**Московский авиационный институт (технический университет)**
**факультет Прикладной математики и физики**

**В.Н. Лукин**

# Базы данных

***(конспект лекций)***

Москва, 2009

# Оглавление

# Предисловия для студентов

### *К версии 3.31*

Примеры в этом конспекте почти все правильные, по крайней мере, те ошибки, которые я заметил, исправлены. Но некоторые из них (из примеров) достаточно убогие, особенно в лекции 14, мне хотелось их заменить. Так что если свои приведете – вам только плюс.

По примеру БД из лекции 3 вопросов на экзамене нет, но это из моей программистской жизни, и выбрасывать ее жалко. Хеширование в таком объеме, как в конспекте, на лекциях не давалось (это было на втором курсе), но на дополнительные вопросы отвечать, возможно, придется, тем более что это тема из моих любимых. Что касается лекций 14-15, то, несмотря на их незаконченность (урезанный курс не дает возможность добраться до логического финала – синтеза базы данных), студенты в них спокойно ориентируются.

Некоторые лекции выходят за границы пары, другие короче. Я не стремился к «методической чистоте». Просто так сложилось исторически (раньше курс был на полтора семестра, и пришлось выбрасывать целые темы).

### *К версии 3.4*

Эта версия – продолжение версии 3.31. На этот раз я на самом деле взялся за модификацию конспекта в начале семестра. Изменения следующих видов: исправление ошибок, обновление примеров и иллюстраций, добавления и уточнения в текст, переработка неудачных лекций, расширение лекции про диаграммы «сущность-связь», в частности, стандарт *IDEF1X* и пакет *ERwin*. Структура конспекта отличается от структуры лекционного курса: лекция по моделям читается раньше, чтобы можно было скорее приступить к лабораторкам.

### *К версиям 3.41-3.44*

***Версия 3.41*** – продолжение версии 3.4 и прощание с 2007 годом. На 29 декабря приведено в порядок почти всё. По сравнению с предыдущей версией почти полностью переработана Лекция 1. Лекция 2 тоже значительно изменена. Лекции 3 и 4 несколько улучшены. Осталось заменить многочисленные примеры одним сквозным, докончить хвост, а также добавить материал по *ERwin*. Кроме того, мне не нравится пример на нормальную форму Бойса-Кодда, при случае заменю. Но это не самые актуальные работы: можно и без сквозного примера обойтись, тем более что придуманная мною тема (про студентов) оказалась не слишком оригинальной и скучноватой. Наконец, *ERwin* изучают на лабораторках и в лекции включать его не так уж обязательно, хотя одну лекцию я на него убил, да и вопрос о нём есть в билетах.

***Версия 3.42*** – это первая работа в 2008 году. В Лекции 8 более аккуратно даны операции расщепления и фактора, добавлены примеры. Везде улучшены (на мой взгляд) иллюстрации, за счёт чего заметно уменьшился размер файла.

***Версии 3.43-3.44*** – продолжение работы осенью 2008 года. Исправлены некоторые ошибки. Лекция 11 (про нормальные формы) изложена получше. Но, возможно, мне только так кажется.

### *К версии 3.5-3.51*

Всё время хочется поставить точку, но она остаётся многоточием. И эта версия не исключение. Выверить мне её, кажется, удалось (если кто обнаружит ошибки – прошу о них сообщить, хотя бы в знак признательности за мой труд, доставшийся вам на халя-

ву). Но если бы этим кончилось! С удлинением лекций на 15 минут появилась возможность расширить материал, но как? Новую тему не вставишь, текст можно увеличить. А в некоторых случаях это невозможно. Так и перетекают темы из одной лекции в другую, не останавливаясь на границе. Из-за этого расширилась тема по проектированию, реально она больше лекции. Добавились слова про транзакции, их в лекции не было и в вопросах пока нет. Сократился «хвост» курса: я уже не надеюсь добраться до алгоритма синтеза, как мечталось, хорошо, что удалось поговорить о сокращении функциональных зависимостей. Соответственно, я убрал слова про структуру неизбыточных покрытий (там, где говорилось о разбиении множества функциональных зависимостей): теперь они не работают на какой-то логический финал. Так что Лекция 16 оказалась весьма редуцированной, и я её удалил. Но я добрался, наконец, до пятой нормальной формы и изложил её по возможности ясно. Список литературы расширился.

Надеюсь, что на сей раз до сессии удастся всё как-то частично закончить, кроме, возможно, примеров. Хотя действительно, конспект – не публикация, и делать его не очень хочется. Студентов только жалко. С удовольствием отберу на экзамене качественный экземпляр этой версии.

08.12.2009

# Лекция 1. Хранение данных

### *Информация. Данные*

Понятие «базы данных» возникло в 60-х годах, наиболее бурное развитие этого направления пришлось на 70-е годы. Тогда же сложился, в основном, и теоретический фундамент этого направления. Состояние вычислительной техники в то время заметно отставало от теоретических разработок, и это не позволило в полной мере оценить их значимость. Последующее увлечение мини- и микро-ЭВМ оттеснило на второй план тематику, связанную с централизованным хранением и коллективным использованием больших массивов данных. Однако история повторяется: микро-ЭВМ, объединенные в сети, быстро достигли, а затем и превысили возможности старых «больших» машин, накопленные данные стали востребованы многочисленными пользователями, и это вновь возродило интерес большим базам данных.

Осознание в 60-х годах проблемы обработки больших объемов данных определилось всей логикой развития информатики. На первых этапах основные усилия были направлены на создание сносной вычислительной техники, после чего, по сути, и появилось настоящее программирование. Разработка теоретических основ программирования позволила оснастить программистов качественным инструментарием, что в свою очередь привело к появлению большого количества программ, особенно прикладных, дающих возможность не только обрабатывать текущую информацию, но и накапливать ее. С момента осознания проблем, связанных с хранением и обработкой больших объемов информации, начинается информатика, ориентированная на пользователя. Информацию необходимо интерпретировать с точки зрения некоторой предметной области – это вызвало появление экспертных систем, базирующихся на понятии искусственного интеллекта.

Таблица иллюстрирует изменение «основного вопроса информатики», и ключевой фигуры в процессе развития вычислительной техники от примитивных счетных машин к современным.

| | *Этап 1* | *Этап 2* | *Этап 3* | *Этап 4* |
|---|---|---|---|---|
| *обеспечение* | аппаратное | программное | информационное | интеллектуальное |
| *субъект* | электроник | программист | пользователь | эксперт |

В последнее время с развитием вычислительных сетей, в том числе, глобальных, стали актуальными проблемы хранения распределенных данных и доступа к ним, а также проблемы доступа к слабоструктурированным данным. Таким образом, вопросы организации данных снова заняли достойное место.

В начале изучения курса принято обсуждать термины, составляющие его базу. Мы тоже не будем отступать от традиции и неформально обсудим те понятия, которые так или иначе относятся к нашему курсу. Гораздо более подробно об этом можно прочитать в [16], где рассматриваются различные варианты толкований. Мы приведём только краткие и, возможно, не всегда бесспорные определения и объяснения.

Информация. В теории информации это первичное понятие, не определяемое в рамках этой теории. Мы будем использовать этот термин в общеупотребительной интерпретации и понимать как *смысловое содержание сообщения, сокращающее неопределённость знания о возможности возникновения некоторого события из множества возможных*.

Данные. Нередко этот термин используется как синоним информации. Будем считать, что данные – *это представление фактов и идей в формализованном виде, пригодном для передачи и переработки в некотором процессе*.

Обработка данных – выполнение систематических последовательных действий с данными.

Информация, относящейся к некоторой задаче, образует ее информационную среду, которая представляется как совокупность носителей данных, включенных в обработку при решении этой задачи. Данные обычно бывают взаимозависимыми – связанными, причем, число связей по мере детализации постановки задачи увеличивается. Взаимосвязанные данные называют системами данных.

Естественно, данные должны где-то храниться. Эти хранилища обычно называют системами хранения информации. Цель существования систем хранения информации – обеспечить выдачу достоверной информации в определенное время, определенному лицу, в определенном месте, за определенную плату.

