ∈ Br1 and tr2 ∈ Br2) => ts = (tr1 union tr2) };

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

ПРОЕКТЫ

ПРО_N	ПРО_РУК
1	Иванов
2	Федоров

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <AND> ПРОЕКТЫ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N	ПРО_N	ПРО_РУК
34	Иванов	22000	310	1	Иванов
35	Петров	30000	310	1	Иванов
36	Сидоров	18000	313	1	Иванов
34	Иванов	22000	310	2	Федоров
35	Петров	30000	310	2	Федоров
36	Сидоров	18000	313	2	Федоров

Hr1 ∩ Hr2 = ∅ ⇒ <AND> == TIMES

<AND>
случай Hr1 = Hr2
Hs = Hr1 union Hr2;
Bs = { ts : ∃ tr1(Hr1), ∃ tr2(Hr2) : (tr1 ∈ Br1 and tr2 ∈ Br2) => ts = (tr1 union tr2) };

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
37	Федоров	20000	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <AND> СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310

Hr1 = Hr2 $\Rightarrow$ <AND> == INTERSECT

<AND>
случай Hr1 ∩ Hr2 ≠ ∅
Hs = Hr1 union Hr2;
Bs = { ts : ∃ tr1(Hr1), ∃ tr2(Hr2) : (tr1 ∈ Br1 and tr2 ∈ Br2) => ts = (tr1 union tr2) };

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРОЕКТЫ

ПРО_N	ПРО_РУК
1	Иванов
2	Федоров

СЛУЖАЩИЕ <AND> ПРОЕКТЫ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
35	Петров	30000	1	Иванов
36	Сидоров	18000	1	Иванов
34	Иванов	22000	2	Федоров
35	Петров	30000	2	Федоров
37	Федоров	20000	2	Федоров

Hr1 ∩ Hr2 ≠ ∅ ⇒ <AND> == NATURAL JOIN

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
35	Петров	30000	1	Иванов
36	Сидоров	18000	1	Иванов
34	Иванов	22000	2	Федоров
35	Петров	30000	2	Федоров
37	Федоров	20000	2	Федоров

ПРО_N	СЛУ_N	ПРО_РУК	СЛУ_ЗАРП	СЛУ_ИМЯ
2	34	Федоров	22000	Иванов
1	35	Иванов	30000	Петров
2	35	Федоров	30000	Петров
1	36	Иванов	18000	Сидоров
1	34	Иванов	22000	Иванов
2	37	Федоров	20000	Федоров

Операция реляционной дизъюнкции

s == **r1 <OR> r2**.

$\forall$ A : <A, T1> $\in$ Hr1 AND <A, T2> $\in$ Hr2 : T1=T2

одноименные атрибуты должны быть определены на одном домене

Hs = Hr1 union Hr2;

Bs = { ts : $\exists$ tr1(Hr1), $\exists$ tr2(Hr2): (tr1 $\in$ Br1 or tr2 $\in$ Br2) => ts = (tr1 union tr2) };

Поведение <OR> определяется схемами ее операндов:

1. Hr1 = Hr2 (пример 2. и 3.) <OR> = UNION

<OR>
случай Hr1 == Hr2

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
37	Федоров	20000	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <OR> СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313
37	Федоров	20000	315

<OR>
случай Hr1 ∩ Hr2 = ∅

ПРОЕКТЫ

ПРО_НАЗ	ПРО_РУК
Проект1	Иванов
Проект2	Федоров

НОМЕРА_ПРОЕКТОВ

ПРО_N
1
2

ПРО_НАЗ { Проект1, Проект2, Проект3 },
ПРО_РУК { Иванов, Федоров },

ПРО_N { 1,2,3 }

ПРОЕКТЫ <OR> НОМЕРА_ПРОЕКТОВ

ПРО_НАЗ	ПРО_РУК	ПРО_N
Проект1	Иванов	1
Проект2	Иванов	1
Проект3	Иванов	1
Проект1	Федоров	1
Проект2	Федоров	1
Проект3	Федоров	1
Проект1	Иванов	2
Проект2	Иванов	2
Проект3	Иванов	2
Проект1	Федоров	2
Проект2	Федоров	2
Проект3	Федоров	2
Проект1	Иванов	3
Проект2	Федоров	3

Полнота Алгебры А

Возможность выразить через ее операторы все операции алгебры Кодда.

Мы уже выяснили что:

- RENAME == <RENAME>,
- PROJECT == {<REMOVE>},
- UNION == <OR>,
- TIMES == <AND>,
- INTERSECT == <AND>,
- [NATURAL JOIN == <AND>,]

Остались операции:

- взятия разности (MINUS),
- ограничения (WHERE),
- соединения общего вида (JOIN),
- реляционного деления (DIVIDE BY).

Выражение операции взятия разности

Утверждается:

r1 **MINUS** r2 = r1 **<AND>** (**<NOT>** r2),
(для r1,r2 : Hr1 == Hr2)

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
37	Федоров	20000	315

<NOT> СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
35	Иванов	22000	310
36	Иванов	22000	310
…	…	…	…
34	Иванов	18000	310
34	Иванов	30000	310
…	…	…	…
36	**Сидоров**	**18000**	**313**
…	…	…	…

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <AND> (<NOT> СЛУЖАЩИЕ_В_ПРОЕКТЕ_2)

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
36	Сидоров	18000	313

Выражение операции ограничения (r WHERE comp)

r – отношение,
comp – условие, составленное из простых условий вида
(a comp-op b) или ***(a comp-op const)***,
a и b – имена атрибутов ограничиваемого отношения,
const – литерально заданная константа,
comp-op - операция сравнения «=», «!=», «>», «<», «>=», «<=».

Выражение операции (r WHERE a comp-op const) через <AND>

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310
35	Петров	30000	310
36	Сидоров	18000	313

СЛУ_ЗАРП : ВЫПЛАТЫ
{22000,30000,18000}

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 WHERE СЛУ_ЗАРП == 22000

ЗАРП_22000

СЛУ_ЗАРП
22000

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <AND> ЗАРП_22000

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
34	Иванов	22000	310

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 WHERE СЛУ_ЗАРП < 30000

ЗАРП_М_30000

СЛУ_ЗАРП
18000
22000

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 <AND> ЗАРП_М_30000

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_N
36	Сидоров	18000	313
34	Иванов	22000	310

Аналогичным образом выражаются операции ограничения с условиями «<», «<=»,«>=», «!=».

Выражение операции (r WHERE a comp-op b) через <AND>

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_РУК_N
34	Иванов	22000	34
35	Петров	30000	34
36	Сидоров	18000	34
37	Федоров	20000	37

СЛУ_N : НомераПропусков { 34,35,36,37,38 }

СЛУЖАЩИЕ WHERE СЛУ_N = СЛУ_РУК_N

СЛУ_РУК

СЛУ_N	СЛУ_РУК_N
34	34
35	35
36	36
37	37
38	38

СЛУЖАЩИЕ <AND> СЛУ_РУК

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_РУК_N
34	Иванов	22000	34
37	Федоров	20000	37

Утверждается, что результат данной операции совпадает с результатом следующего выражения алгебры А:
СЛУЖАЩИЕ <AND> ((((СЛУЖАЩИЕ <REMOVE> СЛУ_N) <REMOVE>СЛУ_ИМЯ) <REMOVE> СЛУ_ЗАРП) <RENAME> (СЛУ_РУК_N, СЛУ_N))
(Выражение WHERE (a = b) через <REMOVE>, <RENAME> и <AND>)

Выражение операции ограничения вида r WHERE (a > b) через <AND>

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ПРЕМ
34	Иванов	22000	18000
35	Петров	30000	22000
36	Сидоров	18000	20000
37	Федоров	20000	22000

СЛУ_ЗАРП, СЛУ_ПРЕМ : ВЫПЛАТЫ { 18000,20000,22000,30000 }

СЛУЖАЩИЕ WHERE СЛУ_ЗАРП > СЛУ_ПРЕМ

ЗАРП_БОЛЬШЕ_ПРЕМ

СЛУ_ЗАРП	СЛУ_ПРЕМ
20000	18000
22000	18000
30000	18000
22000	20000
30000	20000
30000	22000

СЛУЖАЩИЕ <AND> ЗАРП_БОЛЬШЕ_ПРЕМ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ПРЕМ
34	Иванов	22000	18000
35	Петров	30000	22000

Аналогичным образом выражаются операции ограничения с условиями «<», «!=», «<=», «>=».

| A_БОЛЬШЕ_B | = (n)(n-1)/2, где n – | соответствующего домена |.

Выражение операции соединения общего вида (A JOIN B WHERE comp)

Алгебра Кодда:
A JOIN B WHERE comp == (A TIMES B) WHERE comp

Алгебра **A**:

- выполнить одну или несколько операций <RENAME>, чтобы избавиться от общих имен атрибутов;
- выполнить над полученными отношениями операцию <AND>, производящую расширенное декартово произведение;
- выполнить над полученным отношением одну или несколько операций <AND> с отношениями-константами, чтобы должным образом ограничить его.

Выражение операции реляционного деления r1{A, B} DIVIDE BY r2{B}

(r1 PROJECT A) MINUS ((((r1 PROJECT A) TIMES r2) MINUS r1) PROJECT A)

(r1<REMOVE>B)<AND><NOT>(((r2<AND>(r1<REMOVE>B))<AND><NOT>r1) <REMOVE>B)

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	ПРО_N
34	Иванов	1
35	Петров	1
36	Сидоров	1
34	Иванов	2
35	Петров	2
37	Федоров	2

ПРОЕКТЫ

ПРО_N
1
2

X1 = СЛУЖАЩИЕ PROJECT A

СЛУ_N	СЛУ_ИМЯ
34	Иванов
35	Петров
36	Сидоров
37	Федоров

X2 = X1 TIMES ПРОЕКТЫ

СЛУ_N	СЛУ_ИМЯ	ПРО_N
34	Иванов	1
35	Петров	1
36	Сидоров	1
37	Федоров	1
34	Иванов	2
35	Петров	2
36	Сидоров	2
37	Федоров	2

X3 = X2 MINUS СЛУЖАЩИЕ

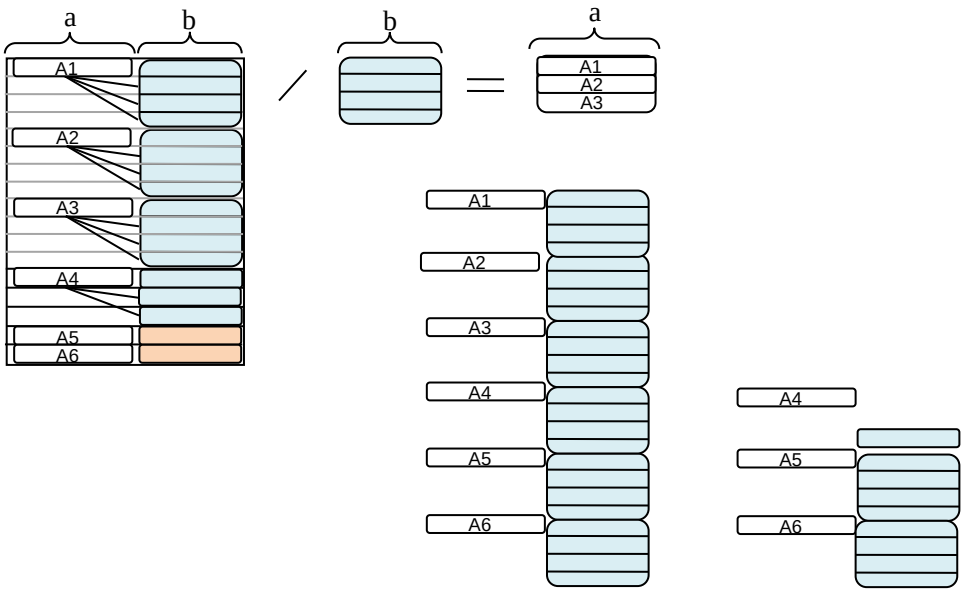
СЛУ_N	СЛУ_ИМЯ	ПРО_N
37	Федоров	1
36	Сидоров	2

X4 = X3 PROJECT A

СЛУ_N	СЛУ_ИМЯ
37	Федоров
36	Сидоров

СЛУЖАЩИЕ DIVIDE BY ПРОЕКТЫ = X1 MINUS X4

СЛУ_N	СЛУ_ИМЯ
34	Иванов
35	Петров


a
b
A1
A2
A3
A4
A5
A6
b
=
a
A1
A2
A3
A1
A2
A3
A4
A5
A6
A4
A5
A6

Избыточность Алгебры А
Избыточность операции <RENAME>

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРО_N : ПРОЕКТЫ {1,2,3}

ПРОN_СЛУПРОN

ПРО_N	СЛУ_ПРО_N
1	1
2	2
3	3

СЛУЖАЩИЕ <AND> ПРОN_СЛУПРОN

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	СЛУ_ПРО_N
34	Иванов	22000	1	1
35	Петров	30000	1	1
36	Сидоров	18000	1	1
34	Иванов	22000	2	2
35	Петров	30000	2	2
37	Федоров	20000	2	2

СЛУЖАЩИЕ<AND>ПРОN_СЛУПРОN<REMOVE>ПРО_N

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

Избыточность операций <AND> или <OR>

В математической логике набор {NOT, AND, OR} традиционен, но избыточен.
Верны тождества:
A AND B = NOT (NOT A OR NOT B)
A OR B = NOT (NOT A AND NOT B).

A <AND> B = <NOT> (<NOT> A <OR> <NOT> B)
A <OR> B = <NOT> (<NOT> A <AND> <NOT> B).

Одна из операций <AND> или <OR> может быть удалена.

Реляционные аналоги штриха Шеффера и стрелки Пирса

«штрих Шеффера» – sh (A, B) = NOT A OR NOT B,
«стрелка Пирса» – pi (A, B) = NOT A AND NOT B.

NOT A = sh (A, A)
A OR B = sh (NOT A, NOT B)
A AND B = NOT sh (A, B)

NOT A = pi (A, A)
A AND B = pi (NOT A, NOT B)
A OR B = NOT pi (A, B)

Реляционные аналоги:
штриха Шеффера (<sh> (r1, r2) = <NOT> r1 <OR> <NOT> r2)
и стрелки Пирса (<pi> (r1, r2) = <NOT> r1 <AND> <NOT> r2).

Формально мы можем свести набор операций Алгебры А к двум операциям:

- <sh> (или <pi>);
- <REMOVE>.

Базисом Алгебры А принято считать следующий набор операций:

- **<NOT>** реляционного отрицания (дополнения),
- **<AND> / <OR>** реляционной конъюнкции (или дизъюнкции),
- **<REMOVE>** удаления атрибута (проекции).

+ операция присваивания переменной отношения (**:=**).

Реляционное исчисление

СЛУЖАЩИЕ {СЛУ_N, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_N}
ПРОЕКТЫ {ПРО_N, ПРО_РУК_N, ПРО_ЗАРП}
Получить имена и номера служащих, которые являются руководителями проектов со средней заработной платой, превышающей 100000.

Выражение реляционной алгебры:
((СЛУЖАЩИЕ JOIN ПРОЕКТЫ WHERE (СЛУ_N=ПРО_РУК_N))
WHERE (ПРО_ЗАРП > 100000))
PROJECT (СЛУ_ИМЯ, СЛУ_N)

- выполнить эквисоединение СЛУЖАЩИЕ и ПРОЕКТЫ по условию СЛУ_N = ПРО_РУК_N;
- ограничить полученное отношение по условию ПРО_ЗАРП > 100000;
- спроецировать результат на атрибуты СЛУ_ИМЯ, СЛУ_N.

Пример формулы реляционного исчисления:
RANGE СЛУ IS СЛУЖАЩИЕ
RANGE ПРО IS ПРОЕКТЫ
СЛУ.СЛУ_ИМЯ, СЛУ.СЛУ_N WHERE
EXISTS ПРО (СЛУ.СЛУ_N=ПРО.ПРО_РУК_N AND ПРО.ПРО_ЗАРП > 100000)

- выдать значения СЛУ_ИМЯ и СЛУ_N для каждого кортежа служащих такого, что существует кортеж проектов со значением ПРО_РУК_N, совпадающим со значением СЛУ_N этого кортежа служащих, и значением ПРО_ЗАРП, большим 100000.

Алгебраическая формулировка является **процедурной**, она задает последовательность действий для выполнения запроса,
Логическая формулировка является описательной (или **декларативной**), она описывает свойства желаемого результата.

Исчисление кортежей

Кортежная переменная
RANGE ИМЯ_ПЕРЕМЕННОЙ **IS** ИМЯ_ОТНОШЕНИЯ
Пример: RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
СЛУЖАЩИЙ.СЛУ_ИМЯ – текущее значение атрибута СЛУ_ИМЯ переменной СЛУЖАЩИЙ.

Правильно построенная формула (Well-Formed Formula, **WFF**)
выражение составленное из «**простых условий»** и логических операторов

«**Простое условие»** - операции сравнения значений атрибутов переменных и/или литерально заданных констант операциями = != < > <= >=
Пример: СЛУЖАЩИЙ.СЛУ_НОМ = 34
СЛУЖАЩИЙ.СЛУ_НОМ != ПРОЕКТ.ПРОЕКТ_РУК

Логические операции:
NOT, **AND**, **OR** и **IF ... THEN** (импликация : IF a THEN b = NOT a OR b)
Приоритет операций:
NOT > AND > OR > IF ... THEN

Если form – WFF, а comp – простое сравнение, то
(NOT form), (comp AND form), (comp OR form) и (IF comp THEN form) - являются WFF.

Пример WFF для отношения СЛУЖАЩИЕ:

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
WFF : IF (СЛУЖАЩИЙ.СЛУ_ИМЯ = ‘Иванов’) THEN (СЛУЖАЩИЙ.ПРО_N = 1)

Область истинности WFF:

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
35	Петров	30000	2
37	Федоров	20000	2

== СЛУЖАЩИЕ WHERE (NOT (СЛУЖАЩИЙ.СЛУ_ИМЯ == 'Иванов') OR (СЛУЖАЩИЙ.ПРО_N == 1))

Пример WFF двух отношений: СЛУЖАЩИЕ и ПРОЕКТЫ:

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРОЕКТЫ

ПРО_N	ПРО_РУК
1	Иванов
2	Федоров

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ
RANGE ПРОЕКТ IS ПРОЕКТЫ
СЛУЖАЩИЙ.СЛУ_ИМЯ = ПРОЕКТ.ПРО_РУК

Эта формула будет принимать значение true для следующих ПАР значений кортежных переменных СЛУЖАЩИЙ и ПРОЕКТ:

СЛУЖАЩИЙ					ПРОЕКТ	
СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N		ПРО_N	ПРО_РУК
34	Иванов	22000	1		1	Иванов
34	Иванов	22000	2		1	Иванов
37	Федоров	20000	2		2	Федоров

Возможный способ реализации - метод вложенных циклов – Nested Loops Join (наиболее общий, но в большинстве случаев неоптимальный способ).

Областью истинности WFF должно быть отношение
для работы с атрибутами можно использовать точечную нотацию

СЛУЖАЩИЙ. СЛУ_N	СЛУЖАЩИЙ. СЛУ_ИМЯ	СЛУЖАЩИЙ. СЛУ_ЗАРП	СЛУЖАЩИЙ. ПРО_N	ПРОЕКТ. ПРО_N	ПРОЕКТ. ПРО_РУК
34	Иванов	22000	1	1	Иванов
34	Иванов	22000	2	1	Иванов
37	Федоров	20000	2	2	Федоров

область истинности == СЛУЖАЩИЕ **JOIN** ПРОЕКТЫ WHERE СЛУ_ИМЯ = ПРОЕКТ_РУК
(с учетом особенности именования атрибутов результирующего отношения).

Если WFF двух переменных всегда истина, например:
(СЛУЖАЩИЙ.СЛУ_ИМЯ = СЛУЖАЩИЙ.СЛУ_ИМЯ) AND (ПРОЕКТ.ПРО_РУК = ПРОЕКТ.ПРО_РУК))
область истинности == СЛУЖАЩИЙ **TIMES** ПРОЕКТ.

Квантор существования (**EXISTS**)
Квантор всеобщности (FORALL)

Если form – это WFF, в которой участвует переменная var, то конструкции
EXISTS var (form) и **FORALL var (form)** тоже представляют собой WFF.

(EXISTS var (form)) == TRUE : ∃ tr(var): form(tr) == TRUE
(FORALL var (form)) == TRUE : ∀ tr(var): form(tr) == TRUE

Свободные и связанные переменные в WFF
Все переменные, входящие в WFF, при построении которой не использовались кванторы, являются **свободными**.

Если имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех вложенных WFF, построенных с ее участием, var является **связанной переменной**.

Только свободные переменные образуют область истинности WFF.
Связанная переменная не видна за пределами связавшего ее квантора.
При вычислении значения связывающей WFF используется не одно значение связанной переменной, а вся область ее определения.

Примеры использования квантора EXIST

RANGE СЛУ1 IS СЛУЖАЩИЕ
RANGE СЛУ2 IS СЛУЖАЩИЕ

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

EXISTS СЛУ2 (СЛУ1.СЛУ_ЗАРП > СЛУ2.СЛУ_ЗАРП)

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

область истинности == все служащие, получающие не минимальную зарплату.

Примеры использования квантора FORALL

RANGE СЛУ1 IS СЛУЖАЩИЕ
RANGE СЛУ2 IS СЛУЖАЩИЕ

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

FORALL СЛУ2 (СЛУ1.СЛУ_ЗАРП >= СЛУ2.СЛУ_ЗАРП)

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
35	Петров	30000	1
35	Петров	30000	2

область истинности == все служащие, получающие максимальную зарплату.

Свободные и связанные **вхождения переменных**
Если переменная var является связанной в WFF form, то во всех WFF, включающих form, вне form может использоваться вхождение того же имени переменной var, которое может быть свободным или связанным, но в любом случае не имеет никакого отношения к вхождению переменной var в WFF form.

Пример WFF использующей повторное вхождение связанной переменной:

EXISTS СЛУ2 (СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ AND СЛУ1.СЛУ_НОМЕР != СЛУ2.СЛУ_НОМЕР)
AND FORALL СЛУ2 (IF СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ THEN СЛУ1.СЛУ_ЗАРП = СЛУ2.СЛУ_ЗАРП)

область истинности == служащие всех проектов с числом участников больше 1-ого и получающие в каждом проекте одинаковую зарплату

Целевой список (**target list**)

Список, определяющий подмножество результирующих атрибутов WFF, состоящий из элементов следующего вида:

- var.attr, где var – имя свободной переменной соответствующей WFF, а attr – имя атрибута отношения, на котором определена переменная var;
- var, что эквивалентно наличию подсписка var.attr1, var.attr2, ..., var.attrn, где {attr1, attr2, ..., attrn} имена всех атрибутов переменной;
- new_name = var.attr; new_name – новое имя атрибута attr переменной var.

Выражение реляционного исчисления

Конструкция вида: **target_list WHERE WFF**
Результат - отношение, тело которого определяется WFF, а схема – целевым списком.

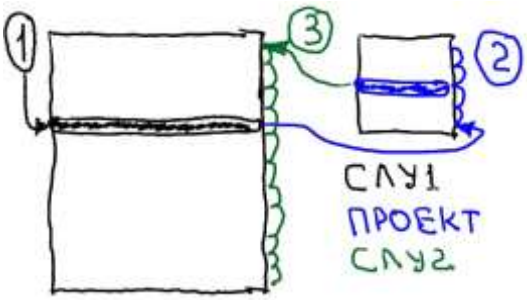
Пример: СЛУЖАЩИЕ DIVIDE BY ПРОЕКТЫ
RANGE СЛУ1, СЛУ2 **IS** СЛУЖАЩИЕ
RANGE ПРОЕКТ **IS** ПРОЕКТЫ
СЛУ1.СЛУ_N, СЛУ1.СЛУ_ИМЯ, СЛУ1.СЛУ_ЗАРП
WHERE FORALL ПРОЕКТ **EXISTS** СЛУ2
(СЛУ1.СЛУ_N = СЛУ2.СЛУ_N AND СЛУ2.ПРО_N = ПРОЕКТ.ПРО_N)

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N
34	Иванов	22000	1
35	Петров	30000	1
36	Сидоров	18000	1
34	Иванов	22000	2
35	Петров	30000	2
37	Федоров	20000	2

ПРОЕКТЫ

ПРО_N
1
2



Результат этого выражения является отношение

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП
34	Иванов	22000
35	Петров	30000

Исчисление доменов

Областью определения переменных являются домены.
Отношение СЛУЖАЩИЕ - доменные переменные:
ИМЯ - значения – допустимые имена
НОМЕР_СЛУЖАЩЕГО - значения – допустимые номера служащих.

Предикат «Условие членства»
для отношения r : Hr = {a_1, a_2, ..., a_n} называют конструкцию вида:
R (a_{i1} : v_{i1}, a_{i2} : v_{i2}, ..., a_{im} : v_{im}),

где:

a_{ij} – некоторый атрибут $\in$ Hr = {a_1, a_2, ..., a_n}, (m <= n);

v_{ij} – литерально задаваемая константа либо имя доменной переменной.

R() == true : $\exists$ tr(Hr) $\in$ Br : $\forall$ <a_i,T_i,v_i> $\in$ tr
Область истинности – наборы свободных доменных переменных, при которых WFF Условия членства принимает истинное значение.

если v_{ij} – константа, то на атрибут a_{ij} накладывается жесткое условие.
если v_{ij} – имя доменной переменной, то условие членства зависит от значения этой переменной (т.е. фактически тело отношения ограничивает значение свободной доменной переменной).

WFF исчисления доменов

СЛУЖАЩИЕ (СЛУ_N:34, СЛУ_ИМЯ:'Иванов', СЛУ_ЗАРП:22000, ПРО_N:1)
true если в теле отношения СЛУЖАЩИЕ содержится кортеж <34, 'Иванов', 22000, 1>.

СЛУЖАЩИЕ (СЛУ_N:34, СЛУ_ИМЯ:'Иванов', СЛУ_ЗАРП:22000, ПРО_N: ПРО_N1)
true для всех комбинаций явно заданных значений и допустимых значений переменной ПРО_N1.

Для нашего отношения СЛУЖАЩИЕ областью истинности этой WFF является два значения доменной переменной ПРО_N1 <1> и <2>, полученные из кортежей <34, 'Иванов', 22000, 1> и <34, 'Иванов', 22000, 2>.

Выражения исчисления доменов

Пример: Выдать номера и имена служащих, получающих не минимальную зарплату:

СЛУ_N1, СЛУ_ИМЯ1 WHERE EXISTS СЛУ_ЗАРП2
СЛУЖАЩИЕ (СЛУ_ЗАРП: СЛУ_ЗАРП2) AND
СЛУЖАЩИЕ (СЛУ_N: СЛУ_N1, СЛУ_ИМЯ: СЛУ_ИМЯ1 , СЛУ_ЗАРП: СЛУ_ЗАРП1) AND
СЛУ_ЗАРП1 > СЛУ_ЗАРП2

На исчислении доменов базируется известный язык, основанных на табличных формах, Query-by-Example (QBE).

Теория проектирования реляционных баз данных

Проблемы проектирования реляционных баз данных:
- **логического проектирования;**
- **физического проектирования**.

Последовательность нормальных форм в теории реляционных баз данных:
- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:
- каждая следующая NF в некотором смысле лучше предыдущей NF;
- при переходе к следующей NF свойства предыдущих NF сохраняются.

Функциональная зависимость (Functional Dependency (FD))

Пусть:

r – некоторое отношение,

$X,Y \in Hr$.

Тогда:

В отношении **r** атрибут **Y функционально зависит** от атрибута **X** в том и только в том случае, если каждому значению X соответствует в точности одно значение Y.

Обозначение:

FD $r.X \rightarrow r.Y$

X - детерминант (**определитель**) для Y,

Y - зависимое от X.

СЛУЖАЩИЕ

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
35	Петров	18000	1	Иванов
36	Сидоров	18000	1	Иванов
37	Федоров	31000	2	Федоров
41	Сидоренко	21000	2	Федоров

Номера всех служащих уникальны $\Rightarrow \exists$ FD:
СЛУ_N $\rightarrow$ СЛУ_ИМЯ.
СЛУ_N $\rightarrow$ СЛУ_ЗАРП
СЛУ_N $\rightarrow$ ПРО_N
СЛУ_N $\rightarrow$ ПРО_РУК
{ СЛУ_N, СЛУ_ИМЯ } $\rightarrow$ СЛУ_ИМЯ
{ СЛУ_N, СЛУ_ИМЯ } $\rightarrow$ СЛУ_ЗАРП
{ СЛУ_N, СЛУ_ИМЯ } $\rightarrow$ ПРО_N
{ СЛУ_N, СЛУ_ИМЯ } $\rightarrow$ { СЛУ_ЗАРП, ПРО_N }
…
ПРО_N $\rightarrow$ ПРО_РУК
(из знания предметной области)

…
Если имена уникальны $\Rightarrow \exists$ FD:
СЛУ_ИМЯ $\rightarrow$ СЛУ_N
СЛУ_ИМЯ $\rightarrow$ СЛУ_ЗАРП
СЛУ_ИМЯ $\rightarrow$ ПРО_N
…
Исходя из тела отношения $\exists$ FD:
СЛУ_ЗАРП $\rightarrow$ ПРО_N

Интересны FD, которые должны выполняться всегда для любых значений отношения.

A $\in$ Hr – ключ $\Rightarrow \forall$ B $\in$ Hr $\exists$ FD A$\rightarrow$B

Если FD трактуются как ограничения целостности, то за их соблюдением должна следить СУБД => одна из основных задач теории баз данных - уметь сократить набор FD до некоторого минимума, поддержка которого гарантирует выполнение, тем не менее, соблюдение всех зависимостей.

Тривиальная FD
FD A $\rightarrow$ B тривиальная : B $\in$ A

Пример:
ABC $\rightarrow$ AB
{СЛУ_ЗАРП, ПРО_НОМ} $\rightarrow$ СЛУ_ЗАРП
A $\rightarrow$ A
Тривиальная FD выполняется всегда и их нельзя трактовать как ограничения целостности.

Примеры логического вывода:
Отношение СЛУЖАЩИЕ.
Из FD СЛУ_N $\rightarrow$ {СЛУ_ЗАРП, ПРО_N } => FD СЛУ_N $\rightarrow$ СЛУ_ЗАРП и СЛУ_N $\rightarrow$ ПРО_N.
Из FD СЛУ_N $\rightarrow$ ПРО_N и ПРО_N $\rightarrow$ ПРО_РУК => FD СЛУ_N $\rightarrow$ ПРО_РУК

Транзитивная FD
FD A$\rightarrow$C транзитивна == $\exists$ B $\in$ Hr: $\exists$ A$\rightarrow$B & $\exists$ B$\rightarrow$C & !$\exists$ C$\rightarrow$A

Замыкание множества FD S
Такое множество FD S+, которое включает все FD, логически выводимые из множества S.

Аксиомы Армстронга

A, B и C $\in$ Hr, (возможно составные, возможно имеющие непустое пересечение).
Тогда истинны следующие утверждения:

1. если B $\in$ A, то A $\rightarrow$ B (**рефлексивность**);
2. если A $\rightarrow$ B, то AC $\rightarrow$ BC (**пополнение**);
3. если A $\rightarrow$ B и B $\rightarrow$ C, то A $\rightarrow$ C (**транзитивность**).

Истинность первой аксиомы:

следует из определения тривиальной FD, поскольку при B $\in$ A FD A $\rightarrow$ B является тривиальной и выполняется всегда.

Истинность второй аксиомы:

Пусть, что FD AC $\rightarrow$ BC не соблюдается.