Предметная область (ПО) – совокупность информационной среды и технологии обработки информации, ориентированная на конечного пользователя. Это любая область знания (производство), о которой могли бы быть собраны данные, пригодные для хранения и обработки. Предметная область в задачах обработки данных выражается моделью данных. Модель может быть представлена совокупностью взаимосвязанных объектов, каждый из которых характеризуется данными предметной области. К сожалению, термин «объект» в настоящее время перегружен смыслами, поэтому чаще в качестве синонима будем употреблять термин «сущность».

Элемент данных объекта будем называть его атрибутом. Атрибут может принимать значение, которое подчиняется некоторым ограничениям. Множество допустимых значений атрибута составляет его домен. Совокупность значений атрибутов объекта будем называть экземпляром объекта.

Атрибуты, относящиеся к объектам предметной области, с точки зрения обработки данных группируются в записи, представляющие собой совокупности связанных атрибутов. В свою очередь, однородные записи объединяются в файл данных. Если записи в файле имеют одну и ту же структуру, файл называется плоским.

Наличие информации предполагает её поступление из некоторого источника на каком-то материальном носителе. Среда, из которой поступает информация, и называется источником данных. Носители информации, используемые в источнике данных, называют первичными документами.

Программная система, предназначенная для хранения, обработки, поиска, распространения, передачи и предоставления информации, называется информационной системой. Рассматривается три основных направления использования информационных систем:

- средства для обработки больших массивов неструктурированной информации;
- средства автоматизации технологических процессов предприятия;
- средства автоматизации труда управленцев.

### *Системы хранения данных на основе файлов*

В ранних прикладных информационных системах обработка данных производилась путём непосредственного доступа к файлам данных. Программы обработки занимались ведением конкретных файлов, то есть, в основном, добавлением, удалением, корректировкой данных, сортировкой и выдачей. Файлы содержали все данные, необходимые для обработки. С увеличением мощности информационных систем количество данных, вовлекаемых в обработку, росло, увеличивалось и количество различных решаемых задач. Логическая структура данных, как правило, определялась разработчиком для конкретной задачи, в крайнем случае, для группы задач. Более того, нередко данные различных приложений отличались и физическими структурами, поэтому использование их для других задач было крайне затруднительно. Приходилось одни и те же данные дублировать в разных задачах. Получающаяся избыточность с высокой вероятно-

стью приводила к противоречивости данных. Исключить избыточность можно, по крайней мере, двумя путями: работать с длинными записями, которые содержали бы все связанные атрибуты, или формировать структуры взаимосвязанных файлов с физическими ссылками. Оба варианта страдают существенными недостатками: в первом случае возникают трудности с поддержкой длинных записей и с ограничением доступа; во втором существенно увеличивается сложность обработки за счет вовлечения в нее одновременно нескольких файлов.

Шагом вперед стало использование обобщенных методов доступа к данным, которые определяют их структуру на нижнем уровне. В этом случае за счёт стандартизации доступа упрощается хотя бы физическая организация данных. Системы, поддерживающие эти методы, обычно называются Системами Управления Файлами – СУФ (*File Manager – FMGR*), они включаются в операционные системы (ОС). Универсальные программы работают с единственным представлением данных или же с фиксированным числом представлений. Примером может служить СУФ в ОС *RTE* фирмы *Hewlett-Packard*, где используется 6 форматов файлов. Однако этот подход всё равно ничего не упрощает на логическом уровне.

Итак, главный недостаток системы, построенной на файлах, связан с тем, что короткие записи, ориентированные на решение частных задач, приводят к избыточности, возникающей из-за повторения одних и тех же данных в разных файлах. Это порождает проблему противоречивости данных, которая усугубляется слабым контролем достоверности данных.

Попытка борьбы с противоречивостью путем объединения записей приводит к следующим неприятностям:

- ведение длинных записей представляет собой трудоемкую задачу;
- система данных на длинных записях отличается крайне низкой гибкостью;
- при недостаточности средств защиты возможен несанкционированный доступ к данным;
- процесс восстановления данных сложен и требует значительного времени;
- стоимость и сложность в эксплуатации таких систем крайне высока.

Представление информационной базы в виде системы взаимосвязанных файлов, уменьшая избыточность, приводит к таким следствиям, как

- сложность в управлении: в прикладных программах на реализацию поддержки одновременной работы с несколькими файлами требуются значительные усилия;
- ограничение разделения данных: в отсутствие общих механизмов реализации связей между файлами этой деятельностью приходится заниматься на уровне прикладных программ, что снижает возможности гибкого разделения данных;
- ограничение по доступности: необходимость быстрой выборки требуемой информации противоречит сложности поиска и интеграции данных.

### *База данных*

Термин «база данных» трактуется далеко не однозначно. Существует, по крайней мере, два его аспекта: бытовой и технический. Не берусь отстаивать эти названия, они очень условны. На бытовом уровне под базой данных понимают информационную систему, в основном, справочную. Говорят: «Я купил базу данных адресов Москвы» или что-то в этом роде. Технический вариант относится к особой структуре организации данных. Она характеризуется своей теорией, методологией, технологией и практикой. Мы, разумеется, будем рассматривать именно этот вариант.

Казалось бы, в рамках строгого технического подхода за много лет существования баз данных и при участии большого количества учёных разных уровней можно сформировать чёткое определение базы данных. Однако на самом деле ни одно из предложенных определений не может считаться исчерпывающе точным. Возникает

подозрение в том, что сама суть такого объекта, как база данных, не вполне определена. Чтобы получить представление о спектре различных определений, достаточно просмотреть несколько работ различных авторов, посвящённых этой теме. Наиболее полно подходы к этому определению изложены в [16]. Мы приведём два из них: первое, восходящее к [2], и второе, данное в стандарте *ISO*, которое приводится в [16].

***Определение 1.*** *Под базой данных (БД) будем понимать совокупность связанных данных конкретной предметной области, в которой определения данных и отношений между ними отделены от процедур.*

***Определение 2.*** *База данных есть совокупность данных, организованных в соответствии с некоторой концептуальной моделью данных, которая описывает характеристики этих данных и взаимоотношения между соответствующими им реалиями и которая предназначена для информационного обеспечения одного или более приложений.*

Здесь появляется термин «концептуальная модель», который относится к наиболее общему формализованному описанию предметной области, включающему описание её объектов и связей между ними.

Несмотря на, казалось бы, более полное Определение 2, оно не содержит упоминания о раздельности данных и процедур их обработки, поэтому к базам данных можно отнести и информационную систему на основе плоских файлов. Первое определение, в свою очередь, грешит нечёткостью. Впрочем, в силу его простоты и понятности, мы будем пользоваться именно им.

**Основное отличие баз данных от систем на основе файлов** состоит в том, что эти системы имеют несколько назначений и несколько представлений о данных, а базы данных – несколько назначений и одно представление о данных, что обеспечивается специальными средствами доступа к данным.

Базы данных призваны ликвидировать неприятности, присущие системам на основе файлов, и они это успешно делают, но по сравнению с ними они тоже имеют некоторые недостатки. Объективно – это довольно высокая стоимость и необходимость специальной подготовки, что в простейших случаях хранения данных представляется излишним. Субъективно – пользователь нередко хочет видеть данные в своих файлах без посредников в виде средств доступа. Кроме того, при переходе к использованию БД наблюдается снижение ответственности исполнителя, что влияет на достоверность данных. В свою очередь, достоверность трудно контролировать из-за отсутствия избыточности. Возникают проблемы и с защитой данных, для этого требуются специальные мероприятия.

### *Система управления базами данных*

Определяя базы данных как среду хранения данных, отделённую от процедур, мы подразумеваем, наличие специальных программных средств доступа к данным – систем управления базами данных (СУБД). Система управления базами данных тоже определяется по-разному. Иногда под этим термином понимают базу данных в первом, бытовом, смысле. Но это ничем не оправдано, просто, видимо, человек хочет поумнее выразиться. Но и правильно понятый термин, как программная система для доступа к базам данных, тоже определяется неоднозначно. Приведём определения из [2] и [10].

***Определение 1.*** *Система управления данными – комплекс программно-аппаратных средств, обеспечивающих доступ к БД и управление данными.*

***Определение 2.*** *Система управления данными – программная система, предназначенная для создания и хранения базы данных на основе некоторой модели данных, обеспе-*

*чения логической и физической целостности содержащихся в ней данных, надёжного и эффективного использования ресурсов, предоставления к ней санкционированного доступа для приложений и конечных пользователей, а также для поддержки функций администратора баз данных.*

Опять, как и при определении баз данных, второе определение более полное, но и более громоздкое. Да и не бесспорное: могут быть СУБД, не обладающие некоторыми из заявленных свойств. Но первое определение слишком общее, всё-таки СУБД должна обладать специфическими свойствами. И они приводятся в [2] как требования к СУБД:

- эффективное выполнение функций предметной области;
- минимизация избыточности;
- предоставление непротиворечивой информации;
- безопасность;
- простота в эксплуатации;
- простота физической реорганизации;
- возможность централизованного управления;
- упрощение приложений.

Если теперь сравнить два приведённых определения, можно увидеть и громоздкость, и неполноту в каждом из них. Ещё одно определение СУБД будет дано в следующей лекции.

### *Администратор БД (АБД)*

Предположим, что в организации разрабатывается большой проект на основе общей БД. Возникают вопросы: по каким правилам работать? Кто определяет структуру данных? Кто регламентирует доступ к данным? Наконец, кто выбирает подходящую СУБД? Как только появляется проект, затрагивающий интересы нескольких пользователей БД, возникает необходимость в долгосрочной функции администрирования.

Администратор БД – это сотрудник, отвечающий за обеспечения необходимого уровня производительности прикладной программной системы. В его обязанности входит выполнение следующих функций:

- координация проектирования, реализации и ведения БД;
- определение структуры данных и правил доступа к данным;
- выбор подходящей СУБД;
- определение и использование эффективных методов доступа;
- выбор оптимальной избыточности данных;
- сбор и обработка статистики функционирования системы;
- определение необходимости и проведение реорганизации структуры или среды хранения базы данных;
- восстановление состояния базы данных при нарушениях целостности;
- оценка перспективы и формирование требований, исходя из особенностей предметной области.

Помимо этих обязанностей, АБД нередко играет роль консультанта по всем вопросам, касающимся баз данных и СУБД, которые исходят и от программистов, и от пользователей, и от администрации. Значит, АБД должен быть не только профессионалом в области баз данных, но должен знать предметную область и обладать психологическими характеристиками, позволяющими успешно общаться со всеми категориями участников работы с программной системой. Наиболее полно об АБД можно прочитать в [2].

# Лекция 2. Модели данных

### *Модель, схема*

Понятие модели используется очень широко. Обычно под моделью понимают упрощённое представление реальности, сохраняющее какие-то важные свойства и абстрагирующееся от несущественных. Модели, предназначенные для разных целей, могут быть представлены в самых разнообразных формах: натурные, математические, графические и т.п. В технологии баз данных существуют специальные модели, позволяющие представить структуру данных и способы их использования. Это, например, модель предметной области, модель данных, модель архитектуры, модель транзакций. В свою очередь, эти модели детализируются, формируется иерархия моделей. Каждая модель имеет своё представление. Графическое представление модели называют *диаграммой*. Иногда этот термин используют как синоним модели.

Далее мы будем рассматривать модели данных. Как и в ряде других случаев, в литературе существует более одного определения этого термина. Интересующихся отсылаем к [10, 15], мы будем пользоваться следующим определением.

***Определение.*** *Модель данных – совокупность методов и средств определения логического представления физических данных, относящихся к предметной области.*

Модель данных – ни что иное, как формализация данных прикладной области для возможности их обработки. Она характеризуется тремя компонентами [5]:

1. Правила структурирования данных для представления точки зрения пользователя на базу данных.
2. Множество допустимых операций, применимых к базе данных, которая находится в допустимом состоянии. Составляет основу языка данных модели.
3. Ограничения целостности, определяющие множество допустимых состояний базы данных.

***Определение.*** *Под схемой базы данных будем понимать ее описание средствами языка определения данных.*

Модель служит для описания свойств данных на протяжении всего периода создания информационной системы. На ранних этапах она дает возможность понять суть информационной составляющей предметной области, оценить состав и взаимосвязь данных. На этапе проектирования БД модель показывает, как структура данных и ограничения целостности представляются в терминах выбранной СУБД. На этапе реализации (развертывания) модель демонстрирует, где и как расположены данные и как к ним обратиться. Схема описывает свойства базы данных в терминах типов хранящихся в ней данных. В зависимости от уровня модели, для которой она предназначена, различают концептуальную, логическую, физическую схемы.

Процесс данных проектирования начинается с установления концептуальных требований, формируется ***концептуальная модель***, которая представляет объекты и их связи без указания способов физического хранения. Затем она переводится в модель данных, совместимую с выбранной СУБД, которая называется ***логической моделью***. Наконец, логическая модель отображается на физическую память: определяется расположение данных и метод доступа к ним. Это внутренняя, ***физическая модель***. Соответственно, рассматривают три уровня проектирования, которые более полно будут рассматриваться в лекции, посвящённой проектированию информационных систем.

При проектировании данных полезно руководствоваться набором простых правил, предложенных в [2]. Соблюдение каждого из них позволит обеспечить нормаль-

ную работу информационной системы, для которой разрабатывается база данных. Вот эти правила:

- основа проектирования – концептуальные требования;
- БД удовлетворяет информационным потребностям;
- БД удовлетворяет требованию производительности;
- БД удовлетворяет вновь возникающим требованиям;
- БД расширяется в соответствии с расширением ПО;
- БД изменяется в зависимости от изменения программной и аппаратной среды;
- в процессе функционирования БД не изменяется корректность данных;
- контроль достоверности данных производится до записи в БД;
- доступ к данным осуществляется с учетом полномочий.

Теперь приведем ещё одно определение СУБД с учетом того, что она должна поддерживать эти правила.

***Определение.*** *СУБД – набор программных средств, позволяющих*

- *обеспечить пользователя языковыми средствами манипулирования данными – языками определения данных (ЯОД) и языками манипулирования данными (ЯМД);*
- *обеспечить поддержку моделей пользователя;*
- *обеспечить реализацию ЯОД и ЯМД: отображение операций над данными в операции над физическими данными;*
- *обеспечить защиту (полномочия) и целостность (согласованность) данных.*

### *Модель «сущность-связь»*

Объекты, которыми оперирует концептуальная модель, представляют собой сущности, которые могут быть в некотором отношении друг к другу. Отношение сущностей показывает их связь. Модели, представляющие собой совокупность сущностей и связей, называются моделями «сущность-связь» или *ER*-моделями [20]. Сущности и связи имеют содержательную (смысловую) интерпретацию. Создание *ER*-модели сопровождается ее графическим представлением в форме *ER*-диаграмм. Различные нотации *ER*-диаграмм поддерживаются специальными средствами проектирования программных систем (*CASE*-средствами).

Модель «сущность-связь» была предложена П.Ченом в 1976 г. для концептуального представления структур данных. В роли основных элементарных данных предметной области выступают *сущности*. Данные могут находиться в некотором отношении друг с другом: образовывать ассоциации, которые называются *связями*. Сущности и связи описываются *атрибутами*.

Связи могут быть двуместными и многоместными, ориентированными и неориентированными. Двуместные связи ещё называются бинарными. Бинарные связи характеризуются кардинальными числами. Если одному экземпляру первой сущности соответствует единственный экземпляр другой, тип связи называется один к одному (*1:1*, обозначается ↔). Если одному экземпляру первой сущности соответствует более одного экземпляра другой, тип связи – один ко многим (*1:M*, обозначается ↔→). Если одному экземпляру первой сущности соответствует более одного экземпляра другой, а одному экземпляру второй сущности соответствует более одного экземпляра первой, говорят о связи многие ко многим (*M:N*, обозначается ←↔→). Такие же типы связей определяются и для атрибутов.

В модели Чена сущности обозначаются прямоугольниками, в которые помещены имена атрибутов. Первичные атрибуты выделены. Связи помечаются ромбиками, в которых указано кардинально число связи. Связь характеризуется атрибутами. В современных представлениях модели «сущность-связь» связи атрибутов не имеют.