⇨ $\exists$ t1 и t2 $\in$ Br : t1{AC} = t2{AC} (**a**), но t1 {BC} != t2 {BC} (**b**).

По аксиоме (1) ($\exists$ тривиальная FD AC $\rightarrow$ A) из равенства (a) => t1 {A} = t2 {A} (**c**).
Из наличия FD A $\rightarrow$ B + (c) => t1 {B} = t2 {B} (**d**).
Тогда из (b) + (d) => t1 {C} != t2 {C},
Но из $\exists$ тривиальной FD AC $\rightarrow$ C по аксиоме (1) + (a) => t1 {C} = t2 {C}.
Противоречие => FD AC $\rightarrow$ BC соблюдается.

Истинность третьей аксиомы:

Пусть, что FD A $\rightarrow$ C не соблюдается.

⇨ $\exists$ t1 и t2 $\in$ Br : t1 {A} = t2 {A}, но t1 {C} != t2 {C}.

Из наличия FD A $\rightarrow$ B => t1 {B} = t2 {B},
Из наличия FD B $\rightarrow$ C => t1 {C} = t2 {C}
Противоречие => FD A $\rightarrow$ C соблюдается.

Аксиомы Армстронга:

1. если $B \in A$, то $A \to B$ (**рефлексивность**);
2. если $A \to B$, то $AC \to BC$ (**пополнение**);
3. если $A \to B$ и $B \to C$, то $A \to C$ (**транзитивность**).

Пять правил Дейта:

4. $A \to A$ (**самодетерминированность**) – прямо следует из правила (1);
5. если $A \to BC$, то $A \to B$ и $A \to C$ (**декомпозиция**) –
   из правила (1) $\exists$ $BC \to B$; + по правилу (3) => $A \to B$;
   аналогично для C, из правила (1) $\exists$ $BC \to C$ + по правилу (3) => $A \to C$;
6. если $A \to B$ и $A \to C$, то $A \to BC$ (**объединение**) –
   из правила (2) (пополняя A и B) следует, что $A \to AB$ и $AB \to BC$;
   + из правила (3) следует, что $A \to BC$;
7. если $A \to B$ и $C \to D$, то $AC \to BD$ (**композиция**) –
   из правила (2) (пополняя C и B) следует, что $AC \to BC$ и $BC \to BD$;
   + из правила (3) следует, что $AC \to BD$;
8. если $A \to BC$ и $B \to D$, то $A \to BCD$ (**накопление**) –
   из правила (2) (пополняя только вторую BC) следует, что $BC \to BCD$;
   из правила (3) следует, что $A \to BCD$.

Замыкание множества атрибутов на множестве функциональных зависимостей

Пусть
r - отношение, Z{} $\in$ Hr, S - некоторое множество {FD}, выполняемых для r.
Тогда **замыканием Z над S**
Z+ = {Yi} : $\forall$ Yi $\in$ Hr : $\exists$ FD Z $\rightarrow$ Yi $\in$ S+.

Вариант алгоритма вычисления Z+:
Z+=Z
DO
    Zprev=Z+
    FOR EACH FD A->B IN S DO
    IF A$\in$Z+ THEN Z+=Z+$\cup$B
    END DO
UNTIL Z+!=Zprev

Пример работы алгоритма вычисления Z+:
Пусть r {A, B, C, D, E, F} и S = {A $\rightarrow$ D, AB $\rightarrow$ E, BF $\rightarrow$ E, CD $\rightarrow$ F, E $\rightarrow$ C}.
Требуется найти {AE}+ над S.
Z+[0] = {AE}
FD A $\rightarrow$ D, E $\rightarrow$ C => Z+[1] = { ACDE }
FD CD $\rightarrow$ F => Z+[2] = { ACDEF }
Z+[3] = { ACDEF }

Назначение Z+ - проверка входит ли данная FD в замыкание S+.
FD Z$\rightarrow$B $\in$ S+ $\Leftrightarrow$ B $\in$ Z+ (от Z на S)

Суперключом отношения r называется $K \in Hr : \exists$ возможный ключ $A : A \in K$.

Следствия:

1. K – суперключ $\Leftrightarrow \forall A \in Hr : \exists$ FD $K \rightarrow A$
2. K – суперключ $\Leftrightarrow K+ == Hr$

Покрытие множества функциональных зависимостей:

Множество FD S2 называется **покрытием множества FD** S1,
если любая FD, выводимая из S1, выводится также из S2.
Т.е., S2 является покрытием S1 тогда и только тогда, когда $S1+ \in S2+$.

Два множества FD S1 и S2 называются **эквивалентными**, если каждое из них является покрытием другого, т. е. $S1+ = S2+$.

Минимальное множество функциональных зависимостей

Множество FD S называется минимальным в том и только в том случае, когда удовлетворяет следующим свойствам:

1. правая часть любой FD из S является множеством из одного атрибута (простым атрибутом);
2. детерминант каждой FD из S обладает свойством минимальности; это означает, что удаление любого атрибута из детерминанта приводит к изменению замыкания S+, т. е. порождению множества FD, не эквивалентного S;
3. удаление любой FD из S приводит к изменению S+, т. е. порождению множества FD, не эквивалентного S.

Пример
СЛУЖАЩИЕ {СЛУ_N, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ, ПРО_РУК}.
Если СЛУ_N - единственный возможный ключ, то минимальное множество:
FD {СЛУ_N → СЛУ_ИМЯ, СЛУ_N → СЛУ_ЗАРП, СЛУ_N → ПРО_НОМ, ПРО_НОМ → ПРО_РУК}

Пример не минимальных множеств:

1. {СЛУ_N→**{СЛУ_ИМЯ, СЛУ_ЗАРП}**, СЛУ_N→ПРО_N, СЛУ_N→ПРО_РУК, ПРО_N→ПРО_РУК},
2. {СЛУ_N→СЛУ_ИМЯ, **{СЛУ_N, СЛУ_ИМЯ}**→СЛУ_ЗАРП, СЛУ_N→ПРО_N, СЛУ_N→ ПРО_РУК},
3. {СЛУ_N→СЛУ_ИМЯ, СЛУ_N→СЛУ_ЗАРП, СЛУ_N→ПРО_N, **СЛУ_N→ПРО_РУК**, ПРО_N→ПРО_РУК}

Из { СЛУ_N, СЛУ_ИМЯ } → СЛУ_ЗАРП получаем СЛУ_N → СЛУ_ЗАРП:

- есть СЛУ_N →СЛУ_ИМЯ, пополняя СЛУ_N =>
- СЛУ_N → { СЛУ_N, СЛУ_ИМЯ } + есть { СЛУ_N, СЛУ_ИМЯ } → СЛУ_ЗАРП, по транзитивности
- СЛУ_N → СЛУ_ЗАРП

Обратно из СЛУ_N → СЛУ_ЗАРП получаем { СЛУ_N, СЛУ_ИМЯ } → СЛУ_ЗАРП:

- рефлексивность: { СЛУ_N, СЛУ_ИМЯ }→ СЛУ_N,
- + есть СЛУ_N → СЛУ_ЗАРП, по транзитивности
- { СЛУ_N, СЛУ_ИМЯ } → СЛУ_ЗАРП

Для любого множества FD S существует и может быть построено **эквивалентное** ему **минимальное множество S-**. Такое множество, вообще говоря, не единственно.

Общая схема построения минимального эквивалентного множества S- по заданному S.

1. Используя правило (5) (декомпозиции), приводим S к эквивалентному множеству FD S1, правые части FD которого содержат только простые атрибуты;
2. Для каждой FD из S2(начальное состояние S1) : детерминант D{D1, D2, …, Dn} n>1, удаляем Di, получая FD S2.
   Если S2+ != S1+, то этот атрибут восстанавливается в S2.
   Проверка выполняется для всех атрибутов из составных детерминантов S2 по одному разу.
3. Для каждой FD f $\in$ S3 (начальное состояние S2) :
   Если S3+ == (S3 MINUS {f})+, то S3 = S3 MINUS {f}

Полученное множество S3 - минимально и эквивалентно исходному множеству FD S.

ПРИМЕР построения минимального множества:
Отношение r {A,B,C,D} и FD S = {A → B, A → BC, AB → C, AC → D, B → C}.

1. декомпозиция - по правилу декомпозиции S эквивалентно множеству
   S1 {A → B, A → C, AB → C, AC → D, B → C}.
2. удаление лишних атрибутов в определителях FD:

- AC→D можно заменить на A→D (удалить атрибут C):
  A → C (пополняя A) => A → AC; + транзитивность => A → D,
  и обратно из ∃ тривиальной AC → A, + транзитивность A → D => AC → D,
  таким образом C в детерминанте AC → D является избыточным.

3. удаление лишних FD:
   AB → C – логически выводится и может быть удалена,
   (A→C + пополняя B => AB → BC, а по правилу декомпозиции (5) => AB → C)
   A → C - логически выводится и может быть удалена,
   (A → B + B → C по правилу транзитивности => A → C).

{A → B, A → D, B → C} - минимальное и эквивалентное исходному S по построению.

Минимальное покрытие множества функциональных зависимостей

Минимальным покрытием множества FD S называют любое минимальное множество FD S1, эквивалентное S.

Корректные и некорректные декомпозиции отношений

Декомпозициями без потерь могут считаться только такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Пример

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
37	Федоров	22000	2	Федоров

ДЕКОМПОЗИЦИЯ 1

СЛУ1

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП
34	Иванов	22000
37	Федоров	22000

СЛУ_N	ПРО_N	ПРО_РУК
34	1	Иванов
37	2	Федоров

СЛУ1 NATURAL JOIN СЛУ2

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
37	Федоров	22000	2	Федоров

ДЕКОМПОЗИЦИЯ 2

СЛУ3

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП
34	Иванов	22000
37	Федоров	22000

СЛУ4

СЛУ_ЗАРП	ПРО_N	ПРО_РУК
22000	1	Иванов
22000	2	Федоров

СЛУ3 NATURAL JOIN СЛУ4

СЛУ_N	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_N	ПРО_РУК
34	Иванов	22000	1	Иванов
34	Иванов	22000	2	Федоров
37	Федоров	22000	1	Иванов
37	Федоров	22000	2	Федоров

(причина - отсутствие в СЛУ4 FD СЛУ_ЗАРП $\rightarrow$ ПРО_N и СЛУ_ЗАРП $\rightarrow$ ПРО_РУК)

Теорема Хита о возможности проведения декомпозиции отношения без потерь

Пусть задано отношение r {A, B, C} (A, B и C, в общем случае, являются составными атрибутами) и выполняется FD A $\rightarrow$ B.
Тогда r = (r PROJECT {A, B}) NATURAL JOIN (r PROJECT {A, C}).

Доказательство.
r – исходное отношение
r1 == (r PROJECT {A, B})
r2 == (r PROJECT {A, C})
r3 == r1 NATURAL JOIN r2

1. Докажем что если {a, b, c} $\in$ r, то {a, b, c} $\in$ r3
   {a, b, c} $\in$ r
   =>(по определению PROJECT) {a, b} $\in$ r1, {a, c} $\in$ r2
   =>(по определению NATURAL JOIN) {a, b, c} $\in$ r3
2. Докажем что если {a, b, c} $\in$ r3, то {a, b, c} $\in$ r
   {a, b, c} $\in$ r3
   =>(по определению NATURAL JOIN) {a, b} $\in$ r1, {a, c} $\in$ r2
   =>(по определению PROJECT) {a, b, X} $\in$ r, {a, Y, c} $\in$ r
   Но в r $\exists$ FD A $\rightarrow$ B => Y == b => {a, b, c} $\in$ r

Конец доказательства.

Пример СЛУЖАЩИЕ_ОТДЕЛЫ_ПРОЕКТЫ {СЛУ_N, СЛУ_ОТД, ПРО_N}.

СЛУ_ОТД_ПРО

СЛУ_N	СЛУ_ОТД	ПРО_N
34	10	1
34	10	2
37	11	1
37	11	2

СЛУ_N → СЛУ_ОТД

СЛУ_ОТД

СЛУ_N	СЛУ_ОТД
34	10
37	11

СЛУ_ПРО

СЛУ_N	ПРО_N
34	1
34	2
37	1
37	2

СЛУ_ОТД_ПРО == СЛУ_ОТД NATURAL JOIN СЛУ_ПРО

FD называется минимальной слева если детерминант нельзя уменьшить без изменения замыкания.

Атрибут B минимально зависит от атрибута A, если ∃ минимальная слева FD A → B.

Например:
В множестве FD { СЛУ_N → СЛУ_ЗАРП, {СЛУ_N, СЛУ_ИМЯ} → СЛУ_ЗАРП}.
СЛУ_ЗАРП зависит от {СЛУ_N, СЛУ_ИМЯ} не минимально.

Диаграммы функциональных зависимостей

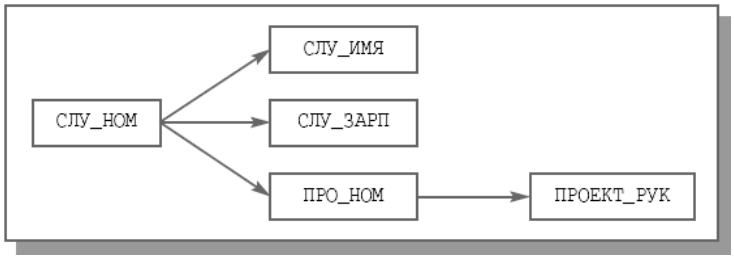


Диаграмма минимального множества FD отношения СЛУЖАЩИЕ

Проектирование реляционных баз данных на основе принципов нормализации

Классический подход, представляет весь процесс проектирования базы данных в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений.

Исходная точка - представление предметной области в виде одного или нескольких отношений с большим количеством атрибутов.

Шаг проектирования – модификация схем отношений, так, чтобы они обладали некоторым новым «улучшенными» свойствами.

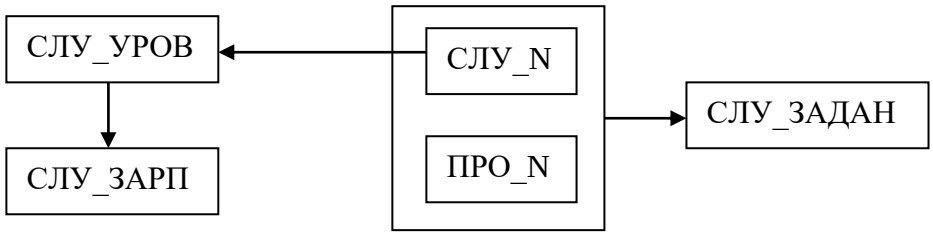
Процесс проектирования принято называть процессом **нормализации схем отношений**, или другими словами декомпозиции исходного отношения, находящегося в предыдущей нормальной форме, на два или более отношений, которые удовлетворяют требованиям следующей нормальной формы.

Нормальная форма представляет собой определенный набор ограничений. Считают, что отношение находится в некоторой нормальной форме, если оно удовлетворяет данному набору ограничений.

Каждая следующая нормальная форма обладает свойствами, в некотором смысле, лучшими, чем предыдущая. При этом свойства всех предыдущих форм сохраняются.

Минимальные функциональные зависимости и вторая нормальная форма

Диаграмма минимального множества FD отношения
СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ



СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ

СЛУ_N	СЛУ_УРОВ	СЛУ_ЗАРП	ПРО_N	СЛУ_ЗАДАН
34	2	22000	1	A
36	1	18000	1	B
37	1	18000	1	D
34	2	22000	2	D
35	3	30000	2	C

Данное отношение удовлетворяет требованию первой нормальной формы 1NF: значения всех ее атрибутов **атомарны**.

Аномалии обновления, возникающие из-за наличия не минимальных функциональных зависимостей

Под **аномалиями обновления** мы будем понимать трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE).

- **Добавление кортежей**. Мы не можем добавить кортеж с данными о служащем, который не участвует ни в одном проекте (ПРО_N - частью первичного ключа).
- **Удаление кортежей**. Мы не можем сохранить кортеж с данными о служащем, завершившем участие в своем последнем проекте.
- **Модификация кортежей**. Чтобы изменить разряд служащего, мы будем вынуждены модифицировать несколько кортежей с соответствующим значением атрибута СЛУ_N.

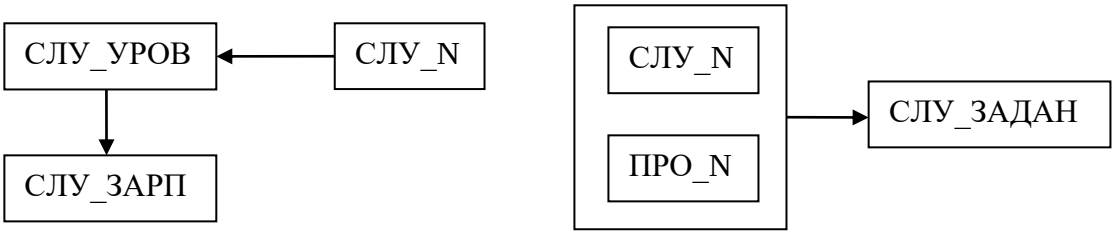
Неформально:
СЛУ_N $\rightarrow$ СЛУ_УРОВ - вызывает избыточность данных, описывая «служащего», а не «служащего в некотором проекте».

Формально:
СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ содержит FD, в которых детерминантом является не возможный ключ отношения, а его часть, в результате чего некоторые FD атрибутов от первичного ключа не являются минимальными.
(Например FD { СЛУ_N, ПРО_N } $\rightarrow$ СЛУ_УРОВ).

Возможная декомпозиция

СЛУЖ {СЛУ_N, СЛУ_УРОВ, СЛУ_ЗАРП} и СЛУЖ_ПРО_ЗАДАН {СЛУ_N, ПРО_N, СЛУ_ЗАДАН}

∃ FD СЛУ_N→{ СЛУ_УРОВ, СЛУ_ЗАРП } => можно декомпозировать без потерь



СЛУЖ

СЛУ_N	СЛУ_УРОВ	СЛУ_ЗАРП
34	2	22000
36	1	18000
37	1	18000
35	3	30000

СЛУЖ_ПРО_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
36	1	B
37	1	D
34	2	D
35	2	C

Отношение находится во **второй нормальной форме** (**2NF**) тогда и только тогда, когда оно находится в первой нормальной форме, и каждый неключевой атрибут, т.е. атрибут, не входящий ни в один возможный ключ, минимально функционально зависит от первичного ключа.

СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ не находится в 2NF

FD {СЛУ_N, ПРО_N} → СЛУ_УРОВ не является минимальной (ПРО_N – можно удалить).

СЛУЖ и СЛУЖ_ПРО_ЗАДАН находятся в 2NF

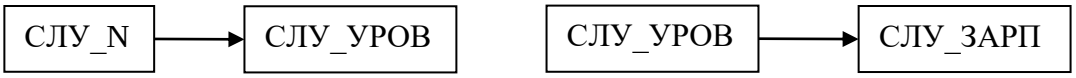
все неключевые атрибуты минимально зависят от первичных ключей СЛУ_N и {СЛУ_N, ПРО_N}.

Аномалии обновлений, возникающие из-за наличия транзитивных функциональных зависимостей в отношении СЛУЖ

Наличие транзитивной СЛУ_N→СЛУ_ЗАРП (СЛУ_N→СЛУ_УРОВ и СЛУ_УРОВ→СЛУ_ЗАРП) вызывает избыточностью хранения атрибута СЛУ_ЗАРП в каждом кортеже служащего:

- **Добавление кортежей**. Невозможно сохранить данные о новом разряде (и соответствующем ему размере зарплаты), пока не появится служащий с новым разрядом.
- **Удаление кортежей**. При увольнении последнего служащего с данным разрядом мы утратим информацию о наличии такого разряда и соответствующем размере зарплаты.
- **Модификация кортежей**. При изменении размера зарплаты, соответствующей некоторому разряду, мы будем вынуждены изменить значение атрибута СЛУ_ЗАРП в кортежах всех служащих, которым назначен этот разряд.

Возможная декомпозиция:
СЛУЖ1 {СЛУ_N, СЛУ_УРОВ} и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}
(∃ FD СЛУ_УРОВ → СЛУ_ЗАРП => можно декомпозировать без потерь)



СЛУЖ1

СЛУ_N	СЛУ_УРОВ
34	2
36	1
37	1
35	3

УРОВ

СЛУ_УРОВ	СЛУ_ЗАРП
2	22000
1	18000
3	30000

Переменная отношения находится в **третьей нормальной форме** (**3NF**) в том и только в том случае, когда она находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

Независимые проекции отношений. Теорема Риссанена

Возможные декомпозиции СЛУЖ {СЛУ_N, СЛУ_УРОВ, СЛУ_ЗАРП}:

1. A{СЛУ_N, СЛУ_УРОВ} и B{СЛУ_УРОВ, СЛУ_ЗАРП}
2. A{СЛУ_N, СЛУ_УРОВ} и B{СЛУ_N, СЛУ_ЗАРП}
3. A{СЛУ_N, СЛУ_ЗАРП} и B{СЛУ_УРОВ, СЛУ_ЗАРП}

3 – с потерями (по теореме Хита)
2 – не устраняет аномалий обновления (невозможно сохранить информацию о разряде)
1 – ок

В процессе нормализации декомпозиция отношения на независимые проекции является предпочтительной. Необходимые и достаточные условия независимости проекций отношения обеспечивает теорема Риссанена.

Теорема Риссанена

Проекции r1 и r2 отношения r являются **независимыми** тогда и только тогда, когда:

- каждая FD в отношении r логически следует (выводится по Армстронгу) из FD в r1 и r2; ($\forall FD \in r \Leftarrow \{FD \in r1 \cup FD \in r2\}$)
- общие атрибуты r1 и r2 образуют возможный ключ хотя бы для одного из этих отношений. ($Hr1 \cap Hr2$ – возм. ключ r1 или r2)

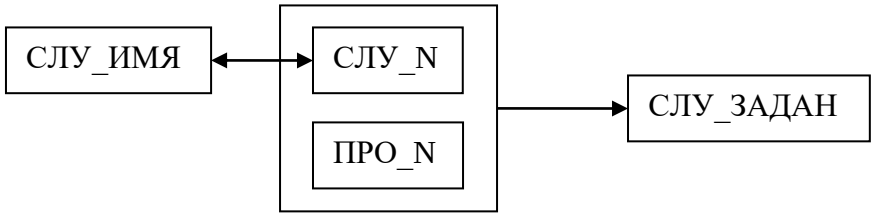
Пример 1: СЛУ_N→СЛУ_ЗАРП <= СЛУ_N→СЛУ_УРОВ и СЛУ_УРОВ→СЛУ_ЗАРП
Пример 2: СЛУ_УРОВ→СЛУ_ЗАРП <=/= СЛУ_N→СЛУ_УРОВ и СЛУ_N→СЛУ_ЗАРП.

Атомарным отношением называется отношение, которое невозможно декомпозировать на независимые проекции.

Далеко не всегда для неатомарных отношений требуется декомпозиция на атомарные проекции. При выборе способа декомпозиции нужно стремиться к получению независимых проекций, но не обязательно атомарных.

Аномалии обновлений, связанные с наличием перекрывающихся возможных ключей

СЛУЖ_ПРО_ЗАДАН1 {СЛУ_N, СЛУ_ИМЯ, ПРО_N, СЛУ_ЗАДАН} с множеством FD:



СЛУЖ_ПРО_ЗАДАН1

СЛУ_N	СЛУ_ИМЯ	ПРО_N	СЛУ_ЗАДАН
34	Иванов	1	A
35	Федоров	2	B
34	Иванов	2	B
35	Федоров	1	A

В отношении СЛУЖ_ПРО_ЗАДАН1 служащие уникально идентифицируются как по номерам удостоверений, так и по именам, каждый служащий может участвовать в нескольких проектах.

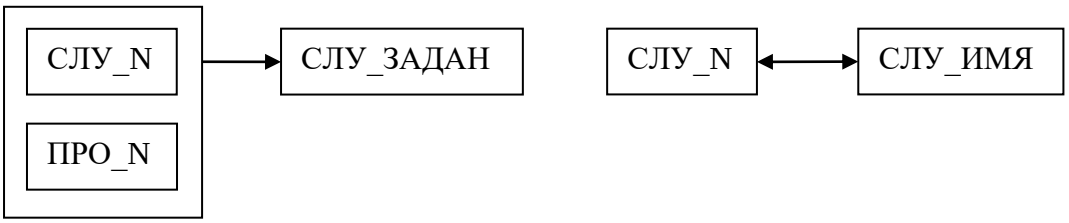
Условия 2NF и 3NF соблюдены, однако имеются **аномалии обновления**:
в случае изменения имени служащего требуется обновить атрибут СЛУ_ИМЯ в нескольких картежах, иначе будет нарушена СЛУ_N→СЛУ_ИМЯ.

Нормальная форма Бойса-Кодда

Отношение находится в **нормальной форме Бойса-Кодда** (**BCNF**) в том и только в том случае, когда оно находится в **3NF** и любая нетривиальная и минимальная FD этого отношения имеет в качестве детерминанта некоторый возможный ключ.

СЛУЖ_ПРО_ЗАДАН1 может быть приведена к BCNF путем одной из двух декомпозиций:

1. СЛУЖ_ИМЯ {СЛУ_N, СЛУ_ИМЯ} и НОМ_ЗАДАН {СЛУ_N, ПРО_N, СЛУ_ЗАДАН},
2. СЛУЖ_ИМЯ {СЛУ_N, СЛУ_ИМЯ} и ИМЯ_ЗАДАН {СЛУ_ИМЯ, ПРО_N, СЛУ_ЗАДАН}



НОМ_ЗАДАН

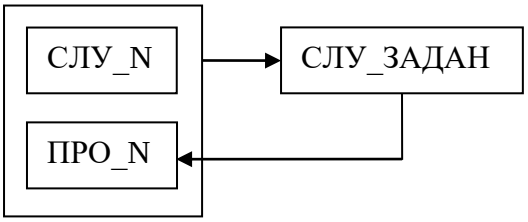
СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
35	2	B
34	2	B
35	1	A

СЛУЖ_ИМЯ

СЛУ_N	СЛУ_ИМЯ
34	Иванов
35	Федоров

Всегда ли следует стремиться к BCNF?

- служащий может участвовать в нескольких проектах
- в каждом проекте служащий выполняет одно задание
- задание может быть связано только с одним проектом.



СЛУЖ_ПРО_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
35	1	A
41	2	B
34	2	C
41	1	D

Возможные ключи:
{СЛУ_N, ПРО_N}
{СЛУ_N, СЛУ_ЗАДАН}

Находится в 3NF (нет неключевых атрибутов), но **НЕ** находится в BCNF (СЛУ_ЗАДАН→ПРО_N)

Аномалии обновления: невозможно удалить данные о единственном служащем, выполняющем задание в некотором проекте, не утратив информацию об этом задании (в каком проекте оно выполняется).

Возможная декомпозиция для получения BCNF устраняющей аномалии:

СЛУЖ_ЗАДАН

СЛУ_N	СЛУ_ЗАДАН
34	A
35	A
41	B
34	C
41	D

ПРО_ЗАДАН

ПРО_N	СЛУ_ЗАДАН
1	A
2	B
2	C
1	D

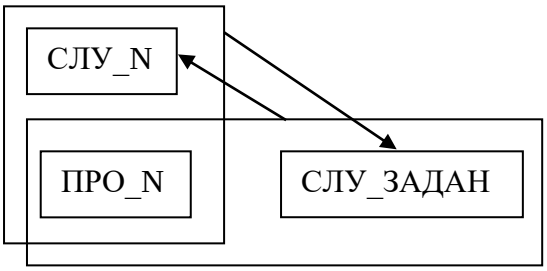
Имеется **проблема**: добавление <34, D> в отношение СЛУЖ_ЗАДАН система должна запретить (D относится к проекту 1, в котором служащий 34 уже выполняет задание A).

{СЛУ_N, ПРО_N}→СЛУ_ЗАДАН не выводится из зависимостей результирующих отношений. Эта FD должна быть определена как ограничение целостности БД общего вида.

Т.о. проекции СЛУЖ_ЗАДАН и ПРО_ЗАДАН - не являются независимыми, а отношение СЛУЖ_ПРО_ЗАДАН атомарно, хотя и не находится в BCNF.

Отношение, с двумя перекрывающимися возможными ключами, находящееся в BCNF

Служащий может участвовать в нескольких проектах.
В каждом проекте служащий выполняет только одно задание.
Каждое задание может присутствовать в разных проектах.
В каждом проекте задание выполняться только одним служащим.



СЛУЖ_ПРО_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
35	1	B
41	2	B
34	2	A
41	1	D

Возможные ключи: {СЛУ_N, ПРО_N} и {ПРО_N, СЛУ_ЗАДАН},
но отношение находится в BCNF, поскольку эти ключи являются единственными детерминантами, а значит эти зависимости минимальные.

Не функциональные зависимости

СЛУЖ_ПРО_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
34	1	B
34	2	A
34	2	B
41	1	A
41	1	D

- служащий может участвовать в нескольких проектах
- в каждом проекте служащий выполняет одинаковый набор заданий

Ключ отношения {СЛУ_N, ПРО_N, СЛУ_ЗАДАН}
Находится в BCBF (отсутствуют нетривиальные FD)

IF(<с,п1,з1>∈СПЗ AND <с,п2,з2>∈СПЗ)THEN(<с,п1,з2>∈СПЗ AND <с,п2,з1>∈СПЗ)

Аномалии обновления:

- **Добавление кортежа**. Если служащий присоединяется к проекту, то необходимо добавить столько кортежей, сколько заданий выполняет этот служащий.
- **Удаление кортежей**. Если служащий прекращает участие в проектах, то отсутствует возможность сохранить данные о заданиях, которые он может выполнять.
- **Модификация кортежей**. При изменении одного из заданий служащего необходимо изменить значение атрибута СЛУ_ЗАДАН в нескольких кортежах, для каждого проекта.

Возможная декомпозиция:

СЛУЖ_ПРО

СЛУ_N	ПРО_N
34	1
34	2
41	1

СЛУЖ_ЗАДАН

СЛУ_N	СЛУ_ЗАДАН
34	A
34	B
41	A
41	D

В отношении нет FD, однако:

1. Декомпозиция без потерь
2. Проблемы с аномалиями решены

Многозначная зависимость.

Впервые описаны Роном Фейджином в 1971 г. Фейджин назвал зависимости этого вида многозначными (*multi-valued dependency* – MVD).

В СЛУЖ_ПРО_ЗАДАН выполняются две MVD: СЛУ_N→→ПРО_N и СЛУ_N→→СЛУ_ЗАДАН

В терминах алгебраических выражений MVD СЛУ_N→→ПРО_N означает, что результат:
Ri = (СЛУЖ_ПРО WHERE (СЛУ_N = ‘N’ AND СЛУ_ЗАДАН = ‘Zi’)) PROJECT {ПРО_N}
‘N’ = const & ∀ Zi, Zj => Ri = Rj

В переменной отношения r с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A (A→→B) в том и только в том случае, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от значения C.

IF (<a, b, c’> AND <a, b’, c>) THEN (<a, b, c>)

В силу симметрии атрибутов мы можем также записать правило в виде:

IF (<a, b, c’> AND <a, b’, c>) THEN (<a, b, c> AND <a, b’, c’>)

Лемма Фейджина о «двойственности» многозначной зависимости:

В отношении r {A, B, C} выполняется MVD A→→B в том и только в том случае, когда выполняется MVD A→→C.