Пример

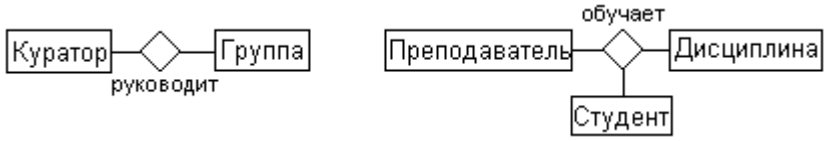



Рис. 2.1. Пример бинарной и тернарной связей

Таблица 2.1. Сущности и атрибуты модели «Факультет»

| *Сущность* | *Атрибут* |
|---|---|
| Студент | зачётка, паспорт, имя_студ, адрес, группа, курс |
| Преподаватель | имя_преп, специальность |
| Группа | номер, год образования |
| Дисциплина | название, специальность |
| Куратор | имя, группа |

***Типы связей***

| *Сущность 1* | *Сущность 2* | *Тип* |
|---|---|---|
| Куратор | Группа | 1:1 |
| Студент | Группа | 1:M |
| Преподаватель | Дисциплина | M:N |

| *Атрибут 1* | *Атрибут 2* | *Тип* |
|---|---|---|
| зачётка | паспорт | 1:1 |
| группа | зачётка | 1:M |
| имя_студ | имя_преп | M:N |

Конец примера

Помимо прочего, в модели вводится ограничение целостности данных, ассоциируемое с двумя множествами сущностей: зависимость по существованию.

### *Независимость данных*

Понятие независимости данных весьма существенно для построения корректной модели данных. Оно непосредственно относится к взаимоотношению моделей, представляющих различные уровни проектирования: концептуальный, логический и физический.

В процессе проектирования на каждом уровне создаётся собственная модель, которая характеризуется своим уровнем абстракции, своей терминологией, своими изобразительными средствами. В идеальном варианте модель верхнего уровня порождает множество моделей следующего, а выбор модели следующего уровня не влияет на модель предыдущего. Другими словами, выбор той или иной СУБД на этапе построения логической модели не должен вызывать изменений в концептуальной, а выбор способа размещения данных в физической модели не должен влиять на логическую. Эти, казалось бы, естественные требования выполняются не всегда. Не так редко при обсуждении концептуальной модели предлагают ту или иную конкретную СУБД, что провоцирует концептуальное проектирование в её терминах. В результате переход на другую СУБД влечёт существенную переработку концептуальной модели, что, конечно крайне нежелательно по соображениям времени и стоимости разработки. То же относится и к паре моделей «логическая и физическая».

Если работа программиста на уровне логической модели с использованием средств доступа, предоставляемых СУБД, не зависит от физического расположения и методов доступа к данным, говорят, что обеспечивается *физическая независимость*.

Если выбор СУБД не влияет на концептуальную модель, говорят о *логической независимости* данных.

*Данные независимы, если обеспечивается логическая и физическая независимость.*

# Лекция 3. Ранние модели данных, применяемые в СУБД

Обычно рассматриваются три основных модели представления данных, которые отличаются ограничениями, накладываемыми на представление данных и виды связей. Это, в порядке хронологии появления, иерархическая, сетевая и реляционная модели. В настоящее время развиваются и другие типы моделей: объектно-ориентированный и слабоструктурированный. Возможно, появятся и другие, но их изучение выходит за рамки данного курса.

## *Иерархическая модель*

Как только стало ясно, что модели обработки данных, основанные на плоских файлах, себя исчерпали, и следует переходить к базам данных, возник вопрос о модели, которая будет в них реализована для доступа к данным. Исторически первой была идея иерархического устройства хранилища. Иерархическая модель данных – модель с отношением подчиненности между данными.

***Определение.*** *Отношение объектов иерархической модели определяется следующими правилами:*

*(А подчинено Б)* $\Rightarrow$ *(Б не подчинено А);*

*(А подчинено Б) & (А подчинено В)* $\Rightarrow$ *(Б тождественно В),*

*где А, Б, В – объекты. Другими словами, иерархическая модель реализуется древовидной структурой.*

Объекты иерархических моделей представлены узлами деревьев, которыми описываются модели. Они представляют собой записи, которые ещё называются *сегментами*. Сегменты состоят из простых элементов данных различных типов. Родительской записи соответствует *произвольное количество* экземпляров подчинённых записей каждого типа. Экземпляр записи каждого типа идентифицируется уникальным для этого типа ключом. База данных представляет собой набор таких деревьев.

В модели предусматривается множество навигационных операций по структуре базы, операция поиска сегмента. Доступ к данным производится через цепочку предков. Разумеется, существуют операции манипулирования данными: вставка, модификация и удаление данных. Вставка допускает как горизонтальное, так и вертикальное добавление узлов. Удаление производится естественным каскадным способом, то есть при удалении записи удаляются все её подчинённые.

В модели поддерживается концепция текущего состояния. Фиксируется текущее положение указателя экземпляра сегмента после последней навигационной операции. Дальнейшее перемещение может производиться относительно этого положения.

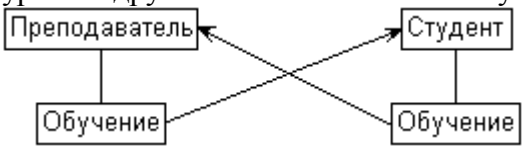
Пример

Рассмотрим две модели, описывающие связь «преподаватель-студент». В первой (А) преподаватель выше по иерархии, во второй (Б) – студент. Первая полезна, когда нужно выяснить, кого учил преподаватель, вторая – когда студент хочет узнать, у кого он учился.


В этих моделях при работе с соответствующими базами данных возникают нежелательные явления – аномалии.

Аномалия включения: В БД (А) невозможно включить студента, который пока ни у кого не обучался, в (Б) невозможно включить не обучающего преподавателя.
Аномалия удаления: так как при удалении сегмента исключаются и все его подчинённые, то при удалении преподавателя из (А) исчезают все его студенты, при удалении студента из (Б) – все его преподаватели.

На второй схеме демонстрируется своеобразный выход из положения: организуются две иерархические базы со ссылками из нижних уровней одной к верхнему уровню другой. Аномалий в этом случае нет, но работа с данными замедляется.




Конец примера

***Достоинства модели***

- простота в понимании и использовании;
- обеспечение определенного уровня независимости по сравнению с файловыми системами;
- простота обслуживания и оценки характеристик, так как благодаря дисциплине удаления нет нужды заботиться о висячих ссылках;
- высокая скорость доступа.

***Недостатки модели***

Основной недостаток – невозможность реализации отношения «многие ко многим» в рамках одной базы данных. Реализация такого отношения на основе двух БД затрудняет управление.

Первая СУБД, построенная по иерархической модели – *IMS* компании *IBM* (1969 г.). Она до сих пор эксплуатируется на разных платформах. Кроме того, примером иерархической СУБД может служить ДИАМС, входящий в программное обеспечение ЭВМ типа СМ-4 (зарубежный аналог – *MAMS* фирмы *DEC*). Есть вариант этой СУБД и для персональных ЭВМ.

## *Сетевая модель*

Вскоре после того, как стал понятен главный недостаток иерархической модели, начались поиски альтернативных подходов к определению подходящей модели данных. Наиболее удачным вариантом оказалась ***сетевая (графовая) модель***, основанная на рекомендациях рабочей группы по базам данных КОДАСИЛ (*CODASYL*), которые стали фактически первым стандартом в области систем баз данных (1969-1978 гг.). В дальнейшем изложении используются терминология и ограничения, введенные этим комитетом. Заметим, что не все графовые модели основаны на этих рекомендациях.

Согласно рекомендациям, база данных делится на области, те – на записи, состоящие из полей. С другой стороны, база состоит из наборов, а те – из записей. Набор включает владельца и подчинённых участников (членов) набора, которые объединяются в линейный список.

Пример

Рассмотрим ту же связь «преподаватель-студент» в сетевой модели. Здесь, помимо сущностей «преподаватель» и «студент», добавляется сущность «обучение», которая подчинена двум предыдущим. Связь «преподаватель-обучение» означает «преподаватель провёл занятие», «студент-обучение» – «студент посетил занятие».


Конец примера

***Определения***

*Запись – иерархия, образованная из простейших (атомарных) элементов данных, их групп и повторяющихся групп.*

*Множество допустимых экземпляров записи называется типом записи.*

*Тип набора – множество экземпляров набора. К набору относятся записи – участники набора и владелец набора. Записи-владельцы (участники) могут быть одновременно владельцами (участниками) других наборов.*

Первое определение совпадает с определением записи в языке КОБОЛ. Третье определение задает отношение «владелец-участник».

Доступ к участнику набора производится только через владельца. Тип набора имеет имя (уникальность владельца). Универсальный владелец в сетевой модели – это СУБД. Через нее выполняется доступ к самым «верхним» владельцам. Допускается и набор без владельца – *сингулярный* набор. В этом случае доступ к информации производит система, играя роль универсального владельца.

***Ограничения КОДАСИЛ***

1. Тип набора определяет отношение 1:M между типом записи-владельца и типом записи-участника.
2. Экземпляр типа записи-участника может участвовать только в одном экземпляре типа набора.

В сетевой модели отношение записей представляет собой граф, не имеющий циклов. Иерархическая модель может быть представлена как частный случай сетевой.

Пример

Отношение «преподаватель-студент» – это отношение *M:N*, и согласно ограничению КОДАСИЛ, его впрямую реализовать нельзя. Однако проблема решатся просто, если преподаватель и студент будут владельцами, а обучение – участником набора. Такая связь часто называется *Y-структурой*. Кроме нее, встречаются другие конфигурации связи. На рисунке справа изображены две таких: иерархическая и многочленная.


Конец примера

Экземпляр типа записи в сетевой базе данных может принадлежать разным наборам, но не нескольким экземплярам набора одного типа, то есть здесь запрещено дублирование данных. Принадлежность записи к различным наборам реализуется ссылками.

Наборы в сетевой базе данных могут иметь управляющие атрибуты. Атрибут *«обязательный-необязательный»* определяет поведение СУБД при удалении владельца экземпляра набора. В первом случае члены набора удаляются при удалении владельца, во втором случае – нет. Атрибут *«автоматический-ручной»* определяет способ включения в набор. В первом случае члены набора включаются в нужный экземпляр набора

автоматически, во втором случае в нужный набор запись включается по команде из прикладной программы.

В сетевой базе данных, как и иерархической, существует понятие текущего состояния, это важная концепция языка манипуляции данными (ЯМД) КОДАСИЛ. В сведения о текущем состоянии входит структура базы, структуры наборов, связи и т.п. Наличие этих данных позволяет эффективно осуществлять доступ к данным, контролировать их целостность. Недостаток такого централизованного представления – трудности с изменением структуры данных.

***Достоинства***

- реализуется отношение «многие ко многим»;
- высокая производительность.

***Недостатки***

- трудность реорганизации базы данных, то есть изменения ее структуры. Обычно реорганизация требует выгрузки данных с последующей их загрузкой в БД с новой структурой. При этом важно не только не потерять данные, но и корректно определить ссылки, в противном случае часть информации будет недоступной. Сама процедура реорганизации требует определенной квалификации администратора БД;
- в процессе эксплуатации за счет некорректных удалений, сбоев и т.п. накапливается мусор – данные, к которым нет доступа. Часть этого мусора оказывается вовсе не мусором, а полезной, но утерянной информацией. Для восстановления целостности БД, а также для ее чистки, требуется кропотливая работа администратора;
- слабая выразительность языка запросов. Обычно он позволяет манипулировать лишь одной записью одновременно, программист во время работы должен хорошо представлять себе пути доступа к данным.

Первая СУБД, построенная по сетевой модели – *IDMS* (1971 г.). Правами на нее обладает компания *Computer Associates,* она до сих пор поставляет и развивает эту СУБД. Примером может служить и СУБД *IMAGE/1000* фирмы *Hewlett-Packard.*

## *Пример базы данных на основе на сетевой модели*

### Постановка задачи

Большая организация имеет санатории разного профиля для выделения путевок в них своим сотрудникам (клиентам). Качество путевки определяется должностью клиента и учреждением, в котором он работает. Профиль санатория должен соответствовать профилю заболевания (определяется поликлиникой). В санатории могут быть места трех типов: палаты (коечный ресурс), люксы и комнаты. Они могут находиться в различных состояниях: свободны, заняты, броня, ремонт и т.п. Кроме сотрудников, клиентами могут быть их родственники и обслуживающий персонал санаторного управления. Последние получают путевки согласно выделенным талонам, которые могут иметь разный статус (выделен, выдан, аннулирован). В процессе формирования путевки она тоже может пребывать в разных статусах (предложена, выделена, напечатана, оплачена). Оплата путевки производится в соответствии со льготами, которые имеет данный клиент.

Требуется реализовать информационную систему, которая обеспечивает максимально бесконфликтное распределение мест в санаториях при условии возможного дефицита их в сезон массовых отпусков.

### Диаграмма

Фрагмент схемы БД, соответствующей приведенной постановке задачи, показан на диаграмме. Овалами обозначены владельцы, прямоугольниками – участники. Связи, естественно, направлены от владельцев к участникам.

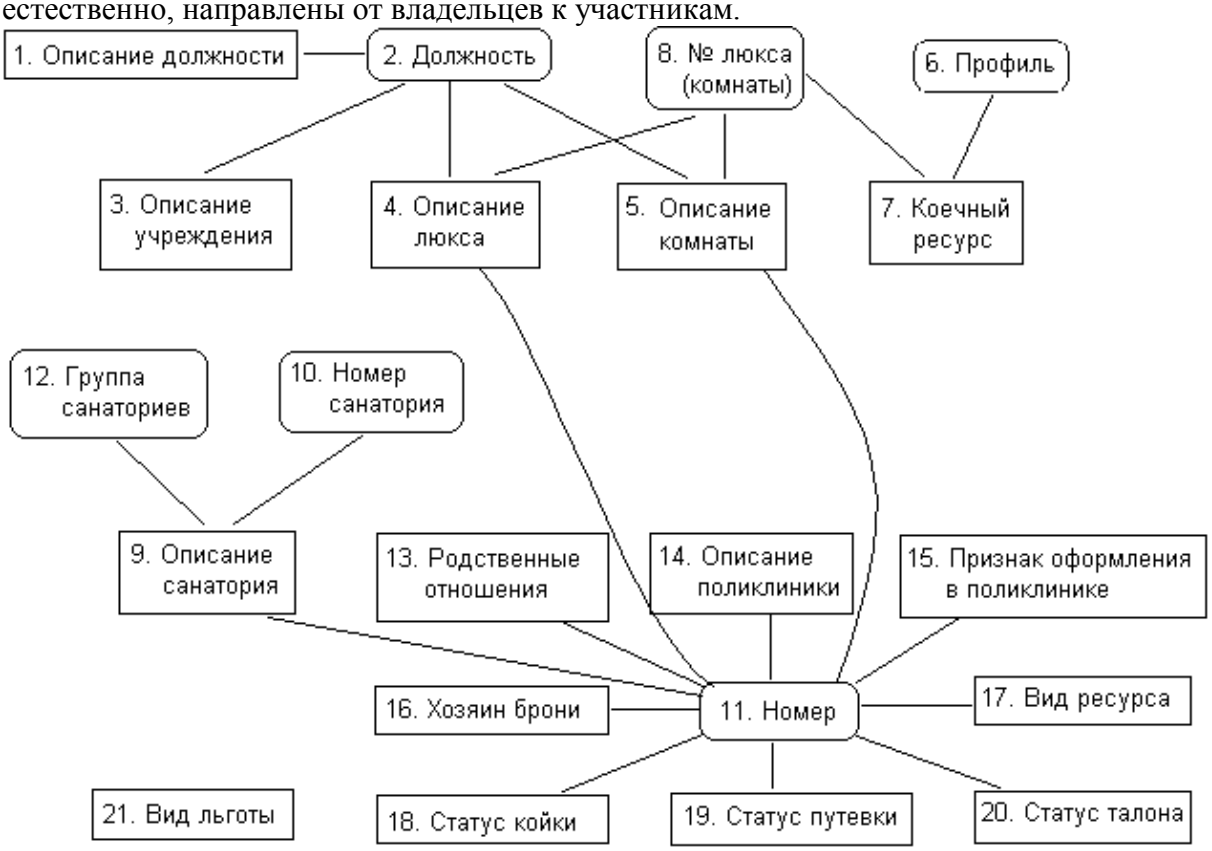



Рис. 3.1. Диаграмма фрагмента базы данных «Санаторий»

Особое место в диаграмме уделяется владельцу «Номер». Он связан со всеми участниками, для которых характерно наличие номера в их определении, например, номер поликлиники, номер путевки, номер (код) родственного отношения. Такое решение позволило избавиться от многочисленных владельцев, представляющих собой номера (коды) сущностей, таких, как «Номер путевки», «Номер талона», «Код люкса» и т.п. Второй интересный набор – это «Вид льготы», не имеющий владельца. Это сингулярный набор, доступ к которому производится непосредственно через СУБД.