Доказательство достаточности условия леммы.
Пусть ∃ MVD A→→B.
r - отношение,
∀ a : a (A ∈ Hr) ∈ t - значение атрибута A ∈ Hr в некотором кортеже тела r,
{b} – множество значений атрибута B ∈ Hr, взятых из всех кортежей тела Vr, в которых значением атрибута A является a.
Предположим, что для этого значения a MVD A→→C не выполняется.
Это означает, что существуют такое допустимое значение c атрибута C и такое значение b∈{b}, что кортеж <a, b, c>∉Vr.
Но это противоречит наличию MVD A→→B. Следовательно, если выполняется MVD A→→B, то выполняется и MVD A→→C.
Необходимость условия леммы доказывается симметрично.

MVD A→→B и A→→C представляют в форме A→→B|C

Многозначные и функциональные зависимости

FD - частный случай MVD, когда множество значений зависимого атрибута состоит из одного элемента.

если $\exists$ FD $A \rightarrow B$ (r,Hr{A,B,C}), то => $\exists$ MVD $A \rightarrow\rightarrow B|C$

Если <a,b,c1> $\in$ r AND <a,b1,c> $\in$ r, тогда в силу $A \rightarrow B$ =>
b1 = b, то есть <a,b1,c> = <a,b,c> =>
<a,b,c> $\in$ r =>
$\exists$ $A \rightarrow\rightarrow B|C$ по определению.

Обратное – не верно.
Многозначная зависимость является обобщением понятия функциональной зависимости.

Теорема Фейджина о декомпозиции отношения с многозначной зависимостью

r отношение на Hr {A, B, C}
Отношение r декомпозируется без потерь на проекции {A, B} и {A, C} тогда и только тогда, когда для него выполняется MVD A→→B|C.

r = r1 NATURALJOIN r2, r1=r PROJECT{AB}, r2=r PROJECT{AC} ⇔ MVD A→→B|C

[согласно определению MVD A→→B : IF (<ab1c> ∈ r AND <abc1> ∈ r) THEN (<abc> ∈ r)]

Доказательство **необходимости** условия теоремы (т.е. ⇒)
Левая часть выполняется, т.е. r = r1 NATURALJOIN r2 (декомпозиция без потерь)
Докажем что в таком случае выполняется MVD A→→B |C.
Пусть <abc1> ∈ r и <ab1c> ∈ r =>
По определению проекций <ab> ∈ r1, <ac> ∈ r2 =>
<abc> ∈ r1 NATURALJOIN r2, если исходная декомпозиция – без потерь =>
<abc> ∈ r – а это и есть определение MVD A→→B.
Чтд, необходимость доказана.

Доказательство **достаточности** условия теоремы (т.е. ⇐).
Пусть имеется многозначная зависимость A→→B|C.
Докажем, что декомпозиция r = r1 NATURALJOIN r2 является декомпозицией без потерь.
Доказательство аналогично теореме Хитта и разбито на две части.
Первая – включение r ⊆ r1 NATURALJOIN r2 – то есть надо доказать что любой кортеж исходного отношения принадлежит результату - доказывается так-же как у Хитта на основе определения операций проекции и естественного соединения и выполняется всегда для любого состояния r.
Докажем вторую половину, а именно что r1 NATURALJOIN r2 ⊆ r.
Пусть <abc> ∈ r1 NATURAL JOIN r2 **=>** <ab> ∈ r1, <ac> ∈ r2 **=>** <abc1> ∈ r и <ab1c> ∈ r **=>**
по определению многозначной зависимости => <abc> ∈ r.
Включение доказано. **Достаточность доказана**. Теорема Фейджина полностью доказана.

Четвертая нормальная форма (4NF)

Переменная отношения r находится в **четвертой нормальной форме** (**4NF**) в том и только в том случае, когда она находится в BCNF, и все MVD r являются FD с детерминантами – возможными ключами отношения r.

4NF является BCNF, в которой многозначные зависимости вырождаются в функциональные.

СЛУЖ_ПРО_ЗАДАН не находится в 4NF, поскольку MVD СЛУ_N→→ПРО_N |СЛУ_ЗАДАН не являются функциональными и детерминант не является возможным ключом отношения.

СЛУЖ_ПРО_N и СЛУЖ_ЗАДАНИЕ находятся в BCNF поскольку не содержат MVD, отличных от FD с детерминантом – возможным ключом (нет нетривиальных FD). Поэтому они находятся в 4NF.

Многозначная зависимость MVD A→→B|C называется **нетривиальной многозначной зависимостью**, если не существует функциональных зависимостей FD A→B или FD A→C.

СЛУ_N→→ПРО_N|СЛУ_ЗАДАН – пример нетривиальной многозначной зависимостью.

MVD A→→B называется **тривиальной** если либо B $\in$ A, либо A $\cup$ B = Hr.
Тривиальная MVD всегда удовлетворяется.
При B $\in$ A она вырождается в тривиальную FD.
При A $\cup$ B = Hr MVD верна по определению.

N-декомпозируемые отношения

Будем называть N-**декомпозируемым** отношением отношение, которое может быть декомпозировано без потерь на N проекций.

СЛУЖ_ПРО_ЗАДАН (ключ Hr и отсутствуют нетривиальные MVD)

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
34	1	B
34	2	A
41	1	A

СЛУЖ_ПРО

СЛУ_N	ПРО_N
34	1
34	2
41	1

ПРО_ЗАДАН

ПРО_N	СЛУ_ЗАДАН
1	A
1	B
2	A

СЛУЖ_ЗАДАН

СЛУ_N	СЛУ_ЗАДАН
34	A
34	B
41	A

СЛУЖ_ПРО NATURAL JOIN ПРО_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
34	1	B
34	2	A
41	1	A
41	1	B

r != r1 NATURALJOIN r2
r != r2 NATURALJOIN r3
r != r1 NATURALJOIN r3

=> по Фейджину – нетривиальные MVD отстутствуют

СЛУЖ_ПРО NATURAL JOIN СЛУЖ_ЗАДАН
<34,2,B> - лишний
ПРО_ЗАДАН NATURAL JOIN СЛУЖ_ЗАДАН
<41,2,A> - лишний

НО
r = r1 NATURALJOIN r2 NATURALJOIN r3

Это говорит о том, что между атрибутами этого отношения имеется некоторая зависимость, но эта зависимость не является ни функциональной, ни многозначной зависимостью.

Зависимость проекции/соединения

Обозначим СПЗ = СЛУЖ_ПРО_ЗАДАН, СП = СЛУЖ_ПРО, ПЗ = ПРО_ЗАДАН, СЗ = СЛУЖ_ЗАДАН

СПЗ = СП NATURAL JOIN ПЗ NATURAL JOIN СЗ
⇔
IF (<сп> ∈ СП AND <пз> ∈ ПЗ AND <сз> ∈ СЗ) THEN <спз> ∈ СПЗ

Ограничение, обеспечивающее возможность восстановления без потерь, определенное на исходном отношении будет выглядеть следующим образом:

IF (<спз'> ∈ СПЗ AND <сп'з> ∈ СПЗ AND <с'пз> ∈ СПЗ) THEN <спз> ∈ СПЗ

Если служащий «С» участвует в проекте «П» выполняя какое-либо задание,
и в проекте «П» выполняется задание «З» каким-либо служащим,
и служащий «С» выполняет задание «З» в каком-либо проекте,
ТО служащий «С» выполняет задание «З» в проекте «П».

Зависимость проекции/соединения (**Project-Join Dependency – PJD**)

Отношение r на Hr{A,B,…Z} (составные, перекрывающиеся атрибуты).
В переменной отношения r удовлетворяется **зависимость проекции/соединения *(AB… Z)** тогда и только тогда, когда любое допустимое значение r можно получить путем естественного соединения проекций этого значения на атрибуты AB…Z.
r = r{A} NATURALJOIN r{B} NATURALJOIN … NATURALJOIN r{Z}
При этом данная зависимость должна выполняться для любого значения тела отношения, удовлетворяющего всем наложенным на него ограничениям предметной области.

Аномалии, вызываемые наличием зависимости проекции/соединения

В СПЗ_2 ∃ PJD *({СЛУ_N, ПРО_N}, {ПРО_N, СЛУ_ЗАДАН}, {СЛУ_N, СЛУ_ЗАДАН})

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	B
34	2	A

IF (<спз'> ∈ СПЗ AND <сп'з> ∈ СПЗ AND <с'пз> ∈ СПЗ) THEN <спз> ∈ СПЗ

- **Добавление кортежей**.
  СПЗ_2 добавить <41,1,A>,

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	B
34	2	A
41	1	A

НЕ УДОВЛЕТВОРЯЕТ ограничению!

34	1	A

  Кортежи <34,1,B>, <34,2,A> и <41,1,A> дают нам пары 34-1, 34-A, 1-A =>
  Ограничение целостности требует включения в отношение картежа <34,1,A>
  [добавление <34,1,A> не нарушает ограничения и не требует других добавлений]
- **Удаление кортежа**.
  Если из СПЗ_2 удаляется кортеж <34,1,A>, то нужно удалить и кортеж <41,1,A>. Поскольку в соответствии с ограничением целостности наличие второго кортежа означает наличие первого.
  [удаление <41,1,A> не нарушает ограничения и не требует других удалений]

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	B
34	2	A

СЛУЖ_ПРО

СЛУ_N	ПРО_N
34	1
34	2

ПРО_ЗАДАН

ПРО_N	СЛУ_ЗАДАН
1	B
2	A

СЛУЖ_ЗАДАН

СЛУ_N	СЛУ_ЗАДАН
34	A
34	B

+ <41,1,A>

СЛУЖ_ПРО

СЛУ_N	ПРО_N
34	1
34	2
41	1

ПРО_ЗАДАН

ПРО_N	СЛУ_ЗАДАН
1	B
2	A
1	A

СЛУЖ_ЗАДАН

СЛУ_N	СЛУ_ЗАДАН
34	A
34	B
41	A

СЛУЖ_ПРО_ЗАДАН = СЛУЖ_ПРО NATURAL JOIN ПРО_ЗАДАН NATURAL JOIN СЛУЖ_ЗАДАН

СЛУ_N	ПРО_N	СЛУ_ЗАДАН
34	1	A
34	1	B
34	2	A
41	1	A

Связь многозначной зависимости и зависимости проекции/соединения

Зависимость проекции/соединения фактически является обобщением понятия многозначной зависимости, введенной нами для четвертой нормальной формы.

Теорема Фейджина (формулировка для зависимости проекции/соединения).

Отношение r(ABC) удовлетворяет зависимости проекции/соединения *(AB, AC) тогда и только тогда, когда имеется многозначная зависимость A->>B|C.

r(ABC) == (r PROJECT(AB)) NATURAL JOIN (r PROJECT (AC)) ⇔ *(AB,AC) ⇔ A->>B|C

Многозначная зависимость - частный случай зависимости проекции-соединения, т.е., если в отношении имеется многозначная зависимость, то имеется и зависимость проекции-соединения на 2 соответствующих атрибута.

Обратное (для любого количества атрибутов), неверно.

Тривиальные и нетривиальные PJD

Зависимость проекции/соединения *(AB…Z) в отношении r называется **подразумеваемой возможными ключами** в том и только в том случае, когда каждый составной атрибут AB…Z является суперключом r, т.е. включает хотя бы один возможный ключ r.

Зависимость проекции/соединения *(AB…Z) в отношении r называется **тривиальной**, если хотя бы один из составных атрибутов AB…Z совпадает с заголовком r.

Нетривиальные PJD, подразумеваемые возможными ключами, существуют во всех отношениях с арностью, большей двух, первичный ключ которых не совпадает с заголовком отношения.

Примеры:
- СЛУЖ_ПРО_ЗАДАН (СЛУ_N - первичный ключ)
  (служащий может работать только в одном проекте и над одним заданием)
  то ∃ PJD *({СЛУ_N, ПРО_N}, {СЛУ_N, СЛУ_ЗАДАН})
- r(ABCD) (A – первичный ключ == A->B,A->C,A->D == PR(AB), PR(AC), PR(AD))
  то ∃ PJD *({AB},{AC},{AD})

Нетривиальные PJD, подразумеваемые возможными ключами неинтересны с точки зрения проектирования базы данных, поскольку они не порождают аномалии обновления.

Пятая нормальная форма

Отношение r находится в **пятой нормальной форме**, или в **нормальной форме проекции/соединения** (**5NF**, **PJ/NF, Project-Join Normal Form**) в том и только в том случае, когда каждая нетривиальная PJD в r подразумевается возможными ключами r.

5NF является «окончательной» нормальной формой, которой можно достичь в процессе нормализации на основе проекций.
У отношения, находящегося в 5NF, отсутствуют аномалии обновлений, которые можно было бы устранить путем его декомпозиции.
Дальнейшая нормализация отношения в 5NF бессмысленна.

Актуальность процесса нормализации

Первый аспект: Реляционная модель данных != SQL модель данных
Подавляющее большинство современных баз данных и средств управления ими опирается на модель данных SQL, которая не тождественна реляционной модели.

(*например, таблица модели SQL может содержать мультимножества строк и не иметь ключа*).

Но если потребовать от проектируемой SQL-базы данных наличия хотя бы одного возможного ключа для каждой таблицы целевой базы данных, то все методы нормализации становятся применимы.
Таким образом, SQL-ориентированную базу данных можно проектировать как реляционную базу данных, если должным образом ограничить используемые средства модели данных SQL.

В дальнейшем под «**реляционными**» базами данных будут пониматься именно SQL-ориентированные базы данных, не противоречащие требованиям реляционной модели данных.

Актуальность процесса нормализации

Второй аспект: актуальность «нормализации»

Основные цели процесса нормализации:

- избежать избыточности хранения данных;
- устранить аномалии обновления отношений.

Хорошо нормализованные реляционные базы данных в значительной степени способствуют росту эффективности приложений. И на первых шагах развития это было особенно актуально.

Теория реляционных баз данных и методы их проектирования активно развиваются уже более 25 лет. И ситуация и в области аппаратного и в области программного обеспечения не стоит на месте.

Однако и сейчас наиболее частой схемой использования СУБД остается работа в информационных системах оперативной обработки транзакций
(On-Line Transaction Processing – OLTP).

Информационные системы оперативной обработки транзакций
On-Line Transaction Processing – OLTP

операционные банковские системы,
системы учета,
системы заказа и резервирования,
и многие другие.

Общая характеристика OLTP систем

- количество транзакций очень велико,
- выполняются транзакции параллельно,
- сложность транзакций низкая,
- необходима высокая эффективность отката транзакции,
- транзакции на вставку/удаления/модификацию превышают запросы на выборку,
- большая часть планируемых запросов известна еще на стадии проектирования.

Вывод:
чем выше уровень нормализации данных в OLTP-приложениях, тем они быстрее и надежней. Возможны отступления для обеспечения эффективности заранее известных запросов, которые критичны для работы приложений.

Системы категории оперативной аналитической обработки
On-Line Analytical Processing – OLAP

OLAP - обобщенный термин, характеризующий принципы построения систем, предназначенных для нахождения зависимостей между данными, для проведения динамического анализа данных и тому подобных задач.
К **OLAP** системам можно отнести системы:
- поддержки принятия решений – Decision Support System (DSS),
- хранилища данных – Data Warehouse,
- системы интеллектуального анализа данных – Data Mining,
- …

OLAP-приложения оперируют с большими массивами данных и характеризуются следующими признаками:

- добавление в систему новых данных происходит относительно редко крупными блоками;
- данные, добавленные в систему, как правило, никогда не удаляются;
- перед загрузкой данные могут проходить подготовительную обработку;
- запросы к системе являются нерегламентированными и достаточно сложными;
- скорость выполнения запросов важна, но не столь критична как в OLTP системах.

Базы данных OLAP-приложений обычно представлены в виде **гиперкубов -** конструкций, каждое измерение которой представляет собой некоторую характеристику данные, а в ячейках самого гиперкуба хранятся значения этих данных.

Физически гиперкуб может быть построен на основе специальной многомерной модели данных – **Multidimensional OLAP** (**MOLAP**) или средствами реляционной модели данных – **Relational OLAP** (**ROLAP**).

В системах OLAP, использующих реляционную модель данных, данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Аномалии обновления мало влияют на работу систем.

Технологически правильным считается для OLAP систем поддерживать отдельную базу данных. Источником данных для такой системы обычно служат различные OLTP системы.

Эффективность работы таких систем обычно сильно зависит от того насколько правильно они спроектированы. Обычно первым шагом проектирования проводят процедуру нормализации схемы системы до максимально возможного уровня нормализации, а вторым шагом проводят процесс денормализации из соображения повышения эффективности выполнения запросов.

Структуры внешней памяти

СУБД хранит во внешней памяти следующие разновидности объектов:

- **тела отношений** – основная содержательная пользовательская часть базы данных;
- **служебные отношения-каталоги –** информация, связанная с именованием объектов базы данных и их конкретными свойствами (схема отношения, ограничения).
- **индексы** - управляющие структуры для обеспечения ускоренного доступа (создаются по инициативе пользователя (администратора) или самой СУБД из соображений повышения эффективности выполнения запросов);
- **журнальная информация -** поддерживающая избыточность данных для удовлетворения потребности в надежном хранении данных;
- **дополнительная служебная информация** - информация необходимая СУБД для нормального функционирования (распределения свободной памяти и т.п.).

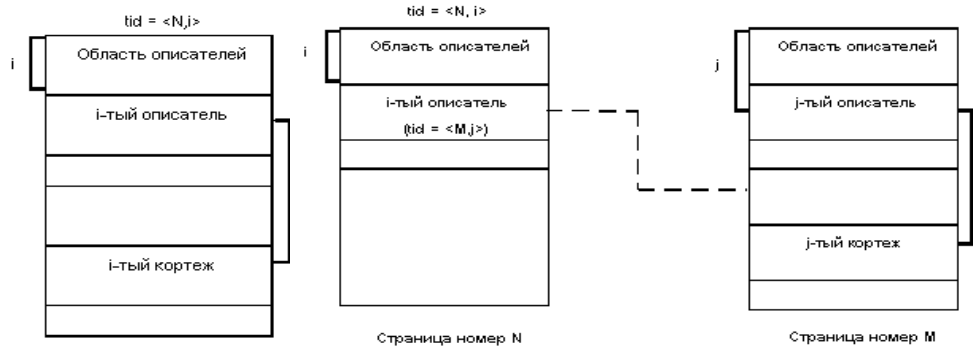
Подходы к физическому хранению отношений

Покортежное хранение отношений.
Единицей физического хранения информации является кортеж.
+ быстрый доступ к целому кортежу,
- много дублирующейся информации,
- лишние обмены с внешней памятью, если нужна часть кортежа.

Хранение отношения по столбцам,
Единицей хранения является столбец отношения с исключенными дубликатами.
+ минимизированы затраты внешней памяти, (за счет исключения дубликатов).
+ возможность использования значений столбца для оптимизации операций соединения.
- сборка целого кортежа (или его части) требует существенных затрат.

Типовой, унаследованной от System R (исследовательская СУБД середины 70-х г. в IBM первый SQL -> IBM DB2), структурой хранения данных является представление внешней памяти как набора сегментов, каждый сегмент состоит из страниц.
Страницы обычно классифицируют на страницы данных и страницы индексной информации. Иногда непосредственно в страницах внешней памяти сохраняют журнальную информацию и «длинные» данные.

Покортежное хранение отношений.
Типовая структура страницы данных:



Каждый кортеж обладает уникальным идентификатором (tid) (tuple identifier), не изменяемым все время существования кортежа. *tuple – кортеж (набор взаимосвязанных величин)*
tid - пара <номер страницы, индекс описателя кортежа в странице>.

Переразмещение кортежа в другую страницу памяти не меняет исходный tid этого кортежа. Меняется только описатель в оригинальной странице, который будет содержать новый tid, указывающий на реальное положение кортежа в новой странице.
Дальнейшее переразмещение кортежа не увеличивают уровень косвенной адресации.

Проблема «длинных» данных
- хранение длинных данных в отдельных (вне базы данных) файлах с заменой данного в кортеже на имя соответствующего файла.
- хранение длинных данных в отдельном наборе страниц внешней памяти, связанном физическими ссылками.
- хранение длинных данных в виде B-деревьев последовательностей байтов в отдельном наборе страниц внешней памяти.

Накладные расходы

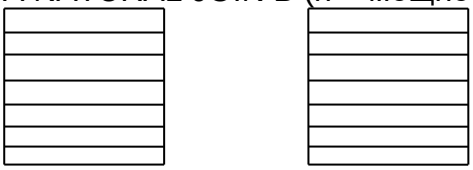
Индексы

Индексы – называют специальные конструкции в СУБД, обеспечивающие механизмы эффективного прямого доступа к произвольному кортежу отношения по значению некоторого ключа и/или последовательного просмотра кортежей в порядке возрастания или убывания значения ключа.

Общей идеей организации индексов является хранение упорядоченного списка значений ключа с привязкой к каждому значению ключа списка идентификаторов кортежей.

При выполнении многих операций языкового уровня требуется сортировка отношений в соответствии со значениями некоторых атрибутов (например NATURAL JOIN).

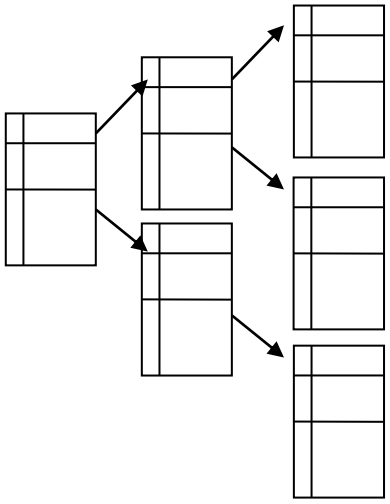
A NATURAL JOIN B (n = мощность отношения A и B)



При отсутствии индекса сложность операции n^2
При наличии индекса в отношении B сложность операции n*log(n)
При наличии индекса в отношении A и B сложность операции 2*n

Различные способы организации индекса отличаются **способом поиска ключа**.

Организация индексов на основе техники B-деревьев



B-дерево - это сбалансированное сильно ветвистое дерево во внешней памяти.

Сбалансированность означает, что длина пути от корня дерева к любому его листу практически одинакова.
Ветвистость дерева - это свойство каждого узла дерева ссылаться но большое число узлов-потомков.

Физическая организация B-дерева представляет собой древовидная мульти-списочная структура внутренних и листовых страниц внешней памяти.

Листовая страница имеет следующую структуру:
(< key_i , $TIDs_i$ > …)

- $key_1 < key_2 < \ldots < key_n$
- $TIDs_i = \{ tid_{i1}, tid_{ij}, tid_{im} \}$ – tid всех кортежей в которых key_i
- все листовые страницы связаны одно- или двунаправленным списком.

Внутренняя страница имеет следующую структуру:
(< key_i , N_i >…)

- $key_1 <= key_2 <= \ldots <= key_n$
- N_i – ссылка на внутреннюю или листовую страницу в которой находятся ключи с значениями лежащими в интервале [key_i … key_{i+1}]

Поиск в B-дереве - это прохождение от корня к листу в соответствии с заданным значением ключа. За счет сильной ветвистости и сбалансированности дерева, для выполнения поиска по любому значению ключа потребуется проход на одинаковую небольшую глубину – т.е. фактически одно и то же (небольшое) число обменов с внешней памятью.
В сбалансированном дереве, с размером внутренней страницы в **n** ключей, для хранении **m** записей потребуется глубина $\mathbf{log_n m}$.

Автоматическое поддержание свойства сбалансированности B+ деревьев

При занесении новой записи выполняется:

- Поиск листовой страницы.
- Считывание листовой страница в буфер (размер больше размера страницы) и его модификация (вставка нового значения).
- Если после вставки размер используемой части буфера не превосходит размера страницы, то операция завершается и буфер может быть немедленно вытолкнут во внешнюю память.
- Если возникло переполнение буфера, то выполняется операция расщепление страницы. Запрашивается новая страница внешней памяти и используемая часть буфера разбивается пополам и вторая половина записывается во вновь выделенную страницу.
- Для обеспечения доступа к заново заведенной страницы, модифицируется внутренняя страница, являющуюся предком ранее найденной страницы. В соответствующее место осуществляется вставка ссылки на вновь созданную страницу. Операция осуществляется в буфере оперативной памяти где снова может произойти переполнение и данная стадия может повторится на более высоком уровне дерева.
- Предельным случаем является переполнение корневой страницы B-дерева. В этом случае она тоже расщепляется на две, и заводится новая корневая страница дерева, т.е. глубина дерева увеличивается на единицу.

Автоматическое поддержание свойства сбалансированности B+ деревьев
При удалении записи выполняются следующие действия:

- Поиск записи по ключу.
- Считывание листовой страница в буфер и реальное удаление записи.
- Если после выполнения удаления размер занятой в буфере области оказывается таковым, что его сумма с размером занятой области в листовых страницах, являющихся предыдущей или последующей для данной страницы, больше, чем размер страницы, операция завершается.
- Иначе производится слияние с предыдущей или последующей страницей, т.е. в буфере производится модификация образа соседней страницы, содержащей общую информацию из данных страниц. Ставшая ненужной листовая страница заносится в список свободных страниц, а также модифицируются сквозные ссылки листовых страниц.
- Для обеспечения прекращения доступа к удаленной листовой странице в оперативной памяти модифицируется образ внутренней страницы, являющейся предком ранее существовавшей листовой страницы. После удаления ссылки в оперативной памяти снова может возникнуть ситуация когда будет возможно слияние модифицированной страницы с соседней и повторение данной стадии на более высоком уровне иерархии.
- Предельным случаем является полное опустошение корневой страницы дерева, которое возможно после слияния последних двух последних потомков корня. В этом случае корневая страница освобождается, а глубина дерева уменьшается на единицу.

Для уменьшения проблем потери эффективности при слишком частом выполнении операций расщепления и слияния, применяются более сложные приемы, в частности:

- **упреждающие расщепления**,
  т.е. расщепления страницы не при ее переполнении, а несколько раньше, когда степень заполненности страницы достигает некоторого уровня;
- **переливания**,
  т.е. поддержание равновесного заполнения соседних страниц;
- **слияния 3-в-2**,
  т.е. порождение двух листовых страниц на основе содержимого трех соседних.

Хеширование

Общей идеей методов хеширования является применение к значению ключа функции свертки (хэш-функции), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.
i = f(X), X – что угодно, i – обычно int индекс в некотором фиксированном интервале

В самом простом, классическом случае, свертка ключа используется как индекс элемента в массиве, содержащем множество пар ключей и tid-ов. Больше одной пары в элементе массива (которые иногда называют «**цепочки переполнения**») появляется в случае возникновения коллизий – когда различным значениям ключа соответствует одно и то-же значение свертки.

Основным требованием к хэш-функции является равномерное распределение значение свертки.

Если таблица заполнена слишком сильно или переполнена, но возникнет слишком много цепочек переполнения, и главное преимущество хеширования - доступ к записи почти всегда за одно обращение к таблице - будет утрачено.

Главным ограничением этого метода является фиксированный размер таблицы. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

В СУБД в этом случае часто применяют техники совмещающие использование B+ деревьев и хеширования.

Журнальная информация

Журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти.
В грамотно организованных системах структура (и тем более, смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление. К ведению файла журнала предъявляются особые требования по части надежности. В частности, обычно стремятся поддерживать две идентичные копии журнала на разных устройствах внешней памяти.

Служебная информация

- внутренние каталоги, описывающие физические свойства отношений: число атрибутов, их размер, типы данных, описание индексов…
- описатели свободной и занятой памяти в страницах отношения
- информация о связывании страниц внешней памяти для одного отношения.

Базы данных, часть II

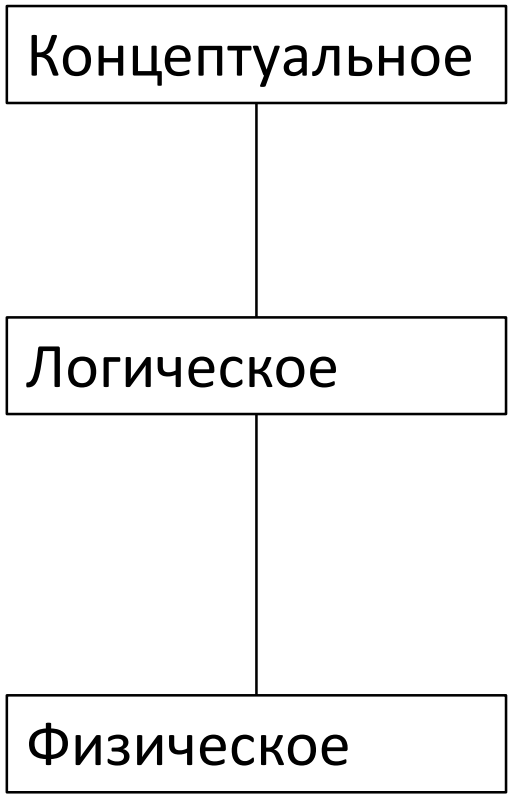
- Концептуальное проектирование баз данных
- Язык баз данных SQL
- Постреляционные модели данных
- Зачетная контрольная работа

(вопросы к зачету: Questions.doc в telegram https://t.me/+O2C1a9ctMy0zNjEy)

Лектор: Морозов Сергей Вячеславович

Концептуальное проектирование баз данных

Виды проектирования БД



Концептуальное проектирование БД – процесс создания модели информации, не зависящий от любых физических аспектов ее представления

Логическое проектирование БД – процесс создания модели информации с учетом выбранной модели организации данных, но независимо от типа целевой СУБД и других физических аспектов реализации

Физическое проектирование БД – процесс описания реализации БД на ЗУ с указанием структур хранения и методов доступа, используемых для эффективной обработки

Концептуальное проектирование баз данных

Основные определения

Предметная область – часть реального мира, рассматриваемая в определенном смысле как единое целое, сведения о которой представляют собой информационные ресурсы в аспектах создания и использования БД.

Информационная модель – способ представления понятий или объектов предметной области, описывающий существенные для данного представления совокупности объектов, их параметры, поведение и отношения между ними.

CASE-система (Computer-Aided Software Engineering) – система поддержки технологий автоматизированного проектирования, реализации и сопровождения сложных программных систем на всех этапах их жизненного цикла.

Недостатки логического проектирования реляционных БД

1. Неудобство для проектировщиков:

а) На ранних стадиях проектирования, как правило, требуется участие специалистов, хорошо знающих предметную область, но не владеющих теорией БД

б) Во многих предметных областях трудно осуществлять моделирование информации на основе плоских таблиц

2. Отсутствие наглядности:

а) Реляционная модель не предлагает какого-либо механизма для разделения объектов предметной области и связей между ними

б) Реляционная модель не обеспечивает достаточных средств для представления семантики данных

3. Невозможность автоматизации процесса проектирования:

Реляционная модель не предоставляет какие-либо формализованные средства для представления функциональных и других зависимостей, на основе которых осуществляется процесс проектирования

Достоинства информационного моделирования

• Построение наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на ранних стадиях проектирования.

• Информационная модель является важной документацией, полезной не только на стадии проектирования БД, но и при ее дальнейшей эксплуатации, сопровождении и развитии.

• На рынке представлены CASE-системы, обеспечивающие автоматизированное преобразование концептуальных схем (диаграмм) в реляционные (язык SQL).

Информационная модель Entity-Relationship

Альтернативные названия: ER-модель, модель «Сущность-Связь», диаграммы Чена, реляционная информационная модель
Предложена: 1976 г., Питер Чен
Назначение: описание моделей предметных областей с целью последующего проектирования БД
Стандарт: отсутствует
Применяемая нотация: графическая (диаграммы), множество альтернативных нотаций
Изучаемая нотация: применяется в CASE-системе Oracle

Основные понятия ER-модели

Сущность (тип сущности) – реальный или представляемый объект, информация о котором должна сохраняться и быть доступной

Атрибут сущности – любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности

Атрибут в реляционной модели: <имя атрибута, имя типа данных>
В ER-модели указание типа атрибута не является обязательным

ЧЕЛОВЕК
фио, например, Иванов И.И.
дата рождения, например, 30.06.1980
пол, например, М или Ж

Основные понятия ER-модели

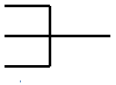
Связь – графически изображаемая ассоциация, устанавливаемая между двумя сущностями.