### СУБД

База данных реализована в сетевой СУБД *IMAGE/1000* фирмы *Hewlett-Packard*. Устройство ее следующее. Общие сведения о базе данных хранятся в корневом файле. Наборы-владельцы (ключевые наборы), как и наборы-участники (детальные наборы) реализованы в виде отдельных файлов, структура которых различна.

Ключевые наборы представлены файлами постоянной длины, определяемой в момент генерации БД. По характеру доступа это перемешанные таблицы (хэш-таблицы). Доступ к записям производится по значению, которое представляется ключом соответствующей хэш-таблицы. Доступ к записям-участникам производится по прямой ссылке на первую запись, соответствующую данному значению ключа.

Детальные наборы представлены файлами произвольной длины. Их структура – совокупность линейных однонаправленных списков, каждый из которых соответствует одной записи-владельцу. Владелец ссылается на голову списка, остальные записи-участники выбираются по ссылочной цепочке. Каждая запись детального файла может участвовать в наборах, принадлежащих различным типам владельцев. Например, запись описания санатория может относиться к владельцу типа «Номер санатория» и к

владельцу типа «Группа санаториев», но разным номерам санаториев не может соответствовать одна и та же запись описания.

### Описание на ЯОД

В СУБД *IMAGE/1000* реализован специальный ЯОД, подробное описание которого выходит за рамки курса. Приведенный фрагмент определения БД «Санатории» дает достаточное представление о характере этого языка. Пояснения приводятся в комментариях (ограничители комментариев – <<..>>).

```
$CONTROL: TABLE, ROOT, FIELD;
BEGIN DATA BASE: SANAT::29; <<Имя корневого файла и номер диска>>
LEVELS:                     <<Уровни доступа к данным>>
               1  INSP       <<Инспектор>>
               10 MNGR       <<Руководитель>>
               11 ADM        <<Администратор>>
ITEMS:         <<Описание атрибутов:>>
               <<имя, тип, длина, доступ по чтению и записи>>
NOM,           X6(1,1);      <<Номер>>
NAME,          X34(1,1);     <<Текст, соответствующий номеру>>
NOMLK,         I1(1,1);      <<Номер люкса (комнаты)>>
OFFICE,        X6(1,1);      <<Код учреждения>>
OFFNAM,        X34(1,1);     <<Название учреждения>>
POST,          X6(1,1);      <<Код должности>>
POSTNAM,       X34(1,1);     <<Название должности>>
SANAT,         X6(1,1);      <<Код санатория>>
SANAM,         X34(1,1);     <<Название санатория>>
PROFIL,        I1(1,1);      <<Код профиля санатория>>
PRONAM,        X34(1,1);     <<Название профиля санатория>>
ADDRESS,       X80;          <<Адрес>>
WAY,           X80;          <<Дорога в санаторий>>
NLUX,          I1;           <<Число люксов в санатории>>
NROOM,         I1;           <<Число комнат в санатории>>
NPAL,          I1;           <<Число коек в палатах санатория>>
               .  .  .
SETS:

<<             Владелец должности          >>
NAME: NKPOS::29,A;           <<Имя владельца; A – автоматический>>
   ENTRY:                    <<Атрибуты>>
      POST(4),               <<Имя атрибута и количество ссылок>>
CAPACITY: 817;               <<Размер набора (таблицы)>>

<<             Номер люкса (комнаты)       >>
NAME: NOLXKO::29,A;
   ENTRY:
      NOM(4),
CAPACITY: 817;

               .  .  .

<<             Описание люксов             >>
NAME: NOLUX::29,D;           <<Имя участника; D – детальный>>
   ENTRY:
      NOM(NOLXKO),           <<Имя атрибута и ссылка на владельца>>
      NOMLK,
      SANAT(NKLPOS),
      NMEST,
      CATEG(NKLMN),
      QUALIT,
CAPACITY: 101;

<<             Описание должности          >>
NAME: NOPOST::29,D;
   ENTRY:
      POST(NOLXKO),
```

```
      POSNAM,
CAPACITY: 311;

<<            Вид льготы    >>
NAME: NOLGO::29,D;
   ENTRY:
      NOMORD,
      NAME,
CAPACITY: 7;

               .  .  .

END.
```

# Лекция 4. Реляционная модель

## *Принципы*

Реляционная модель данных была предложена Коддом как альтернатива наиболее распространенной в то время сетевой модели. В основу модели Кодд положил три базовых принципа (или, по его словам, три стремления):

1) независимость данных на логическом и физическом уровнях – стремление к независимости;
2) создание структурно простой модели – стремление к коммуникабельности;
3) использование концепции языков высокого уровня для описания операций над порциями информации – стремление к обработке множеств.

Основной побудительной причиной исследований, результатом которой стало создание реляционной модели, стало желание четко разграничить логические и физические аспекты управления БД (первое стремление). В качестве дополнительного результата исследований предполагалась разработка теоретических основ организации и управления БД, то есть создание строгой математической модели.

Для реализации трех принципов пришлось отказаться от принятых ранее принципов структуризации данных (повторяющихся групп, связанных структур). В качестве структурной единицы выбрано отношение *n*-го порядка: при соответствующих операторах и концептуальном представлении в виде таблиц оно позволяет реализовать все три предложенных принципа. Отношение *n*-го порядка – математическое множество, в котором порядок строк не имеет значения. Заметим, что понятие *реляционная* БД несколько шире, чем *табличная*: во втором случае предполагается, что к строке можно добраться по номеру, следовательно, порядок строк имеет значение. Традиционно позиционирование данных определялось адресами в памяти, в реляционной модели адресный способ выбора данных заменен ассоциативным. Каждая единица информации в реляционных БД (РБД) ассоциируется с уникальной тройкой: именем отношения, значением ключа, именем атрибута. При таком подходе система должна сама (а) определить, где следует поместить фрагмент данных, (б) выбрать путь доступа при поиске.

## *Модель*

Напомним, что модель данных – это не только структура, это комбинация, по крайней мере, трех составляющих:

1) типов структур данных,
2) операторов или правил вывода, применимых к правильным типам данных,
3) общих правил целостности, которые определяют множество непротиворечивых состояний БД и множество изменений ее состояний.

Структурную часть реляционной модели составляют следующие компоненты:

- *отношения неопределенного порядка*, концептуально представленные таблицами;
- *атрибуты* – атомарные данные, характеризующие отношения и представленные столбцами таблицы;
- *домены* – множества допустимых значений атрибутов;
- *кортежи* – совокупности значений всех атрибутов отношения, взятых по одному для каждого атрибута, представленные строками таблицы;
- *возможные ключи* – множество атрибутов, однозначно определяющее кортеж в отношении;
- *первичные ключи* – для каждого отношения это один из возможных ключей.

Обрабатывающая часть состоит из операторов выбора, проекции, соединения и т.п., которые преобразуют отношения в отношения.

Часть, относящаяся к целостности, состоит из правил: правила целостности на уровне *объектов* и правила целостности на уровне *ссылок*. В любой реализации можно определить ограничения, которые определят меньшее множество возможных непротиворечивых значений.

Кодд приводит критерии, по которым СУБД можно отнести к реляционным. Эти критерии следующие:

- СУБД должна поддерживать таблицы без видимых пользователю навигационных связей;
- язык манипулирования данными должен обеспечивать минимальную возможность реляционной обработки, то есть включать операторы выбора, проекции и соединения.

Если СУБД не удовлетворяет второму критерию, ее называют *табличной (полуреляционной)*. Реляционные СУБД с минимальной возможностью реляционной обработки называются *минимально реляционными*, если же в СУБД в полной мере реализованы две последние составляющие реляционной модели,– это *полностью реляционные* СУБД. СУБД, реализующие полный набор реляционных операторов, называются *реляционно полными*.