Графическое отображение: ненаправленная линия ——————————

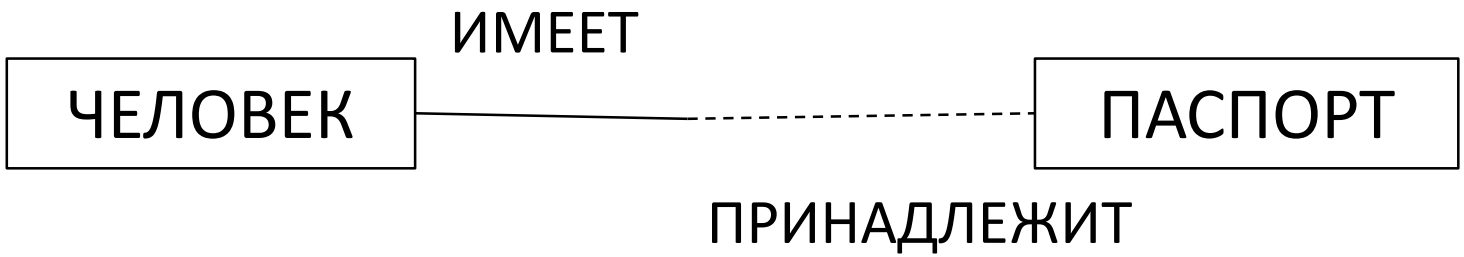
В ER-модели допускаются только бинарные связи, то есть, соединяющие две сущности или сущность саму с собою (рекурсивная связь)

Конец связи называют ролью связи в данной сущности

Характеристики роли:

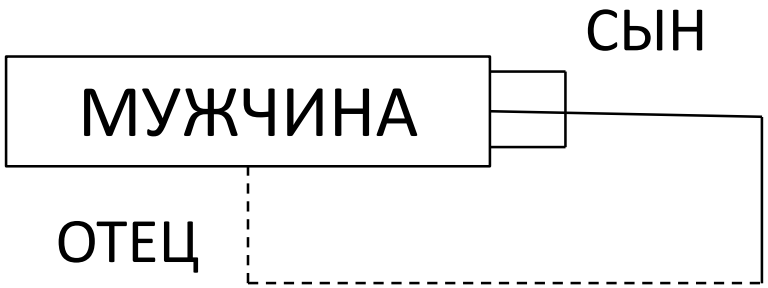
- Имя роли (указывается над линией связи вблизи соответствующего конца)
- Степень роли (допустимое количество экземпляров соответствующей сущности в данной связи):
  - Один экземпляр – одноточечный вход —
  - Много экземпляров – трехточечный вход 
- Обязательность роли:
  - Обязательная (каждый экземпляр данной сущности ДОЛЖЕН участвовать в связи) ————
  - Необязательная (каждый экземпляр данной сущности МОЖЕТ участвовать в связи) ----------

Связь «Один-к-Одному»



- Каждый человек имеет один и только один паспорт
- Каждый паспорт может принадлежать только одному человеку

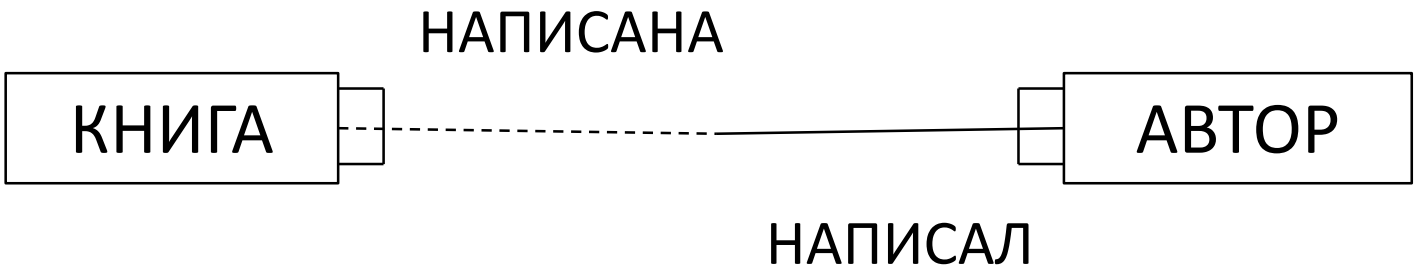
Связь «Один-ко-многим»



• Каждый мужчина является сыном одного и только одного мужчины

• Каждый мужчина может являться отцом одного или более мужчин

Связь «Многие-ко-Многим»



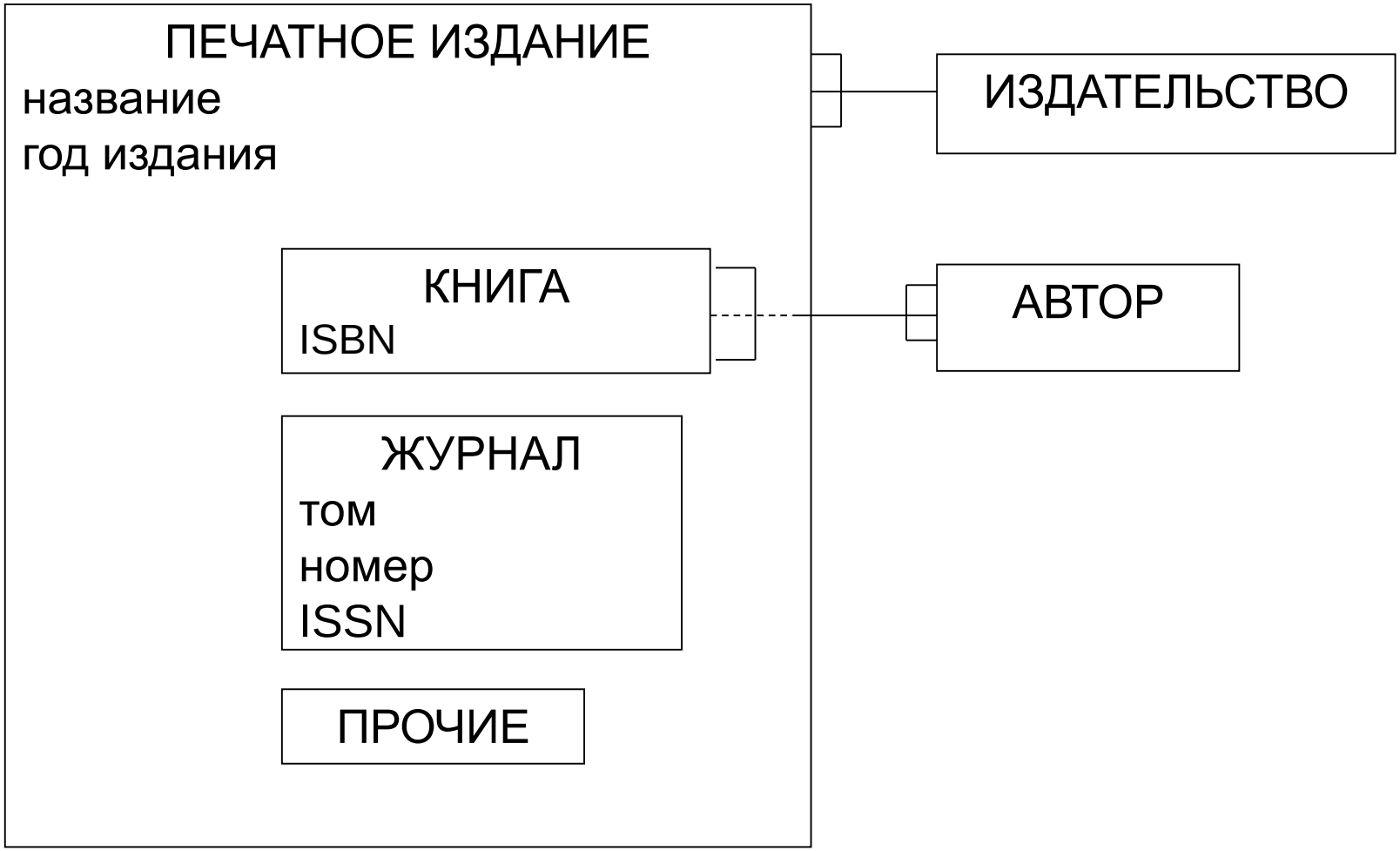
- Каждая книга может быть написана одним или более авторами
- Каждый автор принимал участие в написании одной или более книг

Наследование в ER-модели

Тип сущности (A) может быть расщеплен на несколько взаимно исключающих подтипов (B_1, B_2, …, B_n)
Тип сущности, на основе которого определяются подтипы, называется супертипом
Подтипы наследуют атрибуты и связи супертипа и могут определять собственные атрибуты и/или связи
Простым типом сущности называется тип, не являющийся подтипом и не имеющий подтипов
Правила наследования в ER-модели:

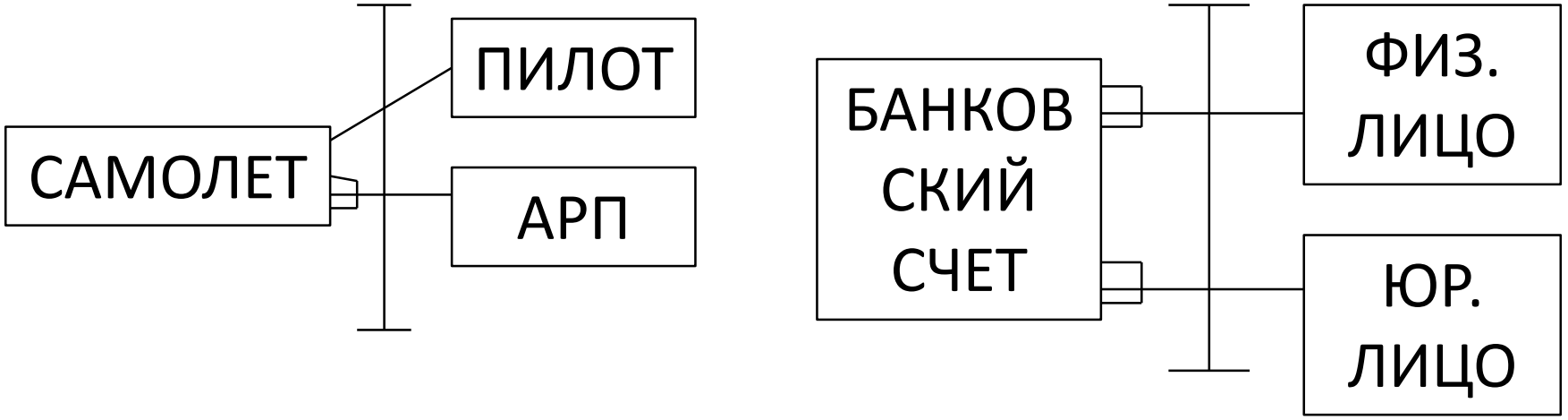
1. Включение ($\forall b \in B_i \rightarrow b \in A$, $i = 1, \ldots, n$)
2. Отсутствие собственных экземпляров у супертипа ($\forall a \in A \rightarrow a \in B_i$, $i = 1, \ldots, n$)
3. Разъединенность подтипов ($\forall b \in B_i \rightarrow b \notin B_j$, $i \neq j$)

Пример ER-модели с наследованием



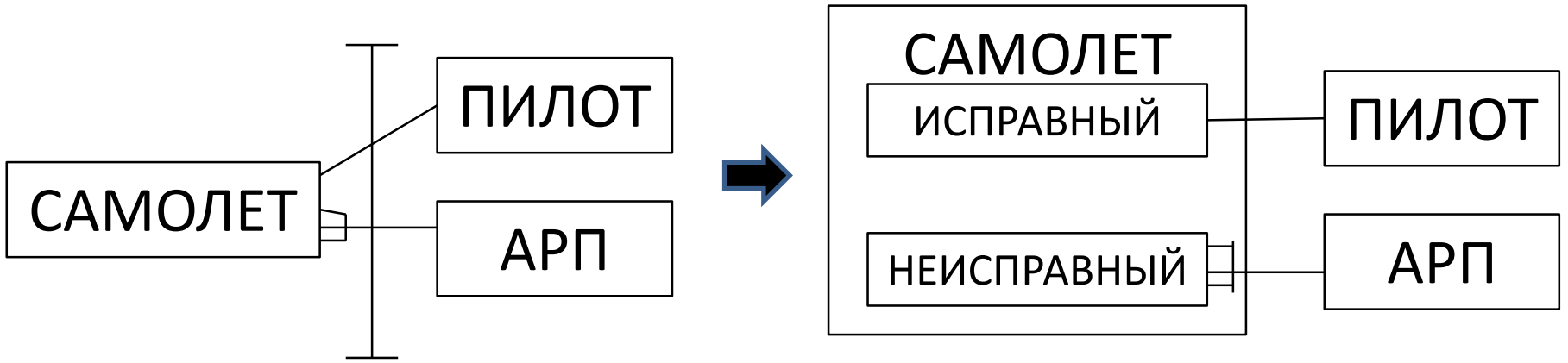
Взаимно исключающие связи

Взаимно исключающими связями называется такой набор связей одной сущности с другими, что для каждого экземпляра сущности может или должен существовать экземпляр только одной связи из данного набора

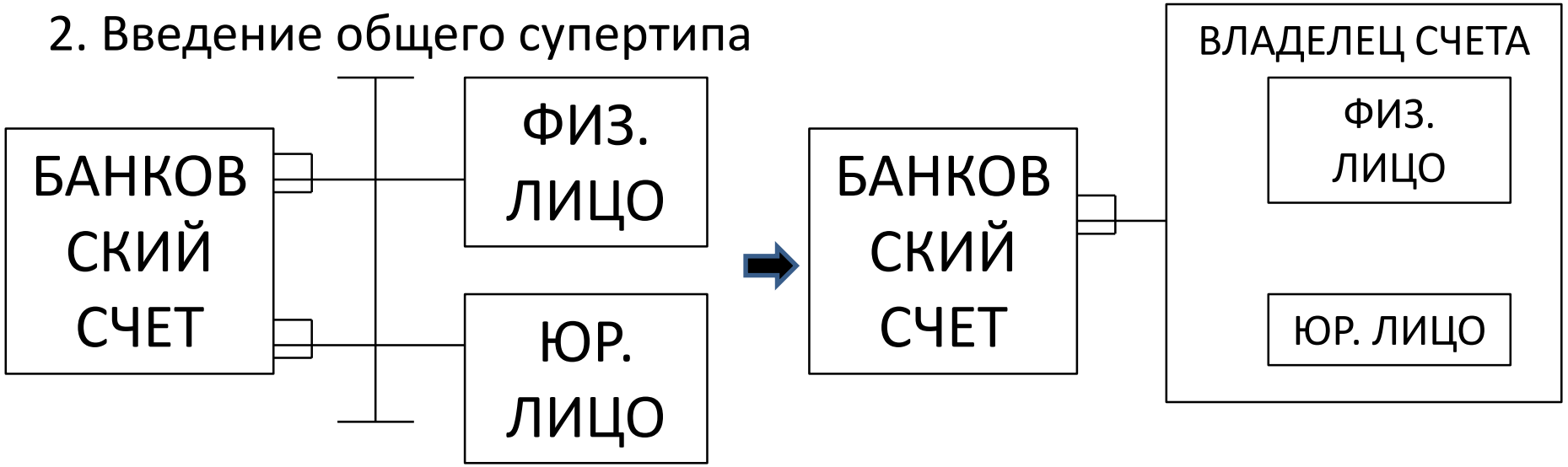


Преобразования диаграмм со взаимно исключающими связями

1. Введение подтипов



2. Введение общего супертипа



Уникальные идентификаторы экземпляров сущностей

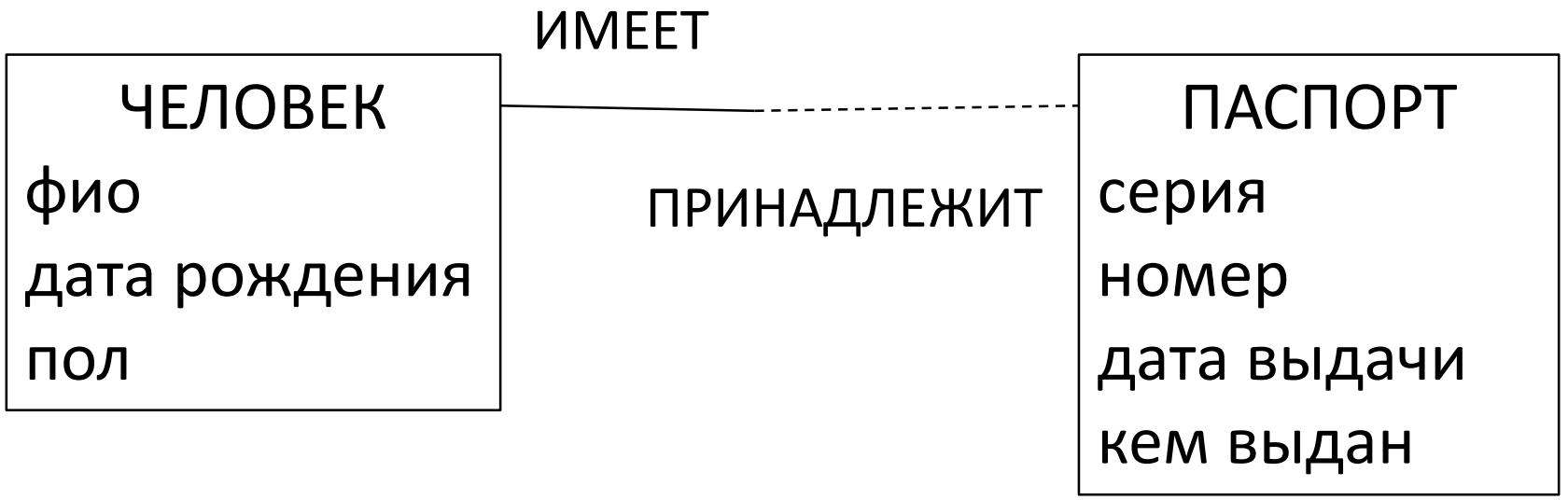
При определении сущности необходимо гарантировать, что каждый ее экземпляр является отличимым от любого другого экземпляра той же сущности. Это достигается путем введения уникальных идентификаторов.

В ER-модели экземпляр сущности может идентифицироваться только своими индивидуальными характеристиками: значениями атрибутов и экземплярами связей.

В качестве уникального идентификатора сущности проектировщик может выбрать:

- Атрибут
- Комбинацию атрибутов
- Связь
- Комбинацию связей
- Комбинацию атрибутов и связей

Выбор уникального идентификатора экземпляров сущности «Человек»

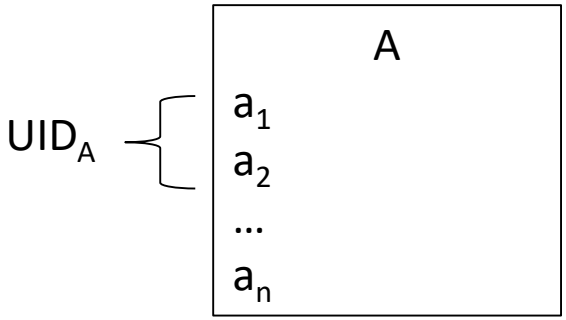


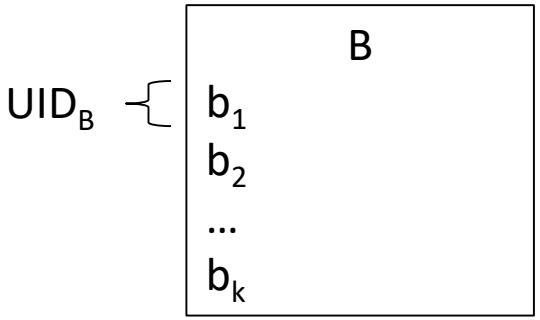
- Для БД предприятий: либо фио, либо фио + дата рождения
- Для БД крупных городов или государств: связь с паспортом, тогда в качестве УИД человека будет использоваться УИД его паспорта (серия + номер)

Алгоритм преобразования ER-модели в реляционную

1. Каждый простой тип сущности превращается в отношение. Экземплярам сущности соответствуют кортежи данного отношения.
2. Каждый атрибут сущности становится атрибутом соответствующего отношения, при этом выбирается тип для представления соответствующих данных.
3. Компоненты уникального идентификатора сущности становятся первичным ключом отношения. Если в состав уникального идентификатора входят связи, к числу атрибутов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (копия первичного ключа соответствующего отношения).

Алгоритм преобразования ER-модели в реляционную





A

a_1	a_2	…	a_n

PRIMARY KEY (a_1, a_2)

B

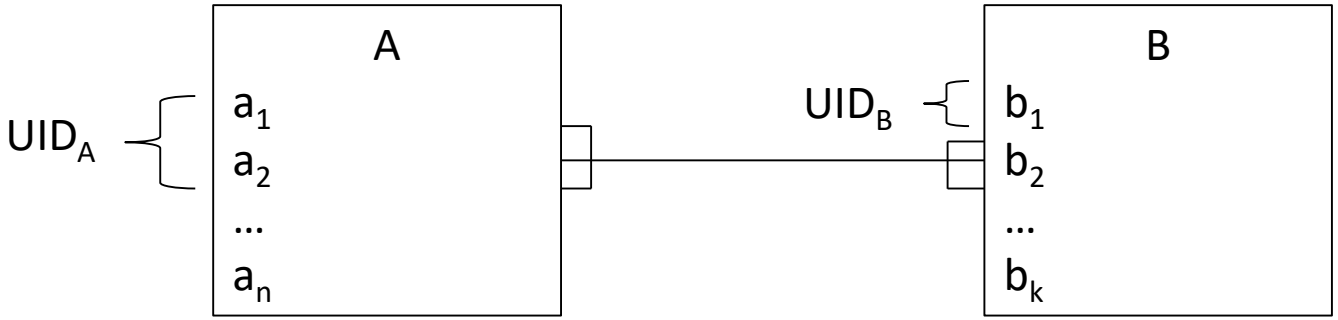
b_1	b_2	…	b_k

PRIMARY KEY (b_1)

Алгоритм преобразования ER-модели в реляционную

4. Связи становятся внешними ключами отношения; для связей «один-ко-многим» внешний ключ объявляется в отношении, соответствующем сущности на конце связи «многие»; для связей «один-к-одному» внешний ключ может быть объявлен в одном из двух отношений по желанию проектировщика, при этом данный внешний ключ должен быть определен и как возможный ключ отношения (для ограничения степени связи); если связь является необязательной, внешний ключ допускает наличие неопределенных значений.
5. Для поддержки связи «многие-ко-многим» создается дополнительное отношение с двумя (возможно составными) атрибутами, каждый их которых содержит копию первичного ключа соответствующего отношения, участвующего в данной связи.

Алгоритм преобразования ER-модели в реляционную



A

a_1	a_2	…	a_n	b_1

B

b_1	b_2	…	b_k

AB

a_1	a_2	b_1

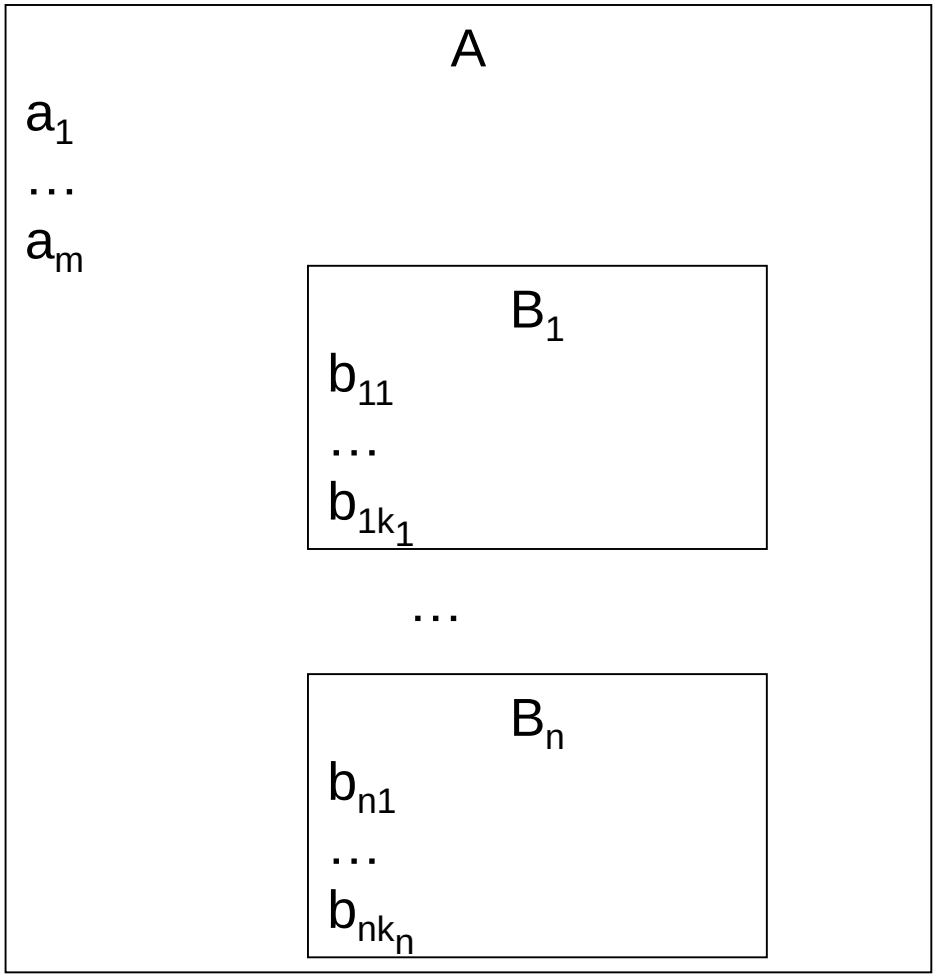
PRIMARY KEY (a_1, a_2)

PRIMARY KEY (b_1)

FOREIGN KEY (b_1) UNIQUE (b_1)

Алгоритм преобразования ER-модели в реляционную

6. Подтипы и супертипы могут быть представлены в реляционной модели двумя способами



Алгоритм преобразования ER-модели в реляционную

6а. Одно отношение для всех подтипов

t_c	a_1	…	a_m	b_{11}	…	b_{1k_1}	…	b_{n1}	…	b_{nk_n}
B_1	X	X	X	X	X	X	N	N	N	N
B_n	X	X	X	N	N	N	N	X	X	X

t_c – код типа (часть первичного ключа)

X – конкретное значение атрибута

N – неопределенное значение (NULL)

Доступ к экземплярам подтипов:

PROJECT A $\{a_1, \ldots, a_m, b_{i1}, \ldots, b_{ik}\}$ (A WHERE t_c = 'B_i')

Достоинства:

1. Соответствие логике супертипов
2. Простой способ доступа к экземплярам супертипов и не слишком сложный – к экземплярам подтипов
3. Сокращение количества отношений

Недостатки:

1. Единственное отношение – узкое место при многопользовательском доступе
2. Усложнение логики приложений БД
3. Непроизводительный расход внешней памяти (хранение NULL)

Алгоритм преобразования ER-модели в реляционную

6б. <u>Каждый подтип представляется отдельным отношением</u>

Воссоздание супертипа:

PROJECT B_1 {a_1, …, a_m} UNION … UNION PROJECT B_n {a_1, …, a_m}

Достоинства:

1. Более понятные правила работы с подтипами
2. Упрощение логики приложений

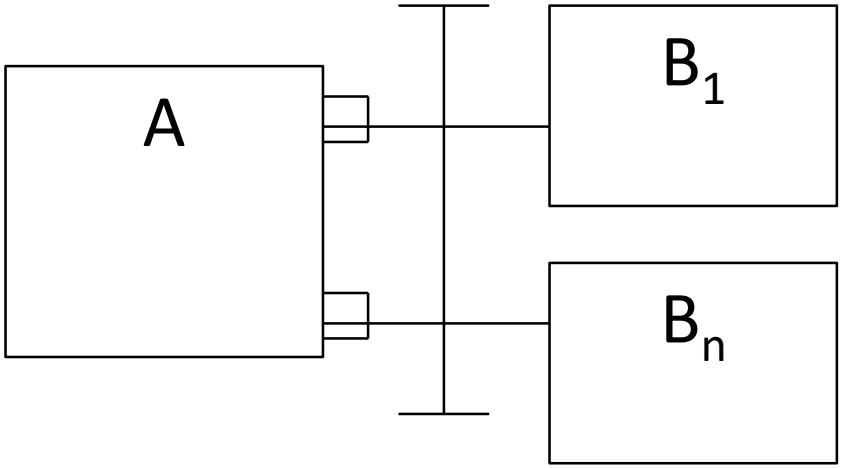
Недостатки:

1. Увеличение количества отношений
2. Усложнение доступа к экземплярам супертипа
3. В общем случае невозможность модификации экземпляров супертипа

Алгоритм преобразования ER-модели в реляционную

7. Представление взаимно исключающих связей:
7а. Преобразование модели со взаимно исключающими связями в модель с наследованием (далее см. пункт 6)

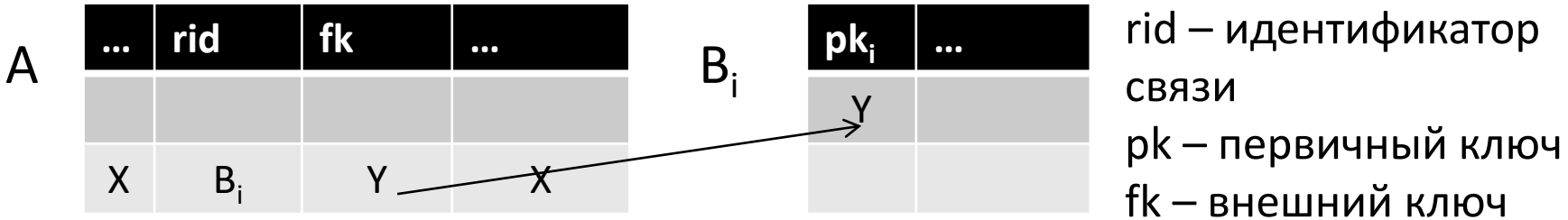
Для моделей, где связи «один-ко-многим» являются взаимоисключающими со стороны сущности со степенью роли «многие», возможно применение еще двух способов преобразования



Алгоритм преобразования ER-модели в реляционную

7. Представление взаимно исключающих связей:

7б. Общее хранение внешних ключей



Достоинства:

- Добавление небольшого количества атрибутов для представления связей

Недостатки:

- Применим при условии, что все первичные ключи сущностей B определены на одном домене;
- Сложная операция соединения: (A WHERE rid='B_i') JOIN B_i WHERE A.fk=B_i.pk_i

Алгоритм преобразования ER-модели в реляционную

7. Представление взаимно исключающих связей:

7в. Раздельное хранение внешних ключей

A

…	pk_1	…	pk_n	…
X	Y	NULL	NULL	X

B_1

pk_1	…
Y	

Достоинства:

- Применим независимо от доменов первичных ключей в B;
- Операция естественного соединения: A NATURAL JOIN B_i

Недостатки:

- Увеличение количества атрибутов для представления связей;
- Непроизводительный расход внешней памяти (хранение NULL)

Язык UML (Unified Modeling Language)

Предложен: 1996 г., Г. Буч, Д. Румбах, А. Якобсон
Назначение: моделирование разных систем (аппаратных, программных, смешанных, с участием человека, и т.д.), описание их статических (структурных) и динамических (поведенческих) свойств
Стандарт: OMG (www.omg.org), текущая версия 2.5.1 (декабрь 2017 г.)
Нотация: графическая (диаграммы), стандарт определяет 14 типов диаграмм
CASE системы: IBM/Rational Software Architect, IBM Rapsody, Borland Together, Sparx Enterprise Architect, Objecteering, Umbrello

Диаграммы классов UML: основные понятия

Диаграмма классов описывает типы объектов моделируемой системы и статические отношения различного рода, которые существуют между ними. Диаграмма классов может включать комментарии и ограничения.
Класс – именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой.
Атрибут класса – именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства (абстракция состояния объекта).
Операция класса – именованная услуга, которую можно запросить у любого объекта этого класса (абстракция поведения объекта).
Стереотип – механизм расширения семантики UML, позволяющий создавать новые элементы UML на основе существующих с учетом особенностей решаемой задачи.