### *Уточнения*

***Определение.*** *Пусть существует n доменов $D_1,\dots,D_n$. Отношение r представляется как подмножество $D_1\times D_2\times\dots\times D_n$, т.е. подмножество упорядоченных n-ок $(d_1,d_2,\dots,d_n)$ – кортежей. Домен $D_i$ представлен i-м элементом. Вместо упорядоченности чаще используют уникальные имена.*

Не каждое отношение может быть объектом реляционной модели. Важное свойство отношений реляционной модели – нормализованность.

***Определение.*** *Отношения* ***нормализованы****, если каждый его атрибут атомарен, то есть, не заменим другим отношением.*

Отношения рассматриваются как множества кортежей. В соответствии с этим считается, что упорядоченность кортежей в отношении не имеет значения, и в отношении нет одинаковых кортежей. Следовательно, всегда существует подмножество атрибутов, которые однозначно определяет кортеж.

***Определение.*** *В отношении существует одна или более групп атрибутов, однозначно идентифицирующих кортеж. Это* ***возможные ключи****. Один из них объявляется* ***первичным****. Атрибуты, входящие в первичный ключ, называются* ***первичными****.*

В отличие от предыдущих моделей данных, в реляционной модели не заданы явные ссылки между таблицами или атрибутами. Между тем, для того, чтобы база данных отражала реальную взаимосвязь объектов предметной области, необходимо как-то ссылаться на данные. В реляционной модели это достигается использованием ключей.

Таким образом, мы сформулировали следующие **свойства отношений**:

1) Нормализованные отношения представляются в виде табличной структуры.
2) Упорядоченность кортежей теоретически несущественна.
3) Все кортежи различны.

В терминологии реляционной модели следующие понятия рассматриваются как синонимы:

*Таблица* синоним *Отношения*,

*Столбец* синоним *Атрибута,*
*Строка* синоним *Кортежа.*

Наконец, осталось дать следующее

***Определение.*** *Реляционная БД – совокупность изменяющихся во времени нормализованных отношений, которые могут быть связаны через общие домены.*

В дальнейших лекциях будет уделено внимание отдельным вопросам реляционной теории, которая лежит в основе модели. Хорошее теоретическое обоснование – одно из основных ее достоинств.

***Основные достоинства реляционной модели*** – теоретическое обоснование, простота определения данных и их реорганизации.
***Недостаток*** – проблемы с организацией связи. Он компенсируется различной степенью нормализации, однако явная связь (сеть, иерархия) чаще более эффективна по времени.

Пример

В приведенном фрагменте БД регистрируются оценки, полученные студентами на экзаменах. Перечень предметов и список преподавателей приведены в отдельных таблицах. Ведомость представлена двумя таблицами: заголовком и списком студентов. Все таблицы связаны ключами. Таблица, из которой производится ссылка, должна содержать так называемый *внешний ключ*. Он может не быть ключом данной таблицы, но его домен либо совпадает с доменом ключа таблицы, на которую ссылается данная, либо сравним с ним.

***Заголовок ведомости***

| *Номер ведомости* | *Код предмета* | *Код преподавателя* | *Дата сдачи* | *Семестр* | *Группа* |
|---|---|---|---|---|---|
| 1 | 10 | 7 | 03.01.2005 | 5 | 306 |
| 2 | 5 | 3 | 07.01.2005 | 7 | 406 |

Ключи – (1) номер ведомости; (2) код предмета, семестр, группа.
Внешние ключи – (1) код предмета; (2) код преподавателя.
Код ведомости имеет уникальное значение. Код предмета, семестр, группа тоже однозначно определяют ведомость. Внешние ключи «Код предмета» и «Код преподавателя» служат для связи с таблицами *Предмет* и *Преподаватель*.

***Предмет***

| *Код предмета* | *Название предмета* |
|---|---|
| 5 | Машинная графика |
| 10 | Базы данных |

Ключи – (1) код предмета.
Таблица представляет собой простой справочник, позволяющий уменьшать избыточность в базе данных.

***Преподаватель***

| *Код преподавателя* | *ФИО преподавателя* | *Должность преподавателя* |
|---|---|---|
| 3 | Чернышов Л.Н. | Доцент |
| 7 | Лукин В.Н. | Доцент |

Ключи – (1) код преподавателя; (2) ФИО преподавателя.
Таблица, как и предыдущая, служит справочником. Она имеет атрибут «Должность», который тоже лучше представить кодом.

***Успеваемость***

| *Номер ведомости* | *Номер студента* | *Оценка* |
|---|---|---|
| 1 | 1 | 5 |
| 1 | 2 | 4 |
| 2 | 1 | 5 |
| 2 | 2 | 3 |
| 1 | 3 | 3 |
| 2 | 3 | 5 |

Ключи – (1) номер ведомости, номер студента.
Внешние ключи – (1) номер ведомости; (2) номер студента.
Основная таблица, отражающая успехи студентов. Каждый кортеж однозначно определяется уникальным кодом ведомости и номером студента в списке группы. Для получения дополнительных атрибутов, например, номера группы, используется внешний ключ «Код ведомости» (связь с таблицей *Заголовок ведомости*), для расшифровки фамилии студента служит внешний ключ «Номер студента».

***Студент***

| *Номер студента* | *Группа* | *Номер зачетки* | *ФИО студента* |
|---|---|---|---|
| 1 | 306 | 9708104 | Иванов И.И. |
| 2 | 306 | 9708035 | Петров П.П. |
| 3 | 306 | 9708117 | Сидоров С.С. |
| 1 | 506 | 9508017 | Алексеев А.А. |
| 2 | 506 | 9508027 | Борисов Б.Б. |
| 3 | 506 | 9508037 | Васильев В.В. |

Ключи – (1) группа, номер студента; (2) номер зачетки.
Каждый из ключей однозначно определяет студента. В зависимости от задачи может использоваться тот или иной ключ, который становится первичным. В нашем случае первичным будет ключ (1).
Конец примера

# Лекция 5. Реляционная алгебра: начальные понятия

## *Схема, отношение. Ключ*

Предмет дальнейшего рассмотрения – реляционная алгебра, которая определяется над отношениями со специфическими операциями. Некоторые понятия, определяемые далее, уже встречались при знакомстве с реляционной моделью. Теперь они даны в более строгом изложении.

Как и раньше, под отношением будем понимать множество кортежей, каждый из которых представлен определенным набором значений атрибутов.

***Определение.*** *Схема – конечное множество имен атрибутов* $\{A_1, A_2, ..., A_n\}$.

Каждому имени $A_i$ ставится в соответствие домен $D_i$: $D_i=dom(A_i)$. Обозначим $D = \cup_{i=1}^{n} D_i$.

***Определение.*** *Отношение r со схемой R – конечное множество отображений* $t=\{t_1, t_2, ..., t_q\}$ *из R в D, причем, для каждого отображения* $t \in r$ *значение каждого атрибута* $A_i$ $(i=1,...n)$ *берется из* $D_i$. *Эти отображения называются кортежами.*

Пример

Дана схема отношений *Рейс = {номер, пункт отправления, пункт назначения, время вылета, время прилета}*. Здесь определены следующие домены:

1) *dom*(*номер*) – множество целых чисел {1..999};
2) *dom*(*пункт отправления*) = *dom*(*пункт назначения*) = {аэропорты};
3) *dom*(*время вылета*) = *dom*(*время прилета*) – время суток (часы, минуты).

Тогда отношение со схемой *Рейс* может выглядеть так:

| | | | | |
|---|---|---|---|---|
| 83 | Новгород | Чита | 11:30 | 17:30 |
| 84 | Чита | Новгород | 20:50 | 3:40 |
| 109 | Новгород | Липецк | 21:50 | 23:50 |
| 213 | Новгород | Байконур | 10:00 | 14:00 |
| 214 | Байконур | Новгород | 16:00 | 20:00 |

Конец примера

Примем следующие обозначения. Будем обозначать первыми заглавными латинскими буквами (*A, B, ...*) имена атрибутов, буквами *R, Q* – схемы атрибутов, строчными первыми (*a, b, ...*) – атрибуты, строчными *r, q* – отношения. Схему $R=\{A_1, A_2, ..., A_n\}$ будем обозначать $R[A_1, A_2, ..., A_n]$ или $A_1A_2...A_n$, отношение *r* со схемой *R* обозначим $r(R)$ или $r(A_1A_2...A_n)$.