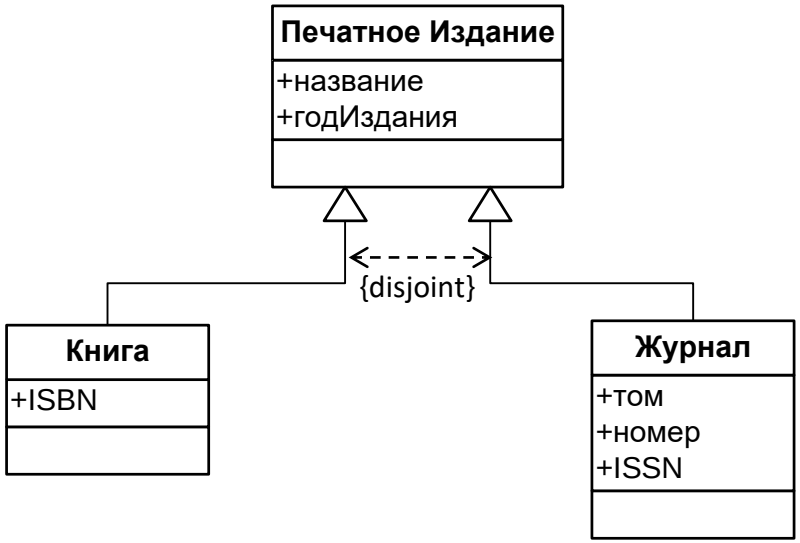
Диаграммы классов UML: пример

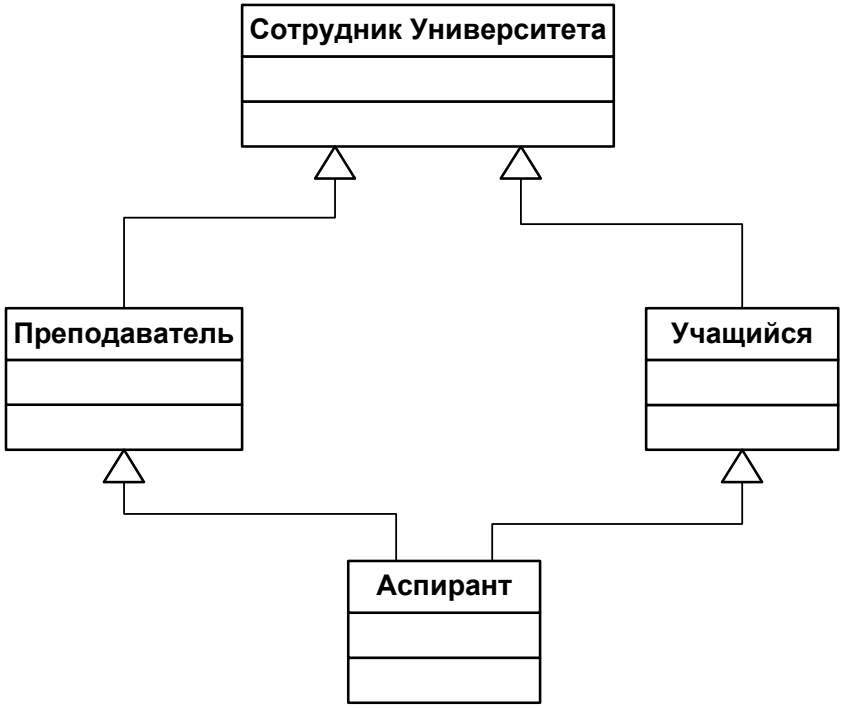
Категории связей UML

Три категории связей: зависимость, обобщение, ассоциация.
Зависимостью называется связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый. Изображение: пунктирная линия, направленная от зависимого класса.
Обобщением называется связь между общим классом (суперклассом) и более специализированной его разновидностью (подклассом). Изображение: сплошная линия с незакрашенным треугольником, направленным к суперклассу.
Правила наследования в ER-модели (см. предыдущую лекцию):

1. Включение ($\forall b \in B_i \rightarrow b \in A$, $i = 1, \ldots, n$) – **выполняется в UML**
2. Отсутствие собственных экземпляров у супертипа ($\forall a \in A \rightarrow a \in B_i$, $i = 1, \ldots, n$) – **не выполняется в UML по умолчанию** (исключение: объявление суперкласса абстрактным)
3. Разъединенность подтипов ($\forall b \in B_i \rightarrow b \notin B_j$, $i \neq j$) – **не выполняется в UML по умолчанию** (исключение: ограничение обобщения {disjoint})

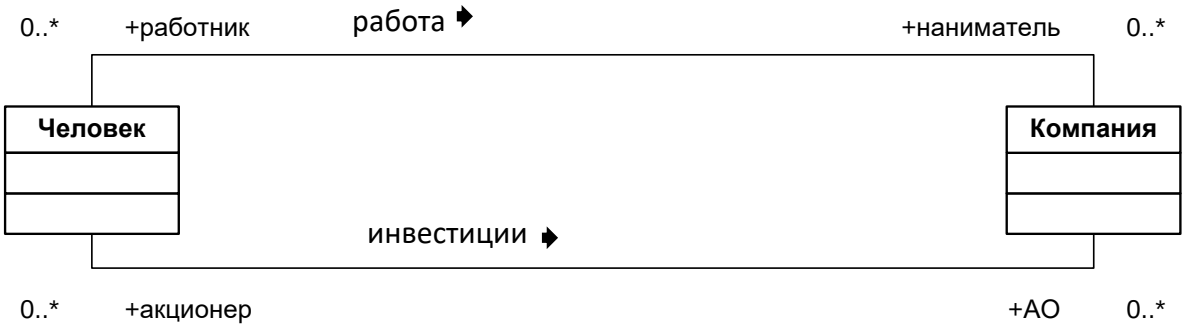
Пример с использованием связей-обобщений UML





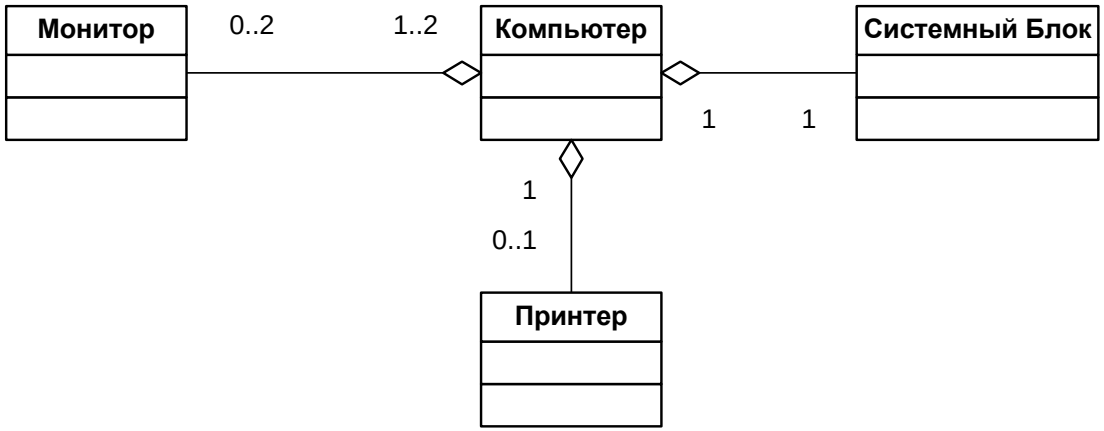
Связи-ассоциации в UML

Ассоциацией называется структурная связь между объектами одного класса и объектами другого или того же самого класса. Допускается создание n-арных ассоциаций, связывающих сразу несколько классов. Изображение: сплошная линия (в общем случае ненаправленная).
Ассоциации может быть присвоено имя, характеризующее природу связи (указывается над линией связи посередине).
Роль класса также характеризуется именем, помещаемым над линией ассоциации ближе к данному классу.
Кратность роли – характеристика, указывающая сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации. Задается числом (1), диапазоном (0..1) или списком чисел и/или диапазонов (2, 4..6, 8..*).

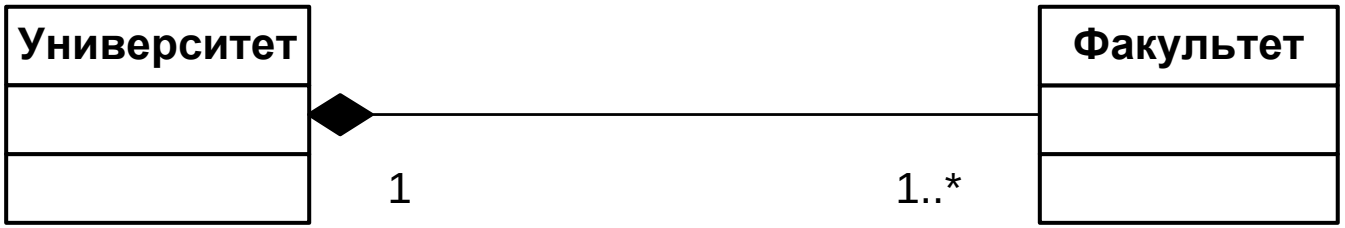


Агрегатные ассоциации в UML

Агрегатные ассоциации используются в UML для отображения отношений «часть—целое» (класс «целое» имеет более высокий концептуальный уровень, чем часть).

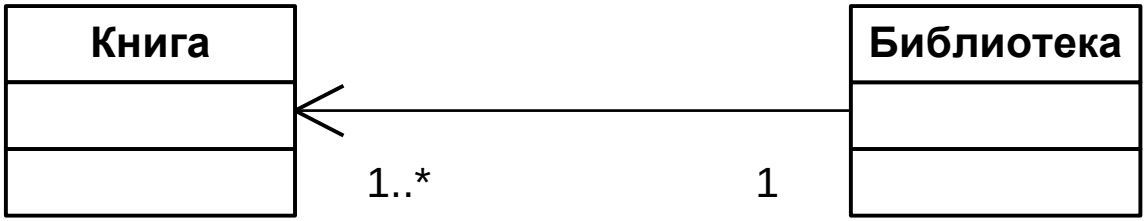


Бывают случаи, когда связь «части» и «целого» настолько сильна, что каждая часть может принадлежать одному и только одному целому и уничтожение целого приводит к уничтожению всех его частей. Агрегатные ассоциации, обладающие данным свойством, называются композициями.

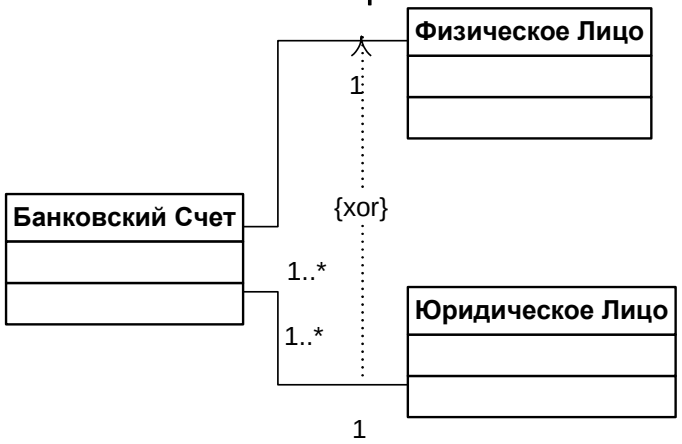


Навигация и взаимно исключающие связи

При наличии ассоциации предполагается возможность навигации между объектами, входящими в экземпляр ассоциации. По умолчанию в UML навигация может проводиться в обоих направлениях. В тех случаях, когда требуется ограничить направление навигации, на линии ассоциации ставится стрелка, указывающая требуемое направление.



Для организации взаимно исключающих связей используется ограничение ассоциации {xor}.



Рекомендации по использованию UML для концептуального проектирования реляционных БД

1. Алгоритм преобразования ER-модели в реляционную применим и к UML.
2. Операции в классах могут быть реализованы не во всякой РСУБД.
3. Связи-зависимости при преобразованиях в РСУБД не используются.
4. Старайтесь избегать множественного наследования и осторожно используйте одиночное наследование классов.
5. Эффективно в РСУБД реализуются только ассоциации «один-ко-многим».
6. Реализация ассоциаций с точно заданными кратностями ролей возможна, но требует определения дополнительных ограничений, что может привести к снижению эффективности.
7. Для РСУБД агрегатные ассоциации неестественны, композиции влияют, как правило, только на способ поддержки ссылочной целостности (каскадное удаление).
8. Для РСУБД поддержка однонаправленных ассоциаций вызывает дополнительные накладные расходы.
9. Определение большого числа ограничений может привести к снижению эффективности.

Язык OCL (Object Constraint Language)

Предложен: 1999 г., часть спецификации UML 1.3
Назначение: определение ограничений целостности данных, соответствующих моделям, представленным в терминах диаграмм UML
Стандарт: OMG (www.omg.org), текущая версия 2.4 (февраль 2014 г.)
Нотация: текстовая
Тип языка: декларативный, стандарт не определяет правила вычисления выражений, а также правила их отображения в императивные языки
CASE системы: Borland Together

Язык OCL (Object Constraint Language)

На диаграммах классов OCL может применяться для определения ограничений, описывающих пред- и постусловия операций классов, а также ограничений, являющихся инвариантами классов.

Инвариант класса – логическое выражение, при вычислении которого для любого объекта данного класса должно получаться значение true в течение всего времени существования этого объекта.

context <class_name> **inv**:
<OCL logical expression>

В выражениях OCL могут использоваться:
- Предопределенные типы данных OCL и операции над данными типами
- Типы данных, определяемые моделью UML (классы)
- Атрибуты классов и роли ассоциаций
- Операции классов, не изменяющие состояния моделируемой системы

Типы данных OCL

- Скалярные
  - **Integer**
  - **Real**
  - **Boolean**
  - **String**
- Коллекции
  - **set** (неупорядоченная коллекция, не содержащая одинаковых элементов)
  - **bag** (неупорядоченная коллекция, которая может содержать одинаковые элементы)
  - **sequence** (упорядоченная коллекция, которая может содержать одинаковые элементы)
  - **orderedSet** (упорядоченная коллекция, не содержащая одинаковых элементов)

Операции над скалярными типами OCL

Скалярный тип	Список операций
Boolean	**and**, **or**, **xor**, **not**, **implies**, **if-then-else-endif**
Integer	*, +, -, /, **abs**(), **div**(), **mod**(), **min**(), **max**(), операции сравнения
Real	*, +, -, /, **abs**(), **floor**(), **round**(), **min**(), **max**(), операции сравнения
String	**concat**(), **size**(), **substring**(), **toLower**(), **toUpper**()

Операции над объектными типами (классами UML)

- Получение значения атрибута

<объект>.<имя атрибута>

- Переход по экземпляру ассоциации

<объект>.<имя роли, противоположной по отношению к объекту>

- Вызов операции класса

<объект>.<имя операции>(<список фактических параметров>)

Литерал **self** – текущий объект класса, в котором определен инвариант

Операции над коллекциями OCL

Используемая нотация: <коллекция> -> <имя операции>(<список фактических параметров>)

Конструкторы коллекций:
select(<лог. выражение>)
reject(<лог. выражение>)
collect(<выражение>)

Кванторы:
exists(<лог. выражение>)
forAll(<лог. выражение>)

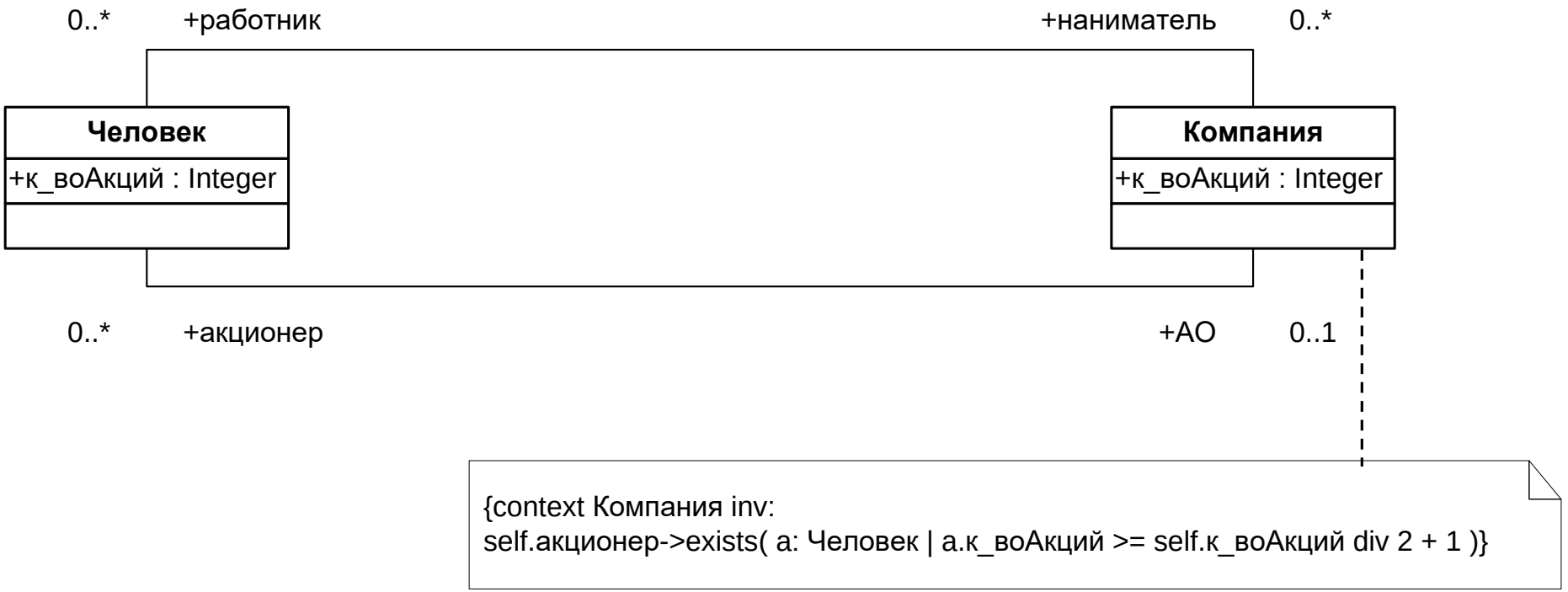
Теоретико-множественные операции:
union
intersect
-

Прочие операции:
count(<element>)
includes(<element>)
excludes(<element>)
size()
at(<индекс>)

В выражениях может использоваться конструкция iterator | body, где iterator – определение переменной, на каждом шаге принимающей значение текущего элемента коллекции, а body – выражение, в котором участвует iterator

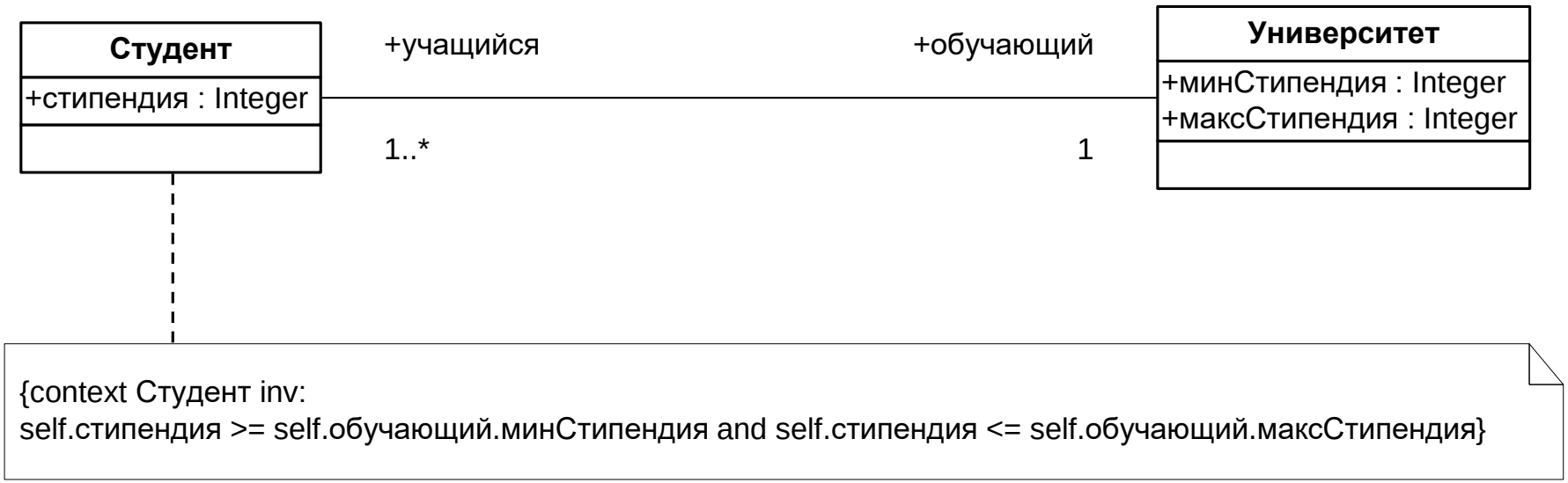
Примеры записи ограничений OCL

В компании должен быть акционер, обладающий контрольным пакетом акций



Примеры записи ограничений OCL

Размер стипендии студента должен находиться в диапазоне, допустимом в университете



Плюсы и минусы использования OCL при проектировании реляционных БД

Плюс:

1. OCL позволяет формально и однозначно определять ограничения целостности данных в терминах концептуальной модели БД

Минусы:

1. Строгость синтаксиса, неочевидность ряда конструкций и текстовая нотация языка OCL противоречат наглядности и интуитивной ясности диаграмм UML
2. OCL не поддерживается большинством CASE-систем, скорее всего придется преобразовывать инварианты OCL в ограничения целостности SQL вручную

Краткое сравнение ER-диаграмм и диаграмм классов UML

- ER-модель концептуально проще UML, в ней меньше понятий, терминов, вариантов применения.
- Поскольку UML может использоваться для унифицированного объектно-ориентированного моделирования всего чего угодно, в этом языке содержится масса различных понятий, терминов и вариантов использования, избыточных с точки зрения проектирования реляционных БД.
- В контексте проектирования реляционных БД структурные методы проектирования, основанные на использовании ER-диаграмм, и объектно-ориентированные методы, основанные на использовании языка UML, различаются, главным образом, лишь терминологией.
- Как правило, выбор средств для информационного моделирования – это вопрос вкуса и сложившихся обстоятельств.
- Неоспоримым преимуществом использования диаграмм классов UML при проектировании реляционных БД является то, что они позволяют оставаться в общем контексте UML и применять другие виды диаграмм для проектирования приложений БД.

Язык баз данных SQL

Предложен: 1974 г., исследовательский проект IBM System R

Первоначальное название: SEQUEL (Structured English QUEry Language)

Стандарт: ANSI, ISO (ISO-9075)

SQL-ориентированные коммерческие СУБД:

- 1979 г. Oracle
- 1981 г. IBM SQL/DS (до конца 90-х)
- 1983 г. IBM DB2
- 1985 г. INFORMIX SQL (в 2001 г. приобретена IBM)
- 1987 г. Sybase SQL Server
- 1989 г. Microsoft SQL Server (по технологии Sybase)

Наиболее распространенные некоммерческие реализации:

- MySQL (с 2010 г. принадлежит Oracle)
- PostgreSQL

Возможности языка SQL

- Формулирование запросов к БД
- Манипулирование данными (создание, модификация, удаление)
- Определение и манипулирование схемой БД
- Определение ограничений целостности данных
- Определение представлений (виртуальных таблиц)
- Определение структур физического уровня, поддерживающих эффективное исполнение запросов
- Авторизация доступа к данным
- Управление транзакциями

Стандартизация языка SQL

Поко-ление	Год	Стандарт	Основные изменения
1	1986	SQL-86	Первоначальная версия
	1989	SQL-89	Четкая стандартизация синтаксиса и семантики операторов выборки данных, манипулирования данными и определения ограничений целостности
2	1992	SQL-92	Манипулирование схемой, транзакциями, сессиями, подключениями, динамический SQL
	1995	-	SQL/CLI (Call-Level Interface)
	1996	-	SQL/PSM (Persistent Stored Modules)
3	1999	SQL:1999	ОО расширения, регулярные выражения, триггеры, рекурсивные запросы
	2003	SQL:2003	Java, XML, поддержка OLAP
	2006	-	Переработана поддержка XML (XQUERY)
	2008	SQL:2008	Улучшена поддержка OLAP, XQUERY, расширение триггеров
	2011	SQL:2011	Улучшена поддержка темпоральных данных
	2016	SQL:2016	JSON, полиморфные табличные функции, поиск строк по шаблону
	2019	-	Поддержка многомерных массивов

Структура стандарта SQL (2003 - 2016)

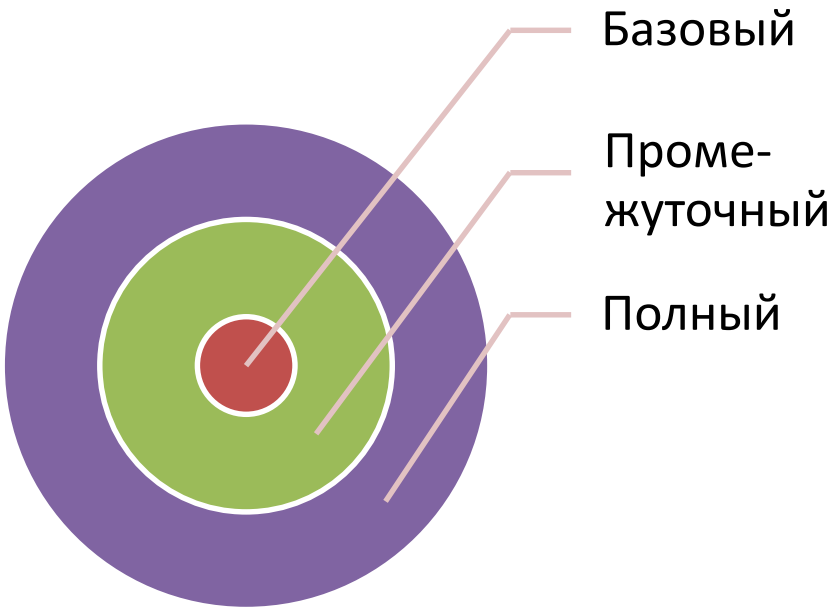
Том 1: Framework
Концептуальная структура стандарта
Том 2: Foundation
Синтаксис, семантика, правила связывания для процедурных языков программирования
Том 3: Call-Level Interface
Интерфейс уровня вызовов (основа SQL-ориентированных API)
Том 4: Persistent Stored Modules
Описание языка SQL/PSM
Том 9: Management of External Data
Языковые средства взаимодействия с внешними данными
Том 10: Object Language Bindings
Правила связывания для объектно-ориентированных языков программирования
Том 11: Information and Definition Schemas
Описание информационной схемы (хранение описателей данных)
Том 13: SQL Routines and Types Using the Java Programming Language
Использование SQL совместно с языком программирования Java
Том 14: XML-Related Specifications
Языковые средства работы с XML документами
Том 15 (2019): Multi-dimensional Arrays
Поддержка работы с многомерными массивами

Критика языка SQL

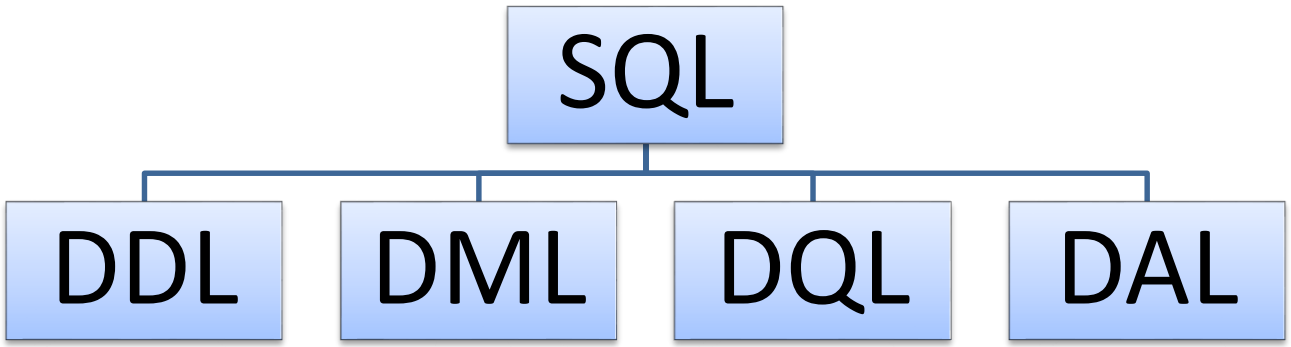
1. Противоречие принципам реляционной модели данных
   - Терминология (РМД – отношение, атрибут, кортеж; SQL – таблица, столбец, строка)
   - Отношение РМД – множество кортежей; таблица SQL – мультимножество строк
   - Заголовок таблицы SQL – упорядоченное множество столбцов (в РМД – множество атрибутов)
   - Во временных таблицах, порождаемых запросами SQL, допускаются безымянные столбцы, а также дублирование имен столбцов
   - В SQL значения NULL допускаются для возможных ключей, существует несколько способов сопоставления внешнего и возможного ключей
2. Чрезмерная избыточность языка
3. Отступления от стандарта в реализациях (диалекты SQL)

Структура языка SQL

С точки зрения разработчиков СУБД



С точки зрения программиста приложений БД



Условное разделение на подъязыки

Типы данных SQL

• Данные, хранящиеся в таблицах SQL, являются типизированными
• СУБД должна отслеживать, чтобы в каждом столбце каждой строки таблицы присутствовали только допустимые для соответствующих типов данных значения
• NULL является допустимым значением для любого типа данных SQL

Булевский тип:
BOOLEAN
Литералы: FALSE, TRUE, UNKNOWN
SQL:2003: «В этой спецификации не проводится различие между NULL-значением булевского типа данных и истинностным значением UNKNOWN, являющимся результатом вычисления предиката, условия поиска или булевского выражения. Они могут использоваться взаимозаменяемо и означают в точности одно и то же»

Числовые типы данных

Точные числовые типы

Целые числа:

SMALLINT, INTEGER, BIGINT

точность представления определяется в реализации

Числа с фиксированной точкой:

NUMERIC(p, s), DECIMAL(p, s)

p – точность (число сохраняемых десятичных цифр): строгая для NUMERIC, минимально допустимая для DECIMAL

s – масштаб (число десятичных цифр в дробной части)

допустимые значения p, s определяются в реализации ($p \geq s$)

значения по умолчанию: s=0, p – определяется в реализации

Приближенные числовые типы

REAL, DOUBLE PRECISION, FLOAT(p)

p – число значащих цифр в мантиссе

точность представления для первых двух типов и максимальное значение p определяются в реализации

Типы символьных и битовых строк

CHAR(x), VARCHAR(x), CLOB(z)

x, z – количество символов в строке: фиксированное для CHAR, максимально допустимое для VARCHAR и CLOB

допустимые значения x, z определяются в реализации (z >> x),

z может указываться в виде nK, nM, nG

вид литералов: ‘abcdef’ или “FBCDE”

набор допустимых символов определяется в реализации (ASCII как минимум)

BIT(x), BIT VARYING(x), BLOB(z)

в отличие от строк состоят из произвольных байтов, не обязательно кодирующих символы

вид литералов: B‘01001101101’ или X’79CA83’

Типы даты, времени

DATE, TIME(p), TIMESTAMP(p), TIME WITH TIMEZONE(p), TIMESTAMP WITH TIMEZONE(p)
p – точность представления долей секунды, максимальное значение определяется в реализации (≥ 6), значение по умолчанию p=0 для TIME, p=6 для TIMESTAMP
Допустимое значение секунд варьируется от 00 до 61

Форматы литералов: DATE ’yyyy-mm-dd’, TIME ’hh:mm-ss:ff…f’, TIMESTAMP ‘yyyy-mm-dd hh:mm-ss:ff…f’
Временная зона задается в виде +hh:mm или –hh:mm

Функции, возвращающие текущие дату и время: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP

Типы временных интервалов

INTERVAL start(p) [TO end(q)]
start, end: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
Поддерживаемые интервалы:
YEAR, YEAR TO MONTH, MONTH
DAY, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND
HOUR, HOUR TO MINUTE, HOUR TO SECOND
MINUTE, MINUTE TO SECOND, SECOND
p – точность представления лидирующего поля, q – точность долей секунды (применяется, если end=SECOND), максимальные значения определяются в реализации, значения по умолчанию p=2, q=6

Формат литералов: INTERVAL ‘1:35’ HOUR TO MINUTE

Преобразование типов данных в SQL

- Поддерживается явное и неявное преобразование значений одного типа данных к другому
- Правила неявного преобразования не сильно отличаются от тех, что существуют в языках программирования
    - Точный числовой тип приводится к другому точному числовому типу с большей точностью или к приближенному числовому типу
    - Приближенный числовой тип приводится к другому приближенному числовому типу с большей точностью
    - Строковый тип фиксированной длины приводится к другому строковому типу фиксированной или переменной длины с большим допустимым количеством символов
    - Строковый тип переменной длины приводится к другому строковому типу переменной длины с большим допустимым количеством символов
- Явное преобразование осуществляется с помощью оператора CAST(expression AS datatype)
- Подробные правила выполнения оператора CAST см. в учебной литературе

Домены в SQL

Домен – базовый тип данных + ограничение области определения значений
Домен является долговременно хранимым именованным объектом схемы БД

Оператор определения домена:
CREATE DOMAIN domain_name AS datatype [DEFAULT value] [constraint_list]
DEFAULT value – определение значения по умолчанию (литерал базового типа, либо значение NULL, либо вызов предопределенной функции, возвращающей литеральное значение базового типа)
constraint_list – список ограничений области определения значений, все ограничения в списке связываются логической функцией AND

Пример:

```
CREATE DOMAIN SALARY AS NUMERIC( 10, 2 )
        DEFAULT 10000.00
        CHECK( VALUE BETWEEN 10000.00 AND 10000000.00 )
        CONSTRAINT SAL_NOT_NULL CHECK( VALUE IS NOT NULL );
```

Домены в SQL

Оператор изменения определения домена:
ALTER DOMAIN domain_name action
Допустимы 4 действия: SET DEFAULT value, DROP DEFAULT, ADD CONSTRAINT [constraint_name] constraint_expression, DROP CONSTRAINT constraint_name
Если к моменту исполнения ALTER DOMAIN ADD CONSTRAINT существуют столбцы таблиц, определенные на данном домене, текущие значения которых противоречат новому ограничению, то СУБД должна отвергнуть этот оператор
Отменить можно только именованное ограничение

Пример:
ALTER DOMAIN SALARY SET DEFAULT 15000.00;
ALTER DOMAIN SALARY DROP CONSTRAINT SAL_NOT_NULL;

Домены в SQL

Оператор отмены определения домена:
DROP DOMAIN domain_name { RESTRICT | CASCADE }

DROP DOMAIN RESTRICT отвергается, если домен использован в определении некоторого столбца таблицы (базовой или виртуальной) или в определении ограничения целостности

DROP DOMAIN CASCADE выполняется всегда по следующим правилам:

- Уничтожаются все ограничения целостности и виртуальные таблицы, в определениях которых задействован данный домен
- Столбцы базовых таблиц, определенные на данном домене, преобразуются к его базовому типу и наследуют значение по умолчанию и все ограничения уничтожаемого домена

Таблицы в SQL

1. Базовые – реально хранимые в БД таблицы
2. Порождаемые – таблицы, формируемые и существующие во время выполнения запросов
3. Представления (виртуальные) – таблицы, существование которых поддерживается алгоритмически

Оператор определения базовой таблицы:
CREATE TABLE table_name (column_list [constraint_list])
column_definition ::= column_name { datatype | domain } [DEFAULT value] [constraint_list]

Данный оператор создает соответствующие описатели в схеме БД и выделяет область во внешней памяти для хранения данных.

Значение по умолчанию для столбца

Действующее значение по умолчанию для столбца определяется следующим образом:

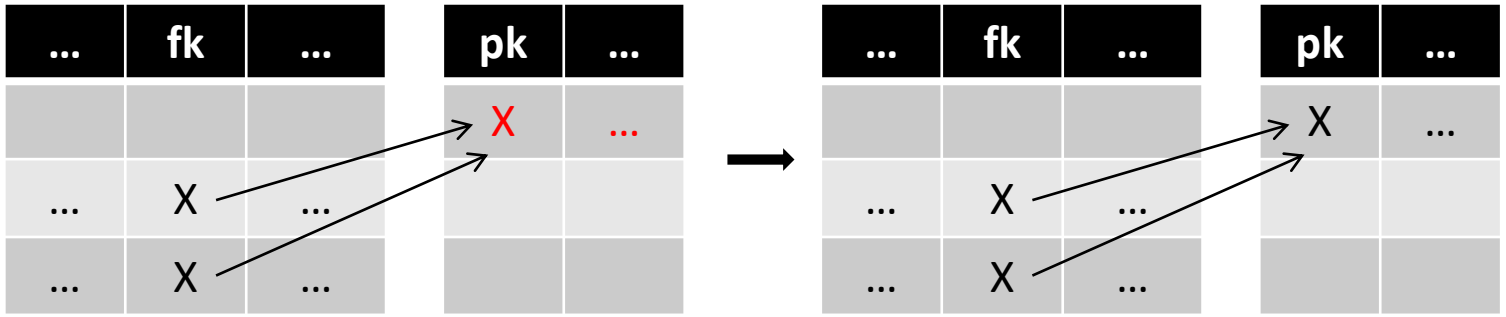
1. Если в определении столбца явно присутствует раздел DEFAULT, то действует значение указанное в данном разделе
2. Иначе, если столбец определен на домене и в определении этого домена явно присутствует раздел DEFAULT, то действует значение, указанное в данном разделе
3. Иначе, если значение NULL не является запрещенным для данного столбца, то оно является значением по умолчанию
4. Иначе, значение по умолчанию у данного столбца отсутствует (в этом случае при любой вставке новой строки в данную таблицу значение для данного столбца должно быть явно задано)

Ограничения целостности столбцов и таблиц

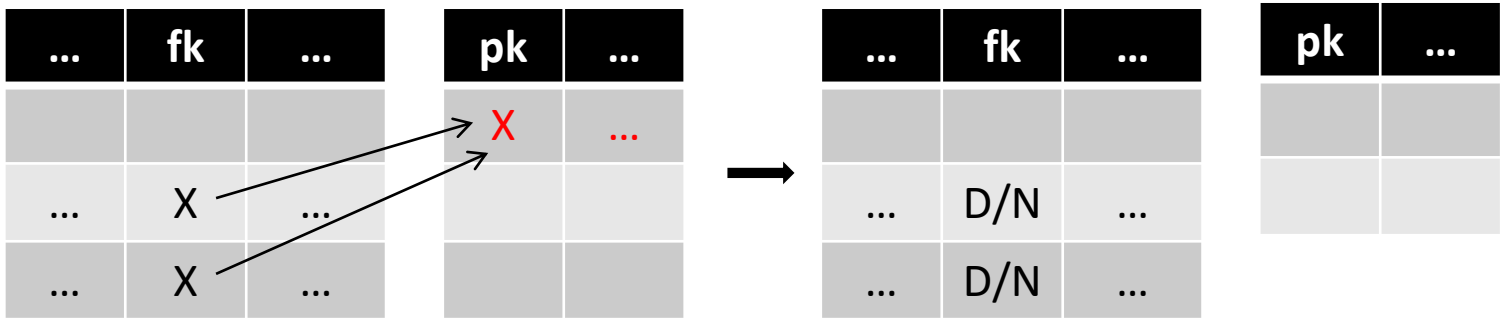
1. NOT NULL (только для столбцов, эквивалентное табличное ограничение для столбца C: CHECK(C IS NOT NULL)
2. Проверочное ограничение: CHECK(logical_expression), попытка обновления таблицы нарушает проверочное ограничение в том и только том случае, если результат вычисления логического выражения равен FALSE
3. Ограничение первичного ключа: PRIMARY KEY(column_list), допускается не более одного определения первичного ключа в таблице, PRIMARY KEY подразумевает NOT NULL для каждого из столбцов, упоминаемых в определении первичного ключа
4. Ограничение возможного ключа: UNIQUE(column_list), NULL допускается, но может встретиться только один раз в столбце возможного ключа
5. Ограничение внешнего ключа:
   для столбца REFERENCES foreign_table_name [(foreign_table_column)] [ON DELETE action] [ON UPDATE action]
   для таблицы FOREIGN KEY column_list REFERENCES foreign_table_name [(foreign_table_column_list)] [ON DELETE action] [ON UPDATE action]

Ссылочные действия (на примере DELETE)

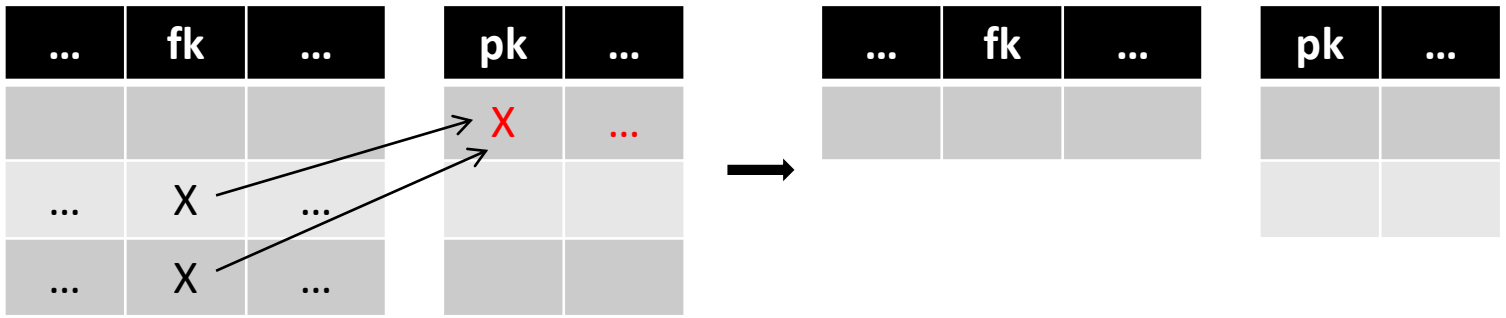
NO ACTION
или
RESTRICT



SET DEFAULT
или
SET NULL

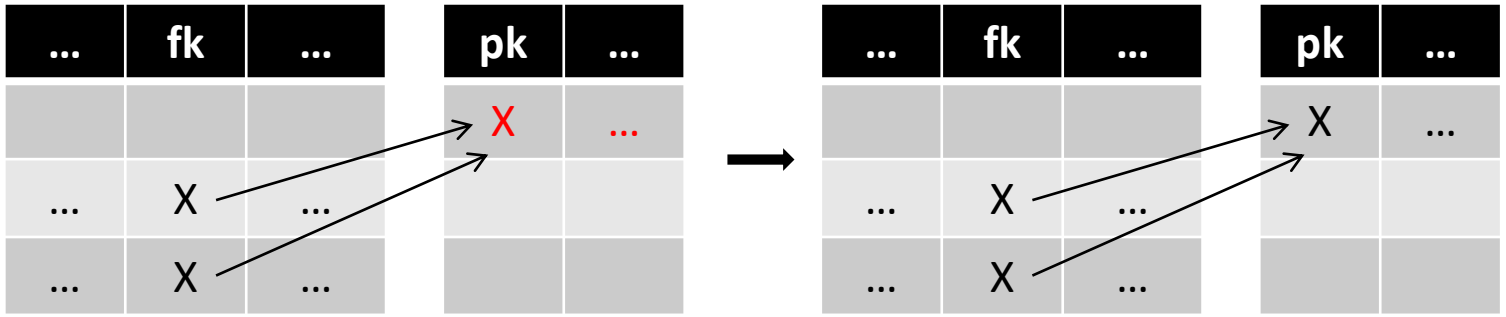


CASCADE

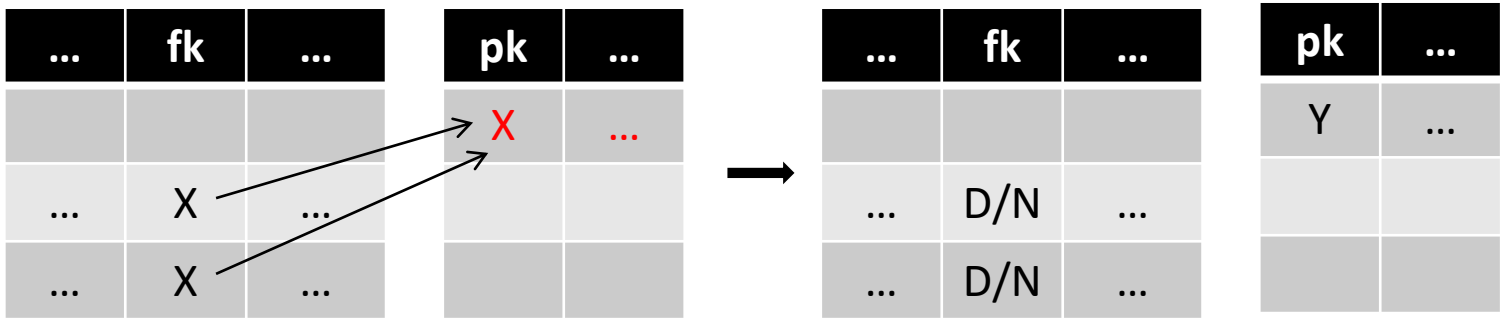


Ссылочные действия (на примере UPDATE)

NO ACTION
или
RESTRICT



SET DEFAULT
или
SET NULL



CASCADE

Изменение определения базовой таблицы

ALTER TABLE table_name action

Возможны следующие действия:
ADD COLUMN column_name { datatype | domain } [DEFAULT value]
[constraint_list]
ALTER COLUMN column_name SET DEFAULT value
ALTER COLUMN column_name DROP DEFAULT
DROP COLUMN column_name { RESTRICT | CASCADE }
ADD CONSTRAINT [constraint_name] constraint_expression
DROP CONSTRAINT constraint_name [{ RESTRICT | CASCADE }]

Добавление или удаление столбцов таблицы

ADD COLUMN column_name { datatype | domain } [DEFAULT value] [constraint_list]
Отличие от определения столбца при создании таблицы: для существующих строк в таблице значением данного столбца является значение по умолчанию
→ добавляемый столбец обязан иметь явно или неявно определенное значение по умолчанию
Пример: ALTER TABLE EMPLOYEE ADD COLUMN EMP_BONUS SALARY DEFAULT NULL CONSTRAINT BONUS CHECK(VALUE < EMP_SALARY);

DROP COLUMN column_name { RESTRICT | CASCADE }
RESTRICT – действие отвергается, если столбец является единственным в таблице или имя столбца используется в определениях ограничений (внешних по отношению к данной таблице) или виртуальных таблиц (представлений)
CASCADE – действие выполняется, если столбец не является единственным в таблице, при этом отменяются определения ограничений и представлений, в которых задействован удаляемый столбец
Пример: ALTER TABLE DEPARTMENT DROP COLUMN DEPT_EMP_NUM CASCADE;

Изменение набора табличных ограничений

ADD CONSTRAINT [constraint_name] constraint_expression
Если к моменту исполнения ALTER TABLE ADD CONSTRAINT в таблице существуют строки, противоречащие новому ограничению, то СУБД должна отвергнуть этот оператор
Пример: ALTER TABLE DEPARTMENT ADD CONSTRAINT DEPT_EMP_NUMBER CHECK((SELECT COUNT(*) FROM EMPLOYEE WHERE DEPT_ID = EMPLOYEE.DEPT_ID) <= 100);

DROP CONSTRAINT constraint_name [{ RESTRICT | CASCADE }]
RESTRICT и CASCADE имеют смысл только для ограничений первичного или возможного ключей (RESTRICT – действие отвергается, если существует хотя бы один внешний ключ, ссылающийся на отменяемый; CASCADE – действие выполняется всегда, при этом отменяются определения всех ссылающихся внешних ключей)

Отмена определения базовой таблицы

DROP TABLE table_name { RESTRICT | CASCADE }

DROP TABLE RESTRICT отвергается, если таблица задействована в определении представлений или ограничений целостности (не считая собственных табличных ограничений)

DROP TABLE CASCADE выполняется всегда по следующим правилам:

- Уничтожаются все ограничения целостности и представления, в определениях которых задействована данная таблица
- Уничтожаются все строки, хранящиеся в данной таблице, а также определения ее столбцов и табличных ограничений (таблица перестает существовать)

Ограничения целостности БД

Иерархия ограничений целостности языка SQL

Дополнительные ограничения базы данных – общие ограничения целостности

Определение:

CREATE ASSERTION constraint_name constraint_expression

Отмена:

DROP ASSERTION constraint_name

Пример:

CREATE ASSERTION DEPT_MNG_CONSTR
CHECK(NOT EXISTS(SELECT * FROM EMPLOYEE EMP, DEPARTMENT DEPT WHERE EMP.EMP_ID = DEPT.DEPT_MANAGER AND CURRENT_DATE - EMP.EMP_BDATE < INTERVAL ‘30’ YEAR));

Базовые средства манипулирования данными

- Средства манипулирования данными присутствуют в прямом, встраиваемом и динамическом SQL
- К базовым средствам манипулирования данными относятся:
  - Оператор вставки строк (INSERT)
  - Оператор модификации строк (UPDATE)
  - Оператор удаления строк (DELETE)

Оператор вставки строк

INSERT INTO table_name [(column_list)] VALUES row_constructor_list

• Если список столбцов пропущен, то предполагается, что используется список из имен всех столбцов таблицы, указанных в том порядке, в котором они были описаны в операторе CREATE TABLE

• В списке столбцов должны быть обязательно указаны те столбцы, для которых не задано значение по умолчанию (явно или неявно)

• Порядок перечисления значений столбцов в конструкторе строки должен совпадать с порядком перечисления столбцов в соответствующем списке

• В качестве значений столбцов могут использоваться литералы, DEFAULT, NULL, а также скалярные запросы, результат выполнения которых состоит из единственной строки, включающей единственный столбец

Пример:

```
INSERT INTO EMPLOYEE VALUES
  ROW( 319, ‘Иванов И.И.’, ‘1980-07-19’, DEFAULT, NULL, NULL ),
  ROW( 320, ‘Петров П.П.’, ‘1983-05-09’, (SELECT EMP_SALARY FROM EMPLOYEE
WHERE EMP_ID = 200), 12, NULL );
INSERT INTO EMPLOYEE( EMP_ID, EMP_NAME, EMP_BDATE ) VALUES
  ROW( 321, ‘Сидоров С.С.’, ‘1962-12-10’ );
```

Оператор вставки строк – вставка результата запроса

Определим таблицу, хранящую информацию о временных сотрудниках организации, проходящих испытательный срок:

```
CREATE TABLE EMPLOYEE_TEMP (
        EMPT_ID INTEGER PRIMARY KEY,
        EMPT_NAME VARCHAR( 200 ) NOT NULL,
        EMPT_BDATE DATE NOT NULL,
        EMPT_ENTRY_DATE DATE NOT NULL
);
```

Переведем временных сотрудников, чей испытательный срок превысил 3 месяца, в основной штат:

```
INSERT INTO EMPLOYEE( EMP_ID, EMP_NAME, EMP_BDATE ) ( SELECT EMPT_ID, EMPT_NAME, EMPT_BDATE  FROM EMPLOYEE_TEMP WHERE CURRENT_DATE – EMPT_ENTRY_DATE >= INTERVAL ‘3’  MONTH );
```

Оператор модификации строк

UPDATE table_name SET column_value_assignment_list WHERE logical_expression
column_value_assignment := column_name = value_expression

• Для всех строк таблицы с указанным именем вычисляется логическое условие. Строки, для которых значением этого условия является TRUE, считаются подлежащими модификации (S_m – множество модифицируемых строк)

• Каждая строка $s \in S_m$ подвергается модификации таким образом, что значение каждого столбца этой строки, указанного в списке модификации, заменяется значением, указанным в правой части соответствующего элемента списка. Если новое значение задается оператором запроса, в который входят имена модифицируемых столбцов, то под значениями этих столбцов в запросе понимаются их значения до модификации

Пример. Перевести всех служащих, выполняющих проект с номером 5, в отдел 12 и повысить им заработную плату на 5000 руб.

```
UPDATE EMPLOYEE SET DEPT_ID = 12, EMP_SALARY = EMP_SALARY + 5000.00
WHERE PRO_ID = 5;
```

Оператор удаления строк

DELETE FROM table_name WHERE logical_expression

• Для всех строк таблицы с указанным именем вычисляется логическое условие. Строки, для которых значением этого условия является TRUE, считаются подлежащими удалению (S_d – множество удаляемых строк)

• Каждая строка $s \in S_d$ удаляется из указанной таблицы.

Пример. Уволить всех служащих, размер заработной платы которых превышает размер заработной платы начальников их отделов.

```
DELETE FROM EMPLOYEE WHERE EMP_SALARY >
            (SELECT EMP.EMP_SALARY
              FROM EMPLOYEE EMP, DEPARTMENT DEPT
              WHERE EMPLOYEE.DEPT_ID = DEPT.DEPT_ID
               AND DEPT.DEPT_MANAGER = EMP.EMP_ID);
```

Учебный пример

```
CREATE TABLE EMPLOYEE (
…
DEPT_ID INTEGER DEFAULT
NULL REFERENCES
DEPARTMENT ON DELETE SET
NULL,
…
);
```

```
CREATE TABLE DEPARTMENT (
        DEPT_ID INTEGER PRIMARY KEY,
        DEPT_EMP_NUM INTEGER NOT NULL
CHECK( VALUE <= 100 ),
        CHECK ( DEPT_EMP_NUM = ( SELECT
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID =
EMPLOYEE.DEPT_ID )),
…
);
```

Операторы модификации таблицы EMPLOYEE вида:
INSERT INTO EMPLOYEE (…, DEPT_ID, …) VALUES ROW (…, X, …);
UPDATE EMPLOYEE SET DEPT_ID = X WHERE …;
DELETE FROM EMPLOYEE WHERE …;
где X – значение DEPT_ID, отличное от NULL, а во множество удаляемых строк включается хотя бы одна строка, в которой значение столбца DEPT_ID отличается от NULL, нарушают выделенное ограничение целостности

???

Триггеры в SQL

Триггер – хранимая в БД процедура, автоматически вызываемая СУБД при возникновении определенных условий. В языке обеспечиваются возможности определения триггеров, которые вызываются при модификации указанной базовой таблицы (определение триггеров над представлениями не допускается).
Предметная таблица – базовая таблица, с которой связывается определение триггера.
Инициирующий оператор – оператор SQL, выполнение которого приводит к срабатыванию триггера.
Основные области применения триггеров:

- Согласование и очистка данных
- Журнализация и аудит
- Операции, не связанные с изменением БД (генерация отчетов, печать документов, рассылка почты)

Определение триггера

Оператор CREATE TRIGGER (отмена – DROP TRIGGER). В операторе указываются:

1. Имя триггера
2. Момент срабатывания: BEFORE (непосредственно до выполнения инициирующего оператора) или AFTER (сразу после его выполнения)
3. Тип инициирующего оператора: INSERT, UPDATE [OF column_list] или DELETE
4. Имя предметной таблицы: ON table_name [REFERENCING OLD {ROW | TABLE} AS name NEW {ROW | TABLE} AS name]
5. Количество срабатываний триггера: FOR EACH ROW (вызывается для каждой вставляемой, модифицируемой или удаляемой строки) или FOR EACH STATEMENT (один раз для всего выполнения инициирующего оператора)
6. Дополнительное (опциональное) условие применимости триггера: WHEN (logical_expression), выражение вычисляется для каждой обновляемой строки (триггер FOR EACH ROW) или для всей таблицы (триггер FOR EACH STATEMENT), и триггер срабатывает только в том случае, если результат вычисления выражения равен TRUE
7. Тело триггера: одиночный оператор или процедура на языке SQL/PSM

Триггеры для учебного примера

```
CREATE TRIGGER CHANGE_DEPT_I AFTER INSERT ON EMPLOYEE FOR EACH ROW
WHEN( EMPLOYEE.DEPT_ID IS NOT NULL )
        UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM + 1
WHERE DEPARTMENT.DEPT_ID = EMPLOYEE.DEPT_ID;

CREATE TRIGGER CHANGE_DEPT_D AFTER DELETE ON EMPLOYEE FOR EACH ROW
WHEN( EMPLOYEE.DEPT_ID IS NOT NULL )
        UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM – 1
WHERE DEPARTMENT.DEPT_ID = EMPLOYEE.DEPT_ID;

CREATE TRIGGER CHANGE_DEPT_U AFTER UPDATE OF DEPT_ID ON EMPLOYEE
REFERENCING OLD ROW AS OLD_EMP NEW ROW AS NEW_EMP FOR EACH ROW
        BEGIN ATOMIC
         UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM – 1
WHERE DEPARTMENT.DEPT_ID = OLD_EMP.DEPT_ID;
        UPDATE DEPARTMENT SET DEPT_EMP_NUM = DEPT_EMP_NUM + 1
WHERE DEPARTMENT.DEPT_ID = NEW_EMP.DEPT_ID;
        END;
```

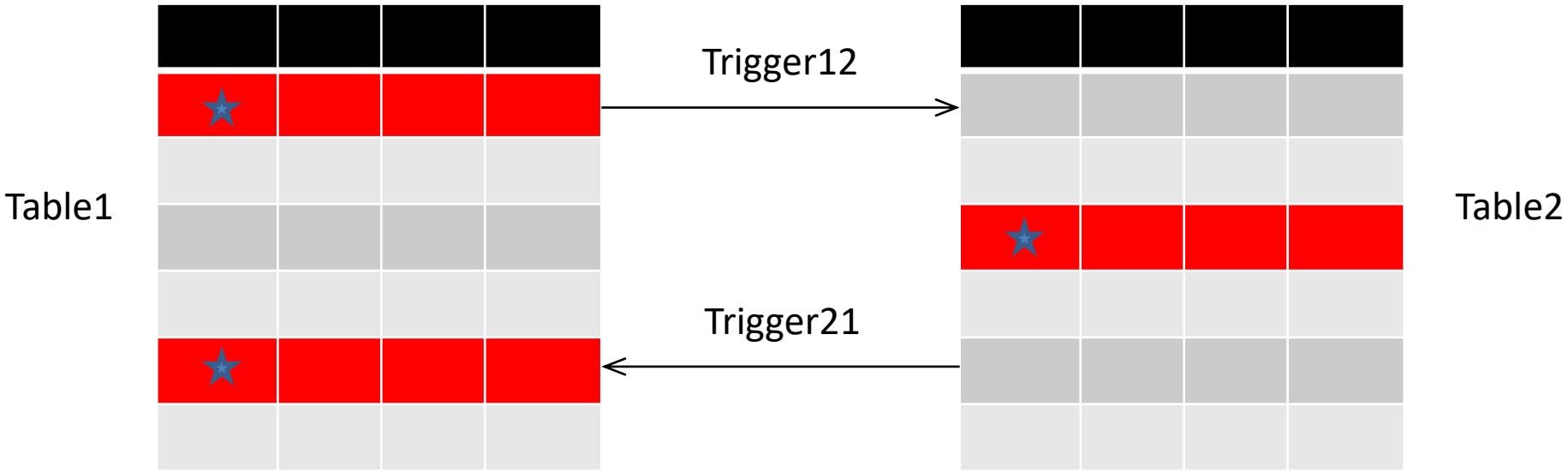
Выполнение триггеров

При выполнении каждого триггера система устанавливает контекст выполнения триггера. Контекст выполнения триггера всегда является атомарным, т.е. инициируемый SQL-оператор либо успешно завершается, либо результаты его действия гарантированно отсутствуют в базе данных. Контекст выполнения триггера включает следующие данные:

- триггерное событие (INSERT, UPDATE или DELETE)
- имя предметной таблицы триггера
- имена столбцов предметной таблицы, специфицированных в определении триггера (для триггеров по UPDATE)
- набор переходов: представление всех вставляемых, модифицируемых или удаляемых строк, а также список уже выполненных триггеров и представлений строк, над которыми эти триггеры выполнялись.

Отслеживание уже выполненных триггеров ведется для предотвращения зацикливания выполнения системы триггеров. Набор переходов изначально пуст, и переходы добавляются при каждом обновлении предметных таблиц выполняемой системы триггеров.

Выполнение триггеров



Набор переходов:

Table1(Row1) -> Trigger 12 -> Table2(Row3)

Table2(Row3) -> Trigger21 -> Table1(~~Row1~~, Row5)

Table1(Row5) -> …

Оператор выборки данных в SQL

```
(5)     SELECT [ ALL | DISTINCT ] select_item_list
(1)             FROM table_reference_list
(2)             [ WHERE logical_expression ]
(3)             [ GROUP BY column_name_list ]
(4)             [ HAVING logical_expression ]
(6)             [ ORDER BY order_item_list ]
```

табличное выражение

Выполнение запроса состоит из нескольких шагов, соответствующих разделам оператора выборки.
СУБД обычно не придерживаются данной схемы выполнения, но результат должен получиться таким, как если бы он получался при точном следовании данной схеме.

Семантика оператора выборки

Шаг 1: FROM table_reference_list
Выполняется операция расширенного декартова произведения таблиц, указанных в списке FROM: $T = T_1$ TIMES T_2 TIMES … TIMES T_n
Аналогом операции переименования RENAME алгебры Кодда в SQL служат псевдонимы таблиц (пример: EMPLOYEE E) и квалифицированные имена столбцов (пример: E.EMP_BDATE)

Шаг 2: [WHERE logical_expression]
Выполняется операция ограничения таблицы T, сформированной на предыдущем шаге, по заданному логическому условию:
T1 = T WHERE logical_expression
T1 содержит только те строки таблицы T, для которых результатом вычисления логического выражения является TRUE (FALSE и UNKNOWN не являются разрешающими).
Отсутствие раздела WHERE в операторе выборки означает WHERE TRUE (T1 ≡ T)

Семантика оператора выборки

Шаг 3: [GROUP BY column_name_list]
На основе таблицы T1, полученной на предыдущем шаге, формируется сгруппированная таблица T2 (строки расставляются в минимальное число групп, таких что во всех строках одной группы значения указанных столбцов одинаковы.
SELECT … FROM EMPLOYEE GROUP BY DEPT_ID;

…	DEPT_ID	…
…	…	…
…	10	…
…	10	…
…	11	…
…	11	…
…	11	…
…	…	…

Шаг 4: [HAVING logical_expression]
Строится таблица T3, содержащая только те группы строк таблицы T2, для которых результатом вычисления логического выражения является TRUE. Логическое выражение HAVING может включать только имена столбцов группировки или агрегатные функции от других столбцов (инварианты группы).
Отсутствие раздела HAVING в операторе выборки означает HAVING TRUE (T3 ≡ T2)
Наличие HAVING при отсутствии GROUP BY означает, что таблица T1 состоит из единственной группы строк, тогда T3 ≡ T1 ∨ T3 ≡ Ø

Семантика оператора выборки

Шаг 5: SELECT [ALL | DISTINCT] select_item_list

Результирующая таблица T4 содержит столько строк, сколько и таблица T1 (в случае отсутствия GROUP BY и HAVING) или же сколько групп строк содержится в таблице T3 (в случае наличия разделов группировки). Число столбцов в таблице T4 зависит от числа элементов в списке выборки:

1. Элемент выборки имеет вид: $Z_i.c_j$ (квалифицированное имя столбца таблицы с псевдонимом Z_i) или $Z_i.*$(все столбцы таблицы с псевдонимом Z_i), для сгруппированных таблиц допустимо указывать только имена столбцов группировки – аналог операции проекции в реляционной алгебре
2. Элемент выборки имеет вид: value_expression [AS column_name], выражение может включать литералы, вызовы функций, имена столбцов таблиц T1 или T3 (в последнем случае – только столбцов группировки, другие столбцы могут являться только аргументами агрегатных функций) – указанное выражение вычисляется для каждой строки таблицы T1 или группы строк таблицы T3, и результат включается в соответствующую строку таблицы T4

DISTINCT – на завершающей стадии из T4 удаляются строки-дубликаты.

Семантика оператора выборки

Шаг 6: [ORDER BY order_item_list]
order_item ::= value_expression [ASC | DESC]
На завершающей стадии выполнения выборки данных производится сортировка строк результирующей таблицы в соответствии со списком элементов сортировки.
В общем случае выражение, входящее в элемент списка сортировки, основывается на именах столбцов результирующей таблицы T4 и именах столбцов таблицы, над которой вычислялся раздел SELECT (T1 или T3). Идея состоит в том, что если некоторое выражение могло бы быть использовано в элементе списка выборки SELECT, то его можно использовать и в элементе списка сортировки.

Конструкция TABLE table_name является сокращенной формой следующего оператора выборки данных: SELECT * FROM table_name

Агрегатные и кванторные функции

Function([DISTINCT | ALL] value_expression)

Исходное мультимножество строк – вся таблица или группа строк в случае сгруппированной таблицы. Для агрегатных функций на основании аргумента (как правило, имени столбца) из заданного мультимножества строк производится мультимножество значений. При наличии DISTINCT из мультимножества значений удаляются дубликаты и NULL. Затем производится вычисление.

COUNT – число строк или значений

MAX – максимальное значение

MIN – минимальное значение

AVG – среднее значение

SUM – суммарное значение

EVERY – квантор всеобщности (TRUE, если вычисление аргумента равно TRUE для каждой строки исходного мультимножества)

SOME или ANY – квантор существования (TRUE, если вычисление аргумента равно TRUE хотя бы для одной строки исходного мультимножества)

Важный частный случай: COUNT(*) – подсчет строк в мультимножестве (при этом все строки считаются различными)

Ссылки на таблицы раздела FROM

FROM table_reference_list

В качестве ссылок на таблицы могут использоваться не только имена базовых таблиц, но и различные выражения запросов. В рамках данного курса будут рассматриваться только наиболее простые случаи.

В самом простом случае в качестве ссылки на таблицу может использоваться:

- имя базовой таблицы (созданной оператором CREATE TABLE)
- имя представления (виртуальной таблицы)
- порождаемая таблица (выражение запроса, заключенное в круглые скобки)
- имя запроса, присоединенного к данному запросу с помощью раздела WITH

Представления (виртуальные таблицы)

Определение представления:
CREATE VIEW view_name [column_name_list] AS query_expression
Имя представления существует в том же пространстве имен, что и имена базовых таблиц. Явное указание имен столбцов представляемой таблицы требуется в том случае, когда эти имена невозможно вывести из соответствующего выражения запроса.
Пример: создать представление с данными о начальниках отделов
CREATE VIEW DEPARTMENT_MANAGER AS SELECT E.EMP_ID, E.EMP_NAME, E.EMP_BDATE, E.EMP_SALARY, D.DEPT_ID, D.DEPT_NAME FROM EMPLOYEE E, DEPARTMENT D WHERE E.EMP_ID = D.DEPT_MANAGER;
Отмена определения представления:
DROP VIEW view_name { RESTRICT | CASCADE }
Из представляемых таблиц можно выполнять выборку данных с помощью оператора SELECT. В общем случае представляемые таблицы не модифицируемы. Стандарт допускает создание представлений с объявлением WITH CHECK OPTION, через которые возможно выполнять операции обновления базы данных (в курсе лекций не рассматриваются).

Порождаемые таблицы и присоединенные запросы

Семантически порождаемые таблицы соответствуют временным представлениям, которые существуют лишь в момент исполнения запроса. Явное указание псевдонима порождаемой таблицы и имен ее столбцов требуется в том случае, когда эти имена невозможно вывести из соответствующего выражения запроса.

Пример: Найти общее число служащих и максимальный размер зарплаты в отделах с одинаковым максимальным размером зарплаты.

```
SELECT SUM( TOTAL_EMP ), MAX_SAL FROM ( SELECT MAX( EMP_SALARY ),
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID IS NOT NULL GROUP BY DEPT_ID ) AS
DEPT_MAX_SAL( MAX_SAL, TOTAL_EMP ) GROUP BY MAX_SAL;
```

Тот же самый запрос можно сформулировать в виде присоединенного запроса с использованием раздела WITH:

```
WITH DEPT_MAX_SAL( MAX_SAL, TOTAL_EMP ) AS ( SELECT MAX( EMP_SALARY ),
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID IS NOT NULL GROUP BY DEPT_ID )
SELECT SUM( TOTAL_EMP ), MAX_SAL FROM DEPT_MAX_SAL GROUP BY MAX_SAL;
```

Задание для размышления

a	b	c	d

Таблица T
Все столбцы определены на типе INTEGER

Укажите, какие из перечисленных ниже операторов выборки данных являются допустимыми в языке SQL:

1. SELECT MIN(a) FROM T WHERE b > 10 HAVING SUM(c) > 100;
2. SELECT b, c, d FROM T GROUP BY b, c;
3. SELECT SUM(a), b FROM T WHERE c = d GROUP BY b;
4. SELECT a, MAX(b), AVG(d) FROM T WHERE c < a GROUP BY d;
5. SELECT COUNT(*) FROM T WHERE a < 0 GROUP BY b HAVING d <> 1;
6. SELECT c FROM T WHERE c >= 5 GROUP BY c HAVING MIN(a) < c;
7. SELECT a, c FROM T WHERE b > -10 GROUP BY b, d;
8. SELECT MAX(a), SUM(b) FROM T GROUP BY d;
9. SELECT a, c+d, AVG(b) FROM T WHERE b > c AND a < d;

Логические выражения в языке SQL

Синтаксически логическое выражение SQL определяется как булевское выражение, которое строится на основе предикатов с использованием логических операций AND, OR и NOT, а также круглых скобок.

В дальнейшем будем использовать следующие обозначения:
s – скалярная величина
R – строковое значение
|R| – степень строки (количество скалярных значений в ней)
T – табличное значение
|T| - количество строк в таблице

Для предикатов выполняются следующие правила:

1. Совместимость типов операндов
2. Равенство степеней строк-операндов ($|R_X| = |R_Y|$ или $|R_X| = |R_T|$, $R_T \in T$)
3. Существование отрицательной формы предиката NOT pred ≡ NOT(pred) – имеются исключения из данного правила

Предикат сравнения, предикат BETWEEN

Предикат сравнения: s_X op s_Y или R_X op R_Y
op::= = | <> | < | > | <= | >=
NULL op s = UNKNOWN, s op NULL = UNKNOWN, NULL op NULL = UNKNOWN

Предикат BETWEEN (проверка вхождения в диапазон значений):
s_X BETWEEN s_Y AND s_Z или R_X BETWEEN R_Y AND R_Z, $|R_X| = |R_Y| = |R_Z|$
Эквивалентные формы (выражение через предикаты сравнения):
$s_X >= s_Y$ AND $s_X <= s_Z$ или $R_X >= R_Y$ AND $R_X <= R_Z$

Пример: найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты начальника отдела

```
SELECT E1.EMP_ID, E1.EMP_NAME, E1.EMP_SALARY FROM EMPLOYEE E1 WHERE
E1.EMP_SALARY BETWEEN ( SELECT AVG( E2.EMP_SALARY ) FROM EMPLOYEE E2
WHERE  E2.DEPT_ID = E1.DEPT_ID ) AND ( SELECT MNG.EMP_SALARY FROM
DEPARTMENT_MANAGER MNG WHERE MNG.DEPT_ID = E1.DEPT_ID );
```

Предикат сравнения с квантором

Квантифицированное сравнение строчного значения с табличным запросом:

$$R_X \text{ op ALL } T = \begin{cases} \text{FALSE} \Leftrightarrow \exists R_T \in T : (R_X \text{ op } R_T) = \text{FALSE} \\ \text{TRUE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) = \text{TRUE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) \neq \text{FALSE} \wedge \exists R_T \in T : (R_X \text{ op } R_T) = \text{UNKNOWN} \end{cases}$$

$$R_X \text{ op SOME } T = \begin{cases} \text{TRUE} \Leftrightarrow \exists R_T \in T : (R_X \text{ op } R_T) = \text{TRUE} \\ \text{FASLE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) = \text{FALSE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) \neq \text{TRUE} \wedge \exists R_T \in T : (R_X \text{ op } R_T) = \text{UNKNOWN} \end{cases}$$

R_X op ANY T ≡ R_X op SOME T

Пример: найти номера сотрудников отдела 12, зарплата которых в этом отделе не является минимальной

```
SELECT EMP_ID FROM EMPLOYEE WHERE DEPT_ID = 12 AND EMP_SALARY >
SOME( SELECT E1.EMP_SALARY FROM EMPLOYEE E1 WHERE EMPLOYEE.DEPT_ID =
E1.DEPT_ID );
```

Выполнение корреляционного запроса

Пример: найти номера сотрудников отдела 12, зарплата которых в этом отделе не является минимальной

```
SELECT EMP_ID FROM EMPLOYEE WHERE DEPT_ID = 12 AND EMP_SALARY >
SOME( SELECT E1.EMP_SALARY FROM EMPLOYEE E1 WHERE EMPLOYEE.DEPT_ID =
E1.DEPT_ID );
```

EMPLOYEE

EMP_ID	…	EMP_SALARY	DEPT_ID	…
100		10000.00	10	
101		15000.00	12	
102		12000.00	10	
103		12000.00	11	
104		14000.00	12	
105		20000.00	12	
106		22000.00	11	
107		11000.00	10	

E1

E1

EMP_SALARY
15000.00
14000.00
20000.00

RESULT

EMP_ID
101
105

Предикат IS NULL

Проверяет, являются ли неопределенными значения всех элементов строки-операнда: R_X IS NULL

	R_X IS NULL	R_X IS NOT NULL	NOT R_X IS NULL	NOT R_X IS NOT NULL
$\|R_X\|=1$, $s=NULL$	TRUE	FALSE	FALSE	TRUE
$\|R_X\|=1$, $s\neq NULL$	FALSE	TRUE	TRUE	FALSE
$\|R_X\|>1$, $\forall s\in R_X \Rightarrow s=NULL$	TRUE	FALSE	FALSE	TRUE
$\|R_X\|>1$, $\forall i\neq j \ \exists s_i\in R_X : s_i=NULL \wedge \exists s_j\in R_X : s_j\neq NULL$	FALSE	FALSE	TRUE	TRUE
$\|R_X\|>1$, $\forall s\in R_X \Rightarrow s\neq NULL$	FALSE	TRUE	TRUE	FALSE

Пример: найти номера и имена служащих, не задействованных в проектах
SELECT EMP_ID, EMP_NAME FROM EMPLOYEE WHERE PRO_ID IS NULL;

Предикат IN

Проверяет факт вхождения скалярного значения или строки в указанное множество: s IN (s_1, s_2, …, s_n) или R_X IN T

$$R_X \text{ IN } T = \begin{cases} \text{TRUE} \Leftrightarrow \exists R_T \in T : (R_X = R_T) = \text{TRUE} \\ \text{FASLE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X = R_T) = \text{FALSE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X = R_T) \neq \text{TRUE} \wedge \exists R_T \in T : (R_X = R_T) = \text{UNKNOWN} \end{cases}$$

Пример: найти номера сотрудников, не являющихся начальниками отделов и получающих зарплату, размер которой равен размеру зарплаты какого-либо начальника отдела

```
SELECT EMP_ID FROM EMPLOYEE WHERE EMP_ID NOT IN ( SELECT
DEPT_MANAGER FROM DEPARTMENT ) AND EMP_SALARY IN ( SELECT
EMP_SALARY FROM DEPARTMENT_MANAGER );
```

Предикаты LIKE и SIMILAR

Сопоставление символьных и битовых строк с заданным шаблоном:
source_string LIKE pattern_string [ESCAPE symbol]
В шаблоне могут использоваться два специальных символа:
‘_’ (подчеркивание) – интерпретируется как произвольный одиночный символ
‘%’ (процент) – интерпретируется как произвольная подстрока произвольной длины
Для битовых строк используются коды соответствующих символов (X’5F’ и X’25’).
Раздел ESCAPE специфицирует одиночный символ (например, ‘\’), используемый для изменения способа интерпретации ‘_’ и ‘%’ (пары символов ‘_’ и ‘\%’ будут интерпретироваться как одиночные символы ‘_’ и ‘%’ соответственно).
Основное отличие SIMILAR от LIKE состоит в возможности использования регулярных выражений внутри шаблона (в курсе лекций не рассматривается).
Пример: подсчитать количество проектов, в названии которых присутствует слово ‘software’
SELECT COUNT(*) FROM PROJECT WHERE PRO_TITLE LIKE ‘%software%’ OR PRO_TITLE LIKE ‘%Software%’;

Предикат OVERLAPS

Служит для проверки перекрытия по времени двух событий: R_X OVERLAPS R_Y, $|R_X|=|R_Y|=2$, строки задаются в виде: (t_S, t_F) или (t_S, interval), где $t_F = t_S$ + interval

Эквивалентное представление в виде предикатов сравнения:

($t_{SX} = t_{SY}$) OR

($t_{SX} > t_{SY}$) AND ($t_{SX} <= t_{FY}$ OR $t_{FX} <= t_{FY}$) OR

($t_{SY} > t_{SX}$) AND ($t_{SY} <= t_{FX}$ OR $t_{FY} <= t_{FX}$)

Пример: найти названия проектов, которые будут выполняться в течение следующего года

SELECT PRO_TITLE FROM PROJECT WHERE (PRO_SDATE, PRO_DURATION) OVERLAPS (CURRENT_DATE, INTERVAL ‘1’ YEAR)

Предикат EXISTS

Проверяет наличие строк в результате запроса

$$\text{EXISTS}(T) = \begin{cases} \text{TRUE} \Leftrightarrow |T| > 0 \\ \\ \text{FALSE} \Leftrightarrow |T| = 0 \end{cases}$$

Запросы с предикатом EXISTS можно переформулировать в виде запросов с предикатом сравнения и агрегатной функцией COUNT(*).

Пример: найти номера отделов, среди служащих которых есть руководители проектов

```
SELECT D.DEPT_ID FROM DEPARTMENT D WHERE EXISTS( SELECT
E.EMP_ID FROM EMPLOYEE E WHERE E.DEPT_ID = D.DEPT_ID AND
EXISTS( SELECT P.PRO_MANAGER FROM PROJECT P WHERE
P.PRO_MANAGER = E.EMP_ID ) );
```

Предикат IS DISTINCT FROM

Позволяет проверить, являются ли две строки дубликатами.

R_X IS DISTINCT FROM R_Y =

$$\begin{cases} \text{FALSE} \Leftrightarrow \forall i = \overline{1, |R_X|} \Rightarrow (s_{Xi} = s_{Yi}) = \text{TRUE} \\ \quad \vee\ s_{Xi} = \text{NULL} \wedge s_{Yi} = \text{NULL},\ s_{Xi} \in R_X,\ s_{Yi} \in R_Y \\ \\ \text{TRUE} \Leftrightarrow \exists i : (s_{Xi} = s_{Yi}) = \text{FALSE} \vee s_{Xi} = \text{NULL} \\ \quad \wedge\ s_{Yi} \neq \text{NULL} \vee s_{Xi} \neq \text{NULL} \wedge s_{Yi} = \text{NULL} \end{cases}$$

Отрицательная форма предиката (IS NOT DISTINCT FROM) в языке SQL отсутствует.

Пример: найти пары номеров проектов, выполняющихся в организации в одно и то же время

```
SELECT P1.PRO_ID, P2.PRO_ID FROM PROJECT P1, PROJECT P2
WHERE P1.PRO_ID <> P2.PRO_ID AND NOT( ( P1.PRO_SDATE,
P1.PRO_DURATION ) IS DISTINCT FROM ( P2.PRO_SDATE,
P2.PRO_DURATION ) );
```

Предикат UNIQUE

Служит для проверки факта отсутствия строк-дубликатов в результате запроса.
UNIQUE(T) = TRUE тогда и только тогда, когда в таблице T отсутствуют строки-дубликаты, в противном случае возвращается FALSE.

Пример: найти названия отделов, служащих которых можно различить по размеру получаемой зарплаты
SELECT DEPT_NAME FROM DEPARTMENT WHERE UNIQUE(SELECT EMP_SALARY FROM EMPLOYEE WHERE EMPLOYEE.DEPT_ID = DEPARTMENT.DEPT_ID);

Предикат MATCH

Условие соответствия строчного значения результату подзапроса:

R_X MATCH [UNIQUE] SIMPLE T = TRUE $\Leftrightarrow \exists s \in R_X : s=NULL \vee \exists R_T \in T : (R_X = R_T)=TRUE$

R_X MATCH [UNIQUE] PARTIAL T = TRUE $\Leftrightarrow \forall s \in R_X \Rightarrow s=NULL \vee \exists R_T \in T : \forall s_{Xi} \neq NULL \Rightarrow (s_{Xi} = s_{Ti})=TRUE, s_{Xi} \in R_X, s_{Ti} \in R_T$

R_X MATCH [UNIQUE] FULL T = TRUE $\Leftrightarrow \forall s \in R_X \Rightarrow s=NULL \vee \forall s \in R_X \Rightarrow s \neq NULL \wedge \exists R_T \in T : (R_X = R_T)=TRUE$

Если указано UNIQUE, то дополнительно проверяется, является ли найденная строка R_T уникальной в T (FALSE, если неуникальная).

Пример: найти номера служащих, зачисленных в один из отделов, для которых в их отделах работают служащие с той же самой датой рождения

```
SELECT EMP_ID FROM EMPLOYEE WHERE ( DEPT_ID, EMP_BDATE ) MATCH FULL (
SELECT E1.DEPT_ID, E1.EMP_BDATE FROM EMPLOYEE E1 WHERE E1.EMP_ID <>
EMPLOYEE.EMP_ID );
```

Транзакции в SQL

Транзакция – последовательность операций над базой данных, которая воспринимается системой как единая операция.

Принципы АСИД (ACID):

- Атомарность (Atomicity) – транзакция выполняется как единая операция и либо результаты всех операций, ее составляющих, отражаются в базе данных, либо они там гарантировано отсутствуют
- Согласованность (Consistency) – транзакция переводит базу данных из одного согласованного состояния в другое согласованное состояние, она успешно завершается в том и только том случае, когда действия ее операций не нарушают целостность БД
- Изоляция (Isolation) – две транзакции, выполняющиеся одновременно (параллельно или квазипараллельно), не должны никоим образом действовать друг на друга (результаты одной транзакции не должны быть видны никакой другой, до тех пор, пока первая транзакция не завершится успешно)
- Долговечность (Durability) – после успешного завершения транзакции все внесенные ею изменения должны быть гарантированно сохранены в БД даже в случае аппаратных или программных сбоев

Инициация транзакций

В SQL транзакции могут образовываться:

- явно с использованием оператора START TRANSACTION
- неявно, когда выполняется оператор, для которого требуется контекст транзакции, а этого контекста не существует

Большинство операторов SQL (за исключением ряда административных операторов) требуют наличие контекста транзакции.

Явная инициация транзакции: START TRANSACTION mode_list

Характеристики транзакции:

1. Режим доступа: READ ONLY или READ WRITE
2. Уровень изоляции: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ или SERIALIZABLE
3. Размер области диагностики (количество сохраняемых диагностических сообщений): DIAGNOSTIC SIZE value (значение по умолчанию определяется в реализации)

Для характеристик транзакций, инициируемых неявно, используются либо значения по умолчанию, либо значения, определяемые оператором SET TRANSACTION mode_list

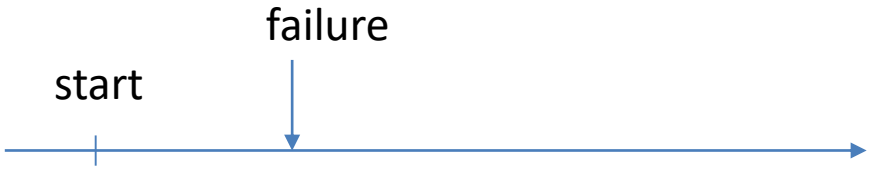
(данный оператор недопустимо выполнять в контексте активной транзакции).

Завершение транзакций

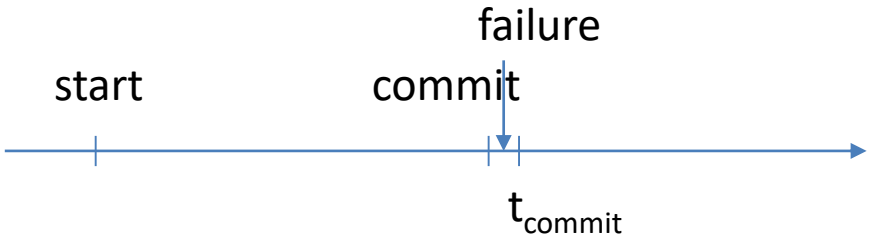
Для завершения стартовавшей транзакции должен быть явно использован один из двух операторов:
COMMIT [WORK] [AND [NO] CHAIN] – фиксация транзакции (завершение с фиксацией результатов в БД)
ROLLBACK [WORK] [AND [NO] CHAIN] – откат транзакции (завершение с возвратом к предыдущему состоянию БД)
Если присутствует раздел AND CHAIN, то по завершении текущей транзакции образуется новая, наследующая все характеристики завершенной.
Оператор COMMIT считается безусловно выполненным только тогда, когда это подтверждает СУБД после выполнения всех действий, необходимых для фиксации результата транзакции. Это делается с целью защиты от сбоев, произошедших в момент выполнения COMMIT.

Поддержка долговечности

Сбой после фиксации транзакции: Транзакция долговечна (состояние БД восстанавливается по журналу или по журналу и резервной копии)



Сбой во время выполнения транзакции: Возврат к состоянию до начала выполнения транзакции



Сбой в процессе фиксации транзакции: транзакция считается незафиксированной и все ее изменения автоматически удаляются из базы данных при ее восстановлении

Учебный пример

```
CREATE TABLE EMPLOYEE (
…
DEPT_ID INTEGER DEFAULT
NULL REFERENCES
DEPARTMENT ON DELETE SET
NULL,
…
);
```

```
CREATE TABLE DEPARTMENT (
        DEPT_ID INTEGER PRIMARY KEY,
        DEPT_EMP_NUM INTEGER NOT NULL
CHECK( VALUE <= 100 ),
        CHECK ( DEPT_EMP_NUM = ( SELECT
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID =
EMPLOYEE.DEPT_ID )),
…
);
```

Операторы модификации таблицы EMPLOYEE вида:
INSERT INTO EMPLOYEE (…, DEPT_ID, …) VALUES ROW (…, X, …);
UPDATE EMPLOYEE SET DEPT_ID = X WHERE …;
DELETE FROM EMPLOYEE WHERE …;
где X – значение DEPT_ID, отличное от NULL, а во множество удаляемых строк включается хотя бы одна строка, в которой значение столбца DEPT_ID отличается от NULL, нарушают выделенное ограничение целостности

???

Транзакции и ограничения целостности

В контексте выполняемой транзакции каждое ограничение целостности может находиться в одном из двух режимов:
1. Режим немедленной проверки (immediate) – ограничение проверяется сразу после выполнения любой операции, изменяющей состояние БД. Операция отвергается, если она нарушает хотя бы одно ограничение целостности, находящееся в данном режиме.
2. Режим отложенной проверки (deferred) – ограничение проверяется при выполнении операции COMMIT. Транзакция откатывается (COMMIT трактуется как ROLLBACK), если ее операции нарушили хотя бы одно отложенно проверяемое ограничение целостности.

Для указания режима проверки к определению ограничения целостности добавляется следующая синтаксическая конструкция (в качестве заключительной конструкции определения ограничения):
INITIALLY { DEFERRED | IMMEDIATE } [[NOT] DEFERRABLE]
По умолчанию предполагается INITIALLY IMMEDIATE NOT DEFERRABLE
Комбинация INITIALLY DEFERRED NOT DEFERRABLE недопустима.

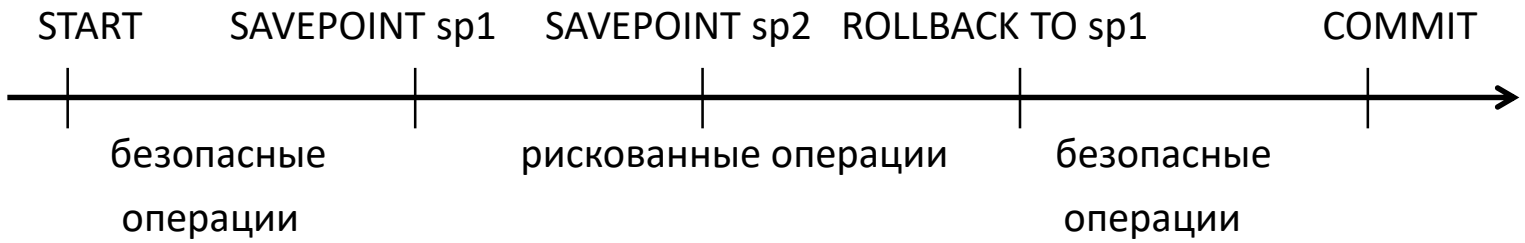
Изменение режима проверки ограничений

Режим проверки ограничений, которые специфицированы с указанием ключевого слова DEFERRABLE, можно изменить в ходе выполнения транзакции с помощью оператора:
SET CONSTRAINTS { ALL | constraint_name_list } { DEFERRED | IMMEDIATE }
Если в списке имен будет явно указано хотя бы одно ограничение, определенное как NOT DEFERRABLE, оператор будет отвергнут.

Ограничения, находящиеся в режиме отложенной проверки, будут проверены сразу при переводе их в режим немедленной проверки (неявно – при выполнении COMMIT или явно – при выполнении SET CONSTRAINTS … IMMEDIATE).

Точки сохранения в транзакциях

Точка сохранения – пометка в последовательности операций транзакции, которую можно использовать для ее частичного отката с сохранением жизнеспособности и результатов операций, выполненных до точки сохранения.
Установление точки сохранения: SAVEPOINT savepoint_name
Удаление точки сохранения: RELEASE SAVEPOINT savepoint_name
Откат до точки сохранения: ROLLBACK TO SAVEPOINT savepoint_name
В одной транзакции возможно установить несколько последовательных точек сохранения. При откате до некоторой точки сохранения неявно выполняется RELEASE для всех точек, определенных в транзакции после данной.

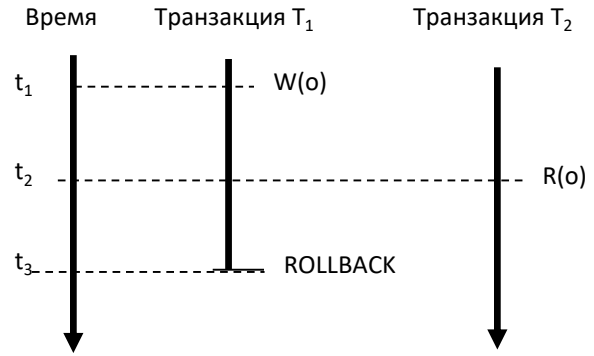


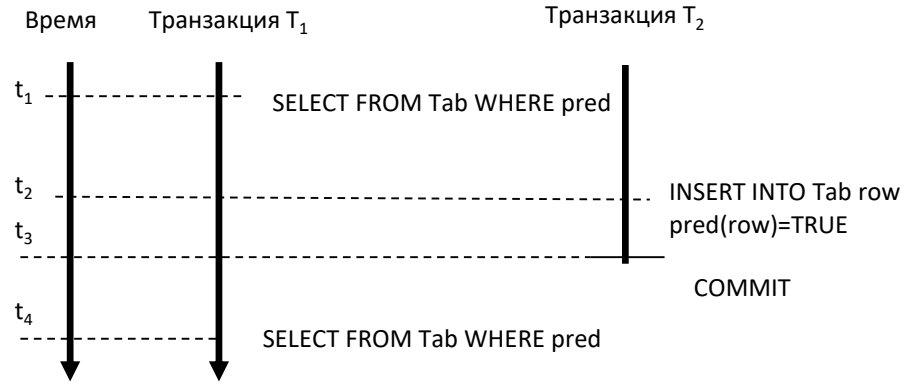
Точки сохранения не нарушают принцип атомарности, поскольку извне транзакции она все равно рассматривается как единая операция (точки сохранения не видны).

Феномены выполнения транзакций

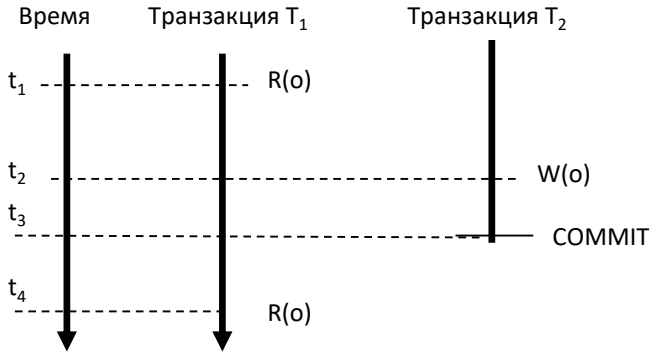
При одновременном (параллельном или квазипараллельном) выполнении нескольких транзакций могут возникнуть следующие феномены.

Грязное чтение (некорректные данные в транзакции T_2)





Неповторяющееся чтение (несовпадение результатов операций чтения одного и того же объекта в транзакции T_1)



Фантомы (появление фантомных строк в таблице Tab при повторном запросе в транзакции T_1)

Уровни изоляции транзакций в SQL

На уровне изоляции READ UNCOMMITTED могут проявляться все три феномена. Рекомендуется его использовать только в тех транзакциях, где точные данные не обязательны (статистическая обработка). Одновременное задание READ UNCOMMITTED и режима доступа READ WRITE не допускается. Каждый следующий уровень устраняет проявление одного из феноменов.

Уровень изоляции	Описание	Грязное чтение	Неповт. чтение	Фантомы
READ UNCOMMITTED	Возможность видеть изменения, произ-водимые еще не зафиксированными параллельными транзакциями	+	+	+
READ COMMITTED	Допускаются изменения объектов, прочитанных другими одновременно выполняющимися транзакциями	-	+	+
REPEATABLE READ	Допускается добавление строк к таблицам, которые удовлетворяют условиям выборки, выполненной в других параллельных транзакциях	-	-	+
SERIALIZABLE	Предельная изолированность одно-временно выполняющихся транзакций	-	-	-

Поддержка авторизации доступа к данным в SQL

Метод авторизации доступа, используемый в SQL относится к мандатным видам защиты данных: с каждым зарегистрированным в СУБД пользователем или ролью (субъектом) и каждым защищаемым объектом БД связывается мандат (или привилегия), определяющий действия, которые может выполнять данный субъект над данным объектом.

В SQL поддерживается принцип сокрытия информации об объектах, содержащихся в схеме БД, от субъектов, которые лишены доступа к этим объектам. Если некоторый субъект не обладает привилегиями доступа к некоторой таблице, то при попытке выполнить какое-либо действие над ней он получит такое же диагностическое сообщение, как если бы данная таблица не существовала.

Создатель объекта базы данных автоматически становится владельцем этого объекта, который обладает полным набором привилегий для выполнения действий над объектом, в том числе привилегией на передачу всех или части своих привилегий другим субъектам.

Привилегии доступа к объектам

Действие	Привилегия	Объекты
Просмотр	SELECT	Таблицы, столбцы, хранимые процедуры
Вставка	INSERT	Таблицы, столбцы
Модификация	UPDATE	Таблицы, столбцы
Удаление	DELETE	Таблицы
Ссылка	REFERENCES	Таблицы, столбцы
Использование	USAGE	Домены, типы и проч. определения
Инициирование	TRIGGER	Таблицы
Выполнение	EXECUTE	Хранимые процедуры
Типизация	UNDER	Определяемые пользователем типы
Передача	GRANT/ADMIN	Привилегии/роли

Привилегии над представлениями основываются на привилегиях по отношению к базовым таблицам данных представлений.

Пользователи и роли

Привилегии доступа к объектам предоставляются пользователям, а также ролям, выполнение которых, в свою очередь, может предоставляться пользователям. С каждым пользователем и каждой ролью связывается уникальный идентификатор авторизации (authID).

Роль – динамически образуемая группа пользователей СУБД, каждый из которых обладает привилегией на исполнение данной роли, а также всеми привилегиями данной роли для доступа к объектам БД. Роли упрощают построение и администрирование системы авторизации доступа.

В стандарте SQL не определяются средства создания и ликвидации идентификаторов пользователей. Для создания и ликвидации ролей поддерживаются операторы CREATE ROLE/DROP ROLE. В стандарте также поддерживается концепция идентификатора псевдопользователя PUBLIC, который соответствует любому пользователю, зарегистрированному в СУБД. Пользователю PUBLIC могут предоставляться привилегии доступа к объектам базы данных, как и любому другому пользователю, при этом они будут распространяться автоматически и на всех вновь создаваемых пользователей.

Создание и ликвидация роли

Создание роли:
CREATE ROLE role_name [WITH ADMIN { CURRENT_USER | CURRENT_ROLE }]
Имя роли должно отличаться от любого идентификатора авторизации (authID), уже определенного в СУБД. При наличии раздела WITH ADMIN привилегию на исполнение данной роли вместе с правом передачи привилегии получает либо текущий пользователь, либо текущая роль SQL сессии. По умолчанию привилегия на исполнение создаваемой роли передается текущему пользователю. Если SQL сессия не имеет пользователя – то текущей роли. Привилегии, требуемые для выполнения CREATE ROLE определяются в реализациях (как правило, выполнение разрешается только администраторам).

Ликвидация роли:
DROP ROLE role_name
Для выполнения DROP ROLE текущий authID SQL сессии (пользователь или роль) должен являться владельцем данной роли. При ликвидации роли автоматически ликвидируются привилегии на ее исполнение у всех пользователей и ролей, которым эта привилегия ранее передавалась.

Передача привилегий

GRANT { ALL PRIVILEGES | privilege_list } ON object_name TO { PUBLIC | authID_list } [WITH GRANT OPTION] [GRANTED BY { CURRENT_USER | CURRENT_ROLE }]

Привилегии передаются от текущего authID сессии (пользователя или роли) к указанным в списке authID (пользователям или ролям). Для этого он должен обладать привилегией на передачу всех или части привилегий из списка, указанного в операторе GRANT.
Если текущий authID обладает правом на передачу только части привилегий из этого списка, то передается именно эта часть и выдается предупреждение, а если он не имеет прав на передачу ни одной из перечисленных привилегий, то фиксируется ошибка.
Если указан раздел WITH GRANT OPTION, то привилегии из списка передаются с правом дальнейшей передачи.
Раздел GRANTED BY позволяет явно указать, от какого authID (текущего пользователя или текущей роли) передаются привилегии.
Избыточная дублирующая передача привилегий от имени одного и того же authID другому (тому же самому) authID игнорируется.

Передача ролей

GRANT role_name_list TO { PUBLIC | authID_list } [WITH ADMIN OPTION] [GRANTED BY { CURRENT_USER | CURRENT_ROLE }]

Оператор позволяет передавать произвольное число ролей произвольному числу пользователей или ролей.
Если текущий authID обладает правом на передачу только части ролей из этого списка, то передается именно эта часть и выдается предупреждение, а если он не имеет прав на передачу ни одной из перечисленных ролей, то фиксируется ошибка.
Если указан раздел WITH ADMIN OPTION, то привилегии на исполнение ролей из списка передаются с правом дальнейшей передачи.
Раздел GRANTED BY позволяет явно указать, от какого authID (текущего пользователя или текущей роли) передаются привилегии на исполнение ролей.

Аннулирование привилегий

REVOKE [GRANT OPTION FOR] privilege_list ON object_name FROM { PUBLIC | authID_list } [GRANTED BY { CURRENT_USER | CURRENT_ROLE }] { RESTRICT | CASCADE }

RESTRICT – действие оператора отвергается, если хотя бы одна из указанных в списке привилегий была передана другому тем authID, у которого привилегия должна быть аннулирована (сочетание с GRANT OPTION FOR означает, что аннулируется привилегия на передачу привилегий из указанного списка, сама привилегия при этом остается).
CASCADE – указанные привилегии аннулируются у всех authID, прямо или косвенно (через промежуточные authID) получивших привилегии от текущего authID, выполняющего данную операцию.
Для успешного выполнения REVOKE необходимо, чтобы текущий authID обладал всеми или частью привилегий из указанного списка. В последнем случае аннулируется именно эта часть (с выдачей предупреждения).

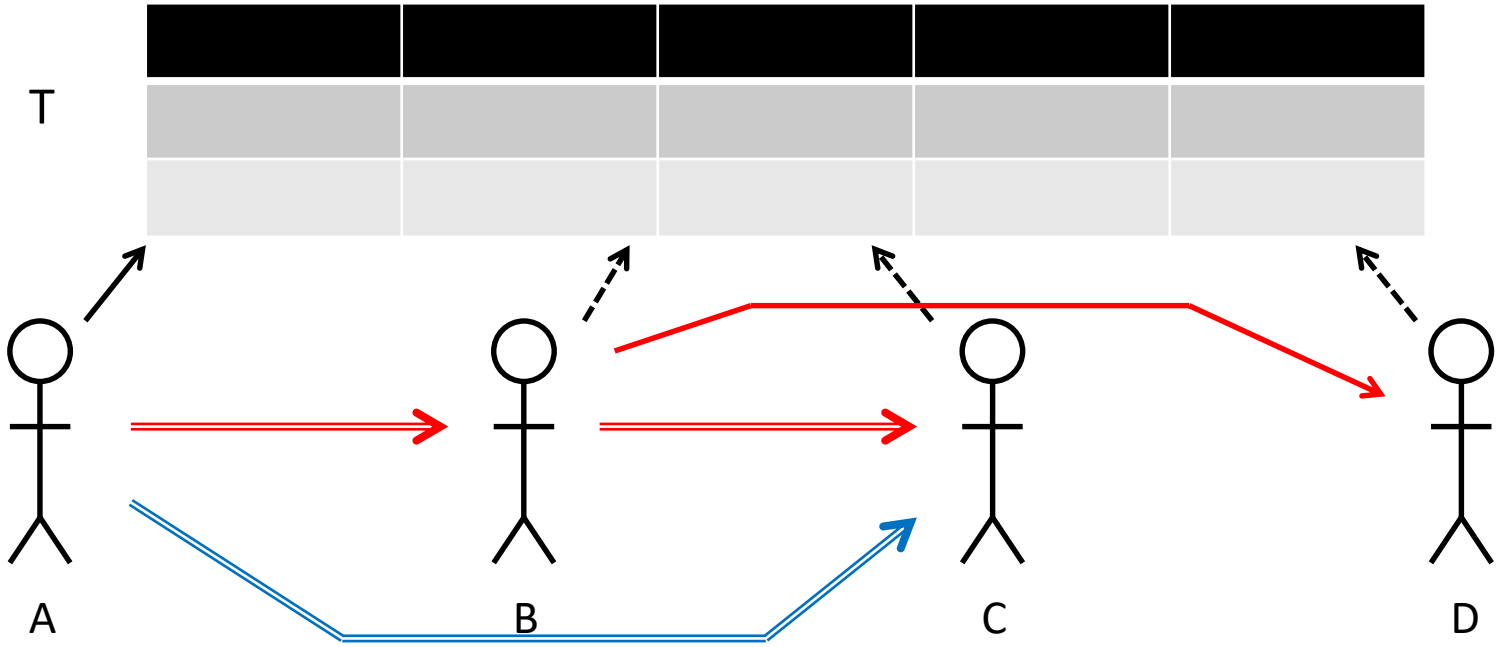
Аннулирование привилегий

При аннулировании привилегий отслеживается путь их передачи по графу идентификаторов авторизации и объектов БД. Поэтому возможны ситуации, когда у некоторого authID останется привилегия даже после ее аннулирования (при условии, что он получил ее несколькими путями).

A – текущий пользователь SQL сессии и владелец таблицы T.

REVOKE priv ON T FROM B CASCADE;

REVOKE priv ON T FROM C RESTRICT;



Аннулирование ролей

REVOKE [ADMIN OPTION FOR] role_name_list FROM { PUBLIC | authID_list } [GRANTED BY { CURRENT_USER | CURRENT_ROLE }] { RESTRICT | CASCADE }

Действие операции аннулирования ролей очень похоже на действие операции аннулирования привилегий. Отличия состоят в том, что аннулируются не привилегии, а роли, а также в том, что для аннулирования привилегии на передачу роли используется раздел ADMIN OPTION FOR.

Если некоторая привилегия или роль была передана PUBLIC, то ей обладают все пользователи. Но нет возможности аннулировать такую привилегию или роль у отдельно указываемого пользователя. Привилегия или роль была передана всем, и аннулировать ее можно только сразу у всех, то есть у псевдопользователя PUBLIC.

Модели данных

Модель данных описывает некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, основанные на этой модели.

По К. Дейту для любой модели данных принято выделять три части:

1. Структурная часть – описывает основные логические структуры данных, которые могут применяться на уровне пользователя при организации БД
2. Манипуляционная часть – обеспечивает модельный язык БД
3. Целостная часть – специфицирует механизмы ограничений целостности, которые обязательно должны поддерживаться во всех реализациях СУБД

Изученные модели данных:

- Дореляционные (60-е – 70-е г.г.):
  - Иерархическая – деревья
  - Сетевая – произвольные ориентированные графы
  - Инвентированные списки – таблицы с физической адресацией записей
- Реляционная (Э. Кодд, 1969 г.)
- Модель данных SQL

Проблема «потери соответствия»

К началу 90-х годов SQL-ориентированные СУБД заняли на мировом рынке доминирующее положение, которое удерживают и по настоящий момент.

В 80-е годы стала приобретать популярность парадигма объектно-ориентированного программирования (языки Smalltalk, Ada, Objective-C, C++). В 90-е годы объектно-ориентированное программирование вошло в моду и стало завоевывать преимущественное положение в программной индустрии. Б*о*льшая часть современных производственных приложений, связанных с потребностью в обработке больших объемов данных, разрабатывается на различных объектно-ориентированных языках (C++, Java, C#).

В связи с этим возникла проблема «потери соответствия» (impedance mismatch) между системами типов и средствами доступа к данным объектно-ориентированных языков программирования и SQL-ориентированных СУБД. Это затрудняет разработку приложений, которые по своей специфике вынуждены часто обращаться к базе данных для доступа к требуемым им данным.

Предложенные подходы

1989 г., М. Аткинсон и др., «Манифест систем объектно-ориентированных баз данных»:

- СУБД должна иметь возможность хранить данные произвольно сложной структуры и обеспечивать развитую систему типов данных
- Реляционная модель данных и язык SQL являются ограниченными и не подходят для этих целей
- Предлагается использовать в СУБД объектно-ориентированную модель и формулируются базовые требования к ООСУБД

1990 г., М. Стоунбрейкер и др., «Манифест систем баз данных третьего поколения»:

- СУБД должна иметь возможность хранить данные произвольно сложной структуры и обеспечивать развитую систему типов данных
- Можно добиться требуемых результатов, эволюционно развивая SQL

1995 г., К. Дейт, Х. Дарвен, «Третий манифест»:

- Идеи первых двух манифестов необоснованы и плохо проработаны
- Классическая реляционная модель данных необходима и достаточна для использования в современных СУБД
- Фактически предлагается новая, «истинная реляционная» модель

Объектно-ориентированная модель

Литеральные типы данных – могут использоваться в качестве типов атрибутов внутри объектов. Значения литеральных типов не могут самостоятельно храниться в базе данных.

Атомарные литеральные типы: числовые, символьные, булевские, темпоральные (принятые как в традиционных языках программирования, так и в языках баз данных).

Конструируемые литеральные типы:

- Запись (аналог структуры в языках C/C++)
- Коллекции:
  - Множество (неупорядоченная, без дубликатов)
  - Мультимножество (неупорядоченная, с дубликатами)
  - Массив (упорядоченная, с возможностью доступа к элементу по индексу)
  - Список (упорядоченная, с возможностью быстрой вставки и удаления элементов)
  - Словарь (множество пар <ключ, значение>)

Объектные типы данных

Объектные типы данных: атомарные (классы) и объектные коллекции.
Каждый объектный тип имеет операцию создания и инициализации (конструктор).
Значения объектных типов имеют объектный идентификатор (OID) и могут самостоятельно храниться в базе данных.
OID автоматически генерируется СУБД, является уникальным во всей БД, не зависит от состояния объекта (значений его атрибутов).
При определении класса указывается его внутренняя структура (набор атрибутов и связей) и набор операций. Одиночное наследование поддерживается в обязательном порядке, поддержка множественного наследования является желательной. Поддерживаются только бинарные связи.
В отличие от литеральных коллекций, объектные коллекции могут самостоятельно храниться в БД, набор операций для каждого из типов коллекций предопределен.
Экстентом объектного типа называется именованная коллекция типа множества, элементами которой являются объекты данного типа.

Виды эквивалентности объектов

В связи с наличием у объектов уникальных идентификаторов различают три вида эквивалентности объектов:

1. Эквивалентность по идентификации (идентичность): объекты являются идентичными тогда и только тогда, когда их идентификаторы совпадают.
2. Поверхностная эквивалентность: два объекта поверхностно эквивалентны тогда и только тогда, когда их атрибуты идентичны (выполняется поверхностное сравнение: простые атрибуты сравниваются по значению, а объектные – по идентификатору).
3. Эквивалентность по значению (равенство): два объекта равны тогда и только тогда, когда значения всех их атрибутов одинаковы (выполняется глубокое сравнение: при наличии атрибутов объектного типа должны быть одинаковыми значения всех их атрибутов рекурсивно).

Очевидно, что два идентичных объекта будут также равны и поверхностно эквивалентны, в то время как обратное неверно. Аналогично, поверхностная эквивалентность объектов означает их равенство, но не наоборот.

Язык OQL

OQL (Object Query Language) – базовое средство манипулирования в объектно-ориентированной модели.

• OQL не является вычислительно полным языком. Он представляет собой простой язык запросов (обеспечивается декларативный доступ к объектам).

• OQL очень близок по синтаксису к SQL/92. Расширения относятся к объектно-ориентированным понятиям, таким как объекты, объектные идентификаторы, путевые выражения (обход по связям), полиморфизм, вызов операций, и проч.

• В OQL поддерживаются средства доступа ко всем возможным структурам данных, допускаемых в структурной части модели.

• OQL является функциональным языком, допускающим неограниченную композицию операций, если операнды не выходят на пределы системы типов (результат любого запроса обладает типом, принадлежащим к объектной модели, и поэтому к результату запроса может быть применен новый запрос).

• Операторы языка OQL могут вызываться из любого языка программирования, для которого определены правила связывания. И, наоборот, в запросах OQL могут присутствовать вызовы операций, запрограммированных на этих языках.

• В OQL не определяются явные операции обновления, а используются вызовы операций, определенных в объектах для целей обновления.

Язык OQL

Результатом допустимых в OQL выражений запросов могут являться:

- индивидуальный объект;
- коллекция объектов;
- коллекция литеральных значений;
- индивидуальное литеральное значение.

Пример: определить руководителей отделов и тех служащих их отделов, зарплата которых превышает 20000 руб.

```
SELECT DISTINCT STRUCT ( DEPT_MNG: D.DEPT_MANAGER,
                                    EMP: ( SELECT E FROM D.CONSISTS_OF AS E WHERE
E.EMP_SALARY > 20000.00 )
            ) FROM DEPARTMENTS D;
```

DEPARTMENTS – имя экстента объектного типа DEPARTMENT

CONSISTS_OF – имя роли (конца связи) типа EMPLOYEE в связи с типом DEPARTMENT

Результат запроса имеет тип set <struct { OID <EMPLOYEE> DEPT_MNG; bag < EMPLOYEE> EMP }>

Прочие языковые средства и поддержка целостности данных

Язык ODL (Object Definition Language) – позволяет описывать схему БД в виде набора интерфейсов объектных типов (описание атрибутов, связей между объектными типами, а также операций и их параметров). ODL не является языком программирования, требуются средства отображения конструкций ODL в объектно-ориентированные языки программирования для реализации типов. В качестве языка манипулирования объектами (создание, модификация, удаление) предполагается использование объектно-ориентированных языков программирования. С этой целью языки программирования расширяются дополнительными конструкциями либо библиотечными элементами. Правила связывания в курсе лекций не рассматриваются.

В объектной модели отсутствует аналог проверочного ограничения целостности реляционной модели данных.
Ссылочная целостность поддерживается для связей «один-ко-многим». В этом случае объекты на стороне связи «один» рассматриваются как предки, а объекты на стороне связи «многие» – как потомки, и ООСУБД обязана следить за тем, чтобы не образовывались потомки без предков.

Подходы к созданию и удалению объектов

Различают перманентные (persistent) и транзитные (transient) объекты. Первые реально сохраняются в БД, т.е. приобретают свойство хранимости (устойчивости). Вторые существуют временно в течение сеанса работы приложения.
Существуют следующие подходы к созданию (конструированию) объектов:

1. Создание экземпляра (путем вызова его конструктора) приводит к его добавлению в базу данных (автоматически предполагается устойчивость экземпляров).
2. Вызов конструктора предполагает создание транзитного экземпляра. Тогда созданный во время работы программы объект уничтожается при ее завершении, если он не сделан устойчивым путем вызова специального метода.
3. Возможно конструировать как перманентные, так и транзитные экземпляры. Тип создаваемого экземпляра определяется специальным параметром конструктора.

Удаление объектов может осуществляться двумя альтернативными способами:

1. Объект физически удаляется в любом случае. Контроль ссылочной целостности на уровне СУБД при этом не производится и полностью возлагается на приложение.
2. Объект имеет счетчик ссылок на него, при ненулевом значении этого счетчика вместо уничтожения объекта выставляется "флаг уничтожения" (tombstone), и объект удаляется только после того, как счетчик обнулится.

Объектно-ориентированные СУБД

Известные реализации: db4o, Objectivity/DB, Versant

- Ни одна из СУБД не поддерживает ограничения целостности, за исключением ссылочной, да и то в ограниченном объеме, поддержка целостности данных фактически является ответственностью приложения
- Неустановленные значения поддерживаются только для объектных ссылок, для других типов – это забота приложения
- Механизмы триггеров, хранимых процедур и т.п., не поддерживаются
- Ограничение доступа к данным со стороны различных пользователей не поддерживается или ограниченно поддерживается в Versant
- Ни ODL, ни OQL не поддерживаются (каждая ООСУБД предлагает собственные языковые средства), запросы, результатами которых являются литеральные значения или их множества не поддерживаются
- db4o поддерживает только Java или C#, причем для этих языков выпускаются два разных продукта, остальные хотя и декларируют независимость от языков, эта независимость является условной (необходимо соблюсти ряд требований, например, по использованию определенных типов данных)
- Перевод существующего приложения достаточно трудоемок (например, необходимо перейти на типы данных, поддерживаемые той или иной ООСУБД)

Современное состояние

С конца 90-х годов в области ООСУБД наблюдался явный упадок. Основные причины:

- Недостатки ООСУБД (предыдущий слайд), главным из которых является зависимость от языков программирования
- Отсутствие крупных игроков на рынке (таких как Oracle и IBM)
- Появление объектно-реляционных возможностей в большинстве популярных SQL-ориентированных СУБД
- Маркетинговая политика крупных компаний, которые сумели внушить пользователям должное доверие к универсальности, надежности и масштабируемости своих SQL-ориентированных решений

С середины 2000-х некоторое повышение интереса к ООСУБД:
2005 г. – возникновение концорциума ODBMS.ORG
2006 г. – сформирована группа (в рамках OMG) по подготовке новой версии стандарта объектно-ориентированной модели данных
2008 г. – первая после длительного перерыва конференция, посвященная проблемам ООСУБД

Объектно-реляционные расширения SQL

Типы данных, введенные в SQL:1999 и не относящиеся к ОО расширениям:

- Массивы: datatype ARRAY [max_cardinality]
- Мультимножества: datatype MULTISET
- Анонимные строчные типы: ROW(field1, …, fieldN), где field ::= name, type, options

Типы данных, определяемые пользователем:

- Индивидуальные типы: CREATE TYPE UDT_name AS base_type_name FINAL;
- Структурные типы: CREATE TYPE UDT_name [UNDER UDT_name] AS (attribute_list) [INSTANTIABLE | NOT INSTANTIABLE] { FINAL | NOT FINAL } [reference_specification] [method_specification_list]

Индивидуальный тип – именованный тип данных, основанный на единственном предопределенном типе. Индивидуальный тип не наследует от своего опорного типа набор операций над значениями.

Структурный тип данных – именованный типы данных, включающий один или более атрибутов любого из допустимых в SQL типа данных. Дополнительные механизмы: наследование, определяемые пользователями методы.

Типизированные таблицы

CREATE TABLE typed_table_name OF UDT_name [UNDER typed_table_name] [(typed_table_element_list)]
typed_table_element ::= constraint_definition | self-ref_column_definition | column_options
CREATE VIEW typed_view_name OF UDT_name UNDER base_table_name AS query_expression
При определении типизированной таблицы указывается ранее определенный структурный тип, и если в нем содержится n атрибутов, то в таблице образуется n+1 столбец, дополнительный (первый) столбец называется самоссылающимся и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой при вставке строк в типизированную таблицу, явно указываться пользователями или состоять из комбинации значений других столбцов. Можно определить подтаблицу типизированной таблицы, если структурный тип подтаблицы является непосредственным подтипом структурного типа супертаблицы.
В случае типизированных представлений выражение запроса должно основываться на единственной базисной таблице или представлении (базисная таблица и определяемое представление должны быть ассоциированы с одним и тем же структурным типом).

Механизмы генерации ссылочных значений

Спецификация ссылки в UDT	Определение самоссылающегося столбца
REF IS SYSTEM GENERATED	REF IS column_name SYSTEM GENERATED
REF USING predefined_type	REF IS column_name USER GENERATED
REF USING (attribute_list)	REF IS DERIVED

Определение типа атрибута – ссылки на другой структурный тип:
REF(UDT_name) [SCOPE typed_table_name REFERENCES ARE [NOT] CHECKED [ON DELETE action]]

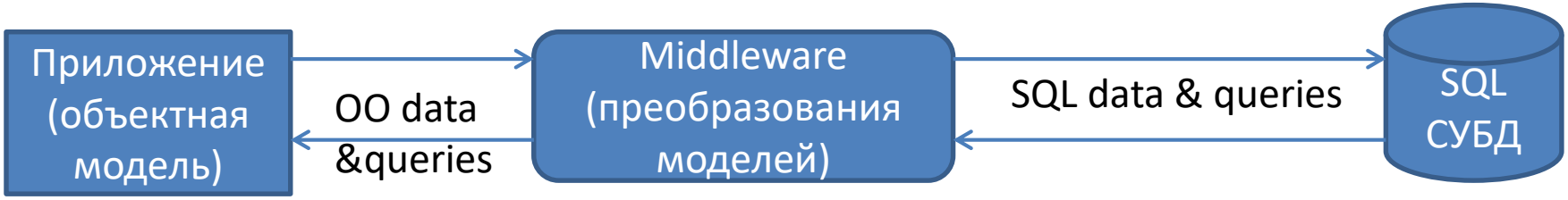
С типизированной таблицей можно обращаться, как с традиционной таблицей, считая, что у нее имеются неявно определенные столбцы (атрибуты структурного типа), а можно относиться к строкам типизированной таблицы, как к объектам соответствующего структурного типа, OID которых содержатся в «самоссылающемся» столбце.

Пример использования типизированных таблиц

```
CREATE TYPE EMP_T AS (
        EMP_NAME VARCHAR( 200 ),
        EMP_BDATE DATE,
        EMP_SALARY SALARY,
        DEPT_ID REF( DEPT_T ),
        PRO_ID REF( PRO_T )
) INSTANTIABLE NOT FINAL
REF IS SYSTEM GENERATED;

CREATE TYPE PROG_T UNDER EMP_T AS (
        PROG_LANG_SKILL VARCHAR( 100 )
) INSTANTIABLE NOT FINAL;

CREATE TABLE EMPLOYEE OF EMP_T( REF IS EMP_ID SYSTEM GENERATED );

CREATE TABLE PROGRAMMER OF PROG_T UNDER EMPLOYEE;
```

Основные подходы к объектно-реляционному отображению

Реализации: Oracle, IBM DB2, Informix, PostgreSQL (частично).
Очень многие разработчики приложений используют промежуточное программное обеспечение (middleware) объектно-реляционного отображения, которое само взаимодействует с базами данных на основе БАЗОВОГО подмножества языка SQL без использования каких-либо объектных расширений.



Основные подходы к построению middleware:

1. Использование OO API (JDBC, ADO.NET), которые обеспечивают доступ к реляционным данным и их извлечение в форме, более привлекательной для разработчиков объектно-ориентированных приложений
2. Объекты подстраиваются под реляционную модель (известные методики преобразования, нормализация, BLOB-стратегия)
3. В БД сохраняются не только состояния объектов, но и метаданные, описывающие их структуру (middleware реализует концепцию объектного кэша)

Истинная реляционная модель данных

Ключевая идея третьего манифеста – чтобы достичь требуемой объектной функциональности, не надо абсолютно ничего делать с реляционной моделью; на основе идей Э. Кодда можно реализовать СУБД, обеспечивающие возможности по части представления и хранения данных произвольно сложной структуры, не меньшие тех, которые обеспечивают объектные и SQL-ориентированные СУБД.
Основное препятствие – тезис Кодда о нормализации отношений (1НФ): в реляционной базе данных должны содержаться только отношения, атрибуты которых определены на доменах, элементы которых являются атомарными (не составными) значениями.

К. Дейт: «Я согласен с Коддом, что желательно оставаться в рамках логики первого порядка, если это возможно. В то же время я отвергаю идею "атомарных значений", по крайней мере, в смысле абсолютной атомарности. В Третьем манифесте мы допускаем наличие доменов, содержащих значения произвольной сложности. Они могут быть даже отношениями. Тем не менее, мы остаемся в рамках логики первого порядка.»

Типы данных истинной реляционной модели

Три категории типов данных: скалярные типы, кортежные типы и типы отношений.

Скалярный тип – инкапсулированный тип, реальная внутренняя структура которого скрыта от пользователей. Предлагаются механизмы определения новых скалярных типов и операций над ними. Типом атрибута определяемого скалярного типа может являться любой определенный к этому моменту скалярный тип, кортежный тип или тип отношения.

Некоторые базовые скалярные типы данных должны быть предопределены в системе. В число этих типов должен входить тип truth value (булевский тип) с двумя значениями true и false.

Кортежный тип – тип данных, определяемый с помощью генератора типа TUPLE с указанием множества пар <имя_атрибута, тип_атрибута> (заголовка кортежа). Типом атрибута кортежного типа может являться любой определенный к этому моменту скалярный тип, любой кортежный тип и тип отношения. Значением кортежного типа является кортеж, представляющий собой множество триплетов <имя_атрибута, тип_атрибута, значение_атрибута>, которое соответствует заголовку кортежа этого типа.

Типы данных истинной реляционной модели

Тип отношения – тип данных, определяемый с помощью генератора типа RELATION с указанием некоторого заголовка кортежа. Значением типа отношения является заголовок отношения, совпадающий с заголовком кортежа этого типа отношения, и тело отношения, представляющее собой множество кортежей, соответствующих этому заголовку.
Кортежные типы и типы отношений не являются инкапсулированными: имеется возможность прямого доступа к атрибутам. Для всех разновидностей типов данных поддерживается модель множественного наследования, позволяющая определять новые типы данных на основе уже определенных типов.
База данных – набор долговременно хранимых именованных переменных отношений, каждая из которых определена на некотором типе отношений.

Третий манифест: «Каждый кортеж в отношении *R* содержит в точности одно значение *v* для каждого атрибута *A* в заголовке отношения *H*. Иными словами, *R* находится в первой нормальной форме, 1NF».

Манипулирование данными и поддержка целостности данных

Эталонные средства манипулирования данными: реляционная алгебра Кодда, реляционная алгебра A.
Языковые средства – язык запросов D:
• для выражения запросов используется алгебраический подход
• запросы, адресуемые к сложным данным, формулируются более точно, чем на SQL
• это же касается сложных операций обновления
• язык обладает вычислительной полнотой
• язык претендует на то, чтобы стать открытым стандартом и заменить SQL.

Любое условное выражение, которое является замкнутой правильно построенной формулой (WFF) реляционного исчисления, должно быть допустимо в качестве спецификации ограничения целостности.
Обязательное требование – определение хотя бы одного возможного ключа для каждой переменной отношения.
Средства поддержки декларативной ссылочной целостности (ограничения внешнего ключа) фигурируют только в разделе рекомендуемых возможностей.

Реализации истинной реляционной модели

Единственное коммерческое решение:
Dataphor (разработка: 1999-2001, с 2001 по 2008: коммерческий продукт компании Alphora, после ее приобретения Database Consulting Group продукт выпускается под открытой лицензией, текущий релиз в 2018 году, исходный код на C#)

Открытые проекты (университеты и индивидуальные разработчики):

- Alf (Ruby)
- Dee (Python)
- DuroDBMS (multiple languages)
- Rel (Java)
- TclRAL (TCL)