Традиционно под кортежем понимается последовательность значений. В реляционной алгебре он обычно трактуется как множество значений, взятых по одному, для каждого атрибута из схемы отношения. Реально кортеж – мультимножество: множество с возможными повторами.

Значение кортежа *t* на атрибуте *A* будем называть *A*-значением кортежа *t* и обозначать $t(A)$. Если $X \subseteq R$, будем называть $t(X)$ *X*-значением кортежа *t*. Предполагается, что существует значение $\lambda$: $t(\varnothing)=\lambda$ для каждого кортежа *t*.

Отношения меняются во времени: кортежи добавляются, удаляются, изменяются, но схема отношения инвариантна. Будем рассматривать отношение как множество возможных состояний, которые оно может принимать.

Ранее мы давали понятие ключа как множества атрибутов, значения которых однозначно определяют кортежи в отношении. Но это толкование ключа слишком ши-

рокое, оно допускает существование подключа, то есть если $K'$ – ключ отношения $r(R)$ и $K' \subset K \subseteq R$, то $K$ тоже ключ.

***Определение.*** *Ключ отношения $r(R)$ – это подмножество $K=\{B_1, B_2, ..., B_m\} \subseteq R$ такое, что для $\forall t_1, t_2 \in r$, $t_1 \neq t_2$, $\exists B \in K$, на котором $t_1(B) \neq t_2(B)$, причём, ни одно $K' \subset K$ не обладает этим свойством.*

Множество атрибутов, включающее ключ как собственное подмножество, называется суперключом.

Заметим, что ключ, по смыслу, должен быть инвариантен ко всем отношениям в схеме. Это определяется семантикой отношения.

Атрибуты, входящие в ключ, будем выделять подчеркиванием, например $r(\underline{AB}CD)$ или $R[\underline{AB}CD]$. Если отношение содержит более одного ключа, каждый из них задается отдельно. Ключи, явно перечисленные в схеме, называются выделенными, остальные – возможными, один из выделенных называется первичным.

### *Изменение отношений во времени*

Напомним, что реляционная база данных определяется как совокупность изменяющихся во времени нормализованных отношений. Самый простой способ изменения содержания отношения – это добавление, удаление и изменение одного кортежа. Далее рассмотрим соответствующие операции.

*Размещение дополнительной информации производится операцией добавления $ADD(r; A_1=d_1, A_2=d_2, ..., A_n=d_n)$.*

Пример

Для добавления рейса из Астрахани в Барнаул достаточно записать

*ADD(Рейс; номер = 117,*
*пункт отправления = Астрахань,*
*пункт назначения = Байконур,*
*время вылета = 22:05,*
*время прилета = 0:43).*

Конец примера

Вариант для фиксированного порядка атрибутов: $ADD(r; d_1, d_2, ..., d_n)$.

Возможные ошибки при добавлении:

1) добавляемый кортеж не соответствует схеме;
2) некоторые значения не принадлежат требуемым доменам;
3) после добавления появляется совпадение по ключу.

В любом случае операция не выполняется.

*Удаление информации производится операцией $DEL(r; A_1=d_1, A_2=d_2, ..., A_n=d_n)$ или $DEL(r; d_1, d_2, ..., d_n)$.*

Если $K=\{B_1, B_2, ..., B_m\}$ – ключ отношения, для удаления достаточно записать $DEL(r; B_1=b_1, B_2=b_2, ..., B_m=b_m)$.

Пример

Если *{пункт отправления, пункт назначения, время вылета}* – ключ отношения *Рейс*, для удаления рейса из Новгорода в Читу следует записать

*DEL(Рейс; пункт отправления = Новгород,*
*пункт назначения = Чита,*
*время вылета = 11:03).*

Если ключ – это *номер рейса*, достаточно записать
*DEL(Рейс; номер = 83).*

Конец примера

Возможная ошибка – отсутствие удаляемого кортежа. Заметим, что допускается удаление последнего кортежа, то есть, пустое отношение может существовать.

*Модификация информации производится операцией изменения. Пусть* $\{C_1, C_2, \ldots C_p\} \subseteq \{A_1, A_2, \ldots A_n\}$. *Тогда операция модификации записывается следующим образом:*
$CH(r; A_1=d_1, A_2=d_2, \ldots, A_n=d_n; C_1=c_1, C_2=c_2, \ldots, C_n=c_p)$
*или, в случае ключа,*
$CH(r; B_1=b_1, B_2=b_2, \ldots, B_m=b_m; C_1=c_1, C_2=c_2, \ldots, C_n=c_p)$.

Пример

Для изменения времени вылета и времени прилета рейса 109 из отношения *Рейс* запишем

*CH(Рейс;* *номер = 109,*
*пункт отправления = Новгород,*
*пункт назначения = Липецк,*
*время вылета = 21:50,*
*время прилета = 23:50;*
*время вылета = 20:00,*
*время прилета = 22:00*).

Если в ключ – это номер рейса, достаточно записать

*CH(Рейс;* *номер = 109;*
*время вылета = 20:00,*
*время прилета = 22:00).*

Конец примера

Того же эффекта можно достигнуть последовательным удалением изменяемого кортежа и добавлением нового. Поэтому ошибки модификации представляют собой объединение ошибок удаления и добавления.

# Лекция 6. Операции реляционной алгебры

Операции, рассмотренные в предыдущей лекции – это операции не над *отношениями*, а над отдельными *кортежами*. Далее мы рассмотрим операции над отношениями. Это, во-первых, обычные булевы операции, а во-вторых, группа специальных операторов.

## *Булевы операции*

К булевым операциям относятся операции пересечения, объединения, разности. Пусть *r, s* – отношения со схемой *R*. Они могут рассматриваться как подмножества множества всех кортежей, определяемых этой схемой, поэтому к ним применимы булевские операции.

Пересечением называется отношение $q(R) = r(R) \cap s(R)$, содержащее кортежи, которые одновременно принадлежат и *r*, и *s*. Объединением называется отношение $q(R) = r(R) \cup s(R)$, содержащее кортежи, которые принадлежат либо *r*, либо *s*. Разностью называется отношение $q(R) = r(R) - s(R)$, содержащее кортежи, которые принадлежат *r*, но не принадлежат *s*. Или формально:

$$r \cap s = \{t | (t \in r) \& (t \in s)\};$$
$$r \cup s = \{t | (t \in r) \vee (t \in s)\};$$
$$r - s = \{t | (t \in r) \& (t \notin s)\}.$$

Заметим, что $r \cap s = r - (r - s)$, то есть достаточно лишь двух операций.

Обозначим *dom(R)* множество всех кортежей над атрибутами из схемы *R* и их доменами: $dom(R) = \{t(d_1\ d_2\ \dots\ d_n) |\ d_i \in dom(A_i)\}$. Дополнение отношения определим как $\bar{r}(R)$: $\bar{r} = dom(R) - r(R)$. Но если какой-либо атрибут из *R* имеет бесконечный домен, $\bar{r}$ будет тоже иметь бесконечное число кортежей, то есть по определению не будет отношением.

***Определение.*** *Пусть $r(A_1, A_2, \dots, A_n)$ – отношение, $D_i = dom(A_i)$. Тогда активный домен атрибута $A_i$ относительно r – это множество $adom(A_i,r) = \{d \in D_i \mid \exists t \in r, t(A_i)=d\}$.*

Пусть *adom(R,r)* – множество всех кортежей над атрибутами из *R* и их активными доменами относительно *r*: $adom(R, r) = \{t(d_1\ d_2\ \dots\ d_n) |\ d_i \in adom(A_i, r)\}$. Тогда активным дополнением *r* будем называть $\tilde{r} = adom(R, r) - r$. Так как число значений атрибутов, принадлежащих кортежам из *r*, конечно, то активное дополнение всегда будет отношением.

Пример

*r* (*A B C*)

| | | |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_1$ | $c_2$ |

*s* (*A B C*)

| | | |
|---|---|---|
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_2$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |

Тогда

$r \cap s$ (*A B C*)

| | | |
|---|---|---|
| $a_1$ | $b_2$ | $c_1$ |

$r \cup s$ (*A B C*)

| | | |
|---|---|---|
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_2$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_1$ | $c_2$ |

*r - s* (*A B C*)

| | | |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_1$ | $c_2$ |

Пусть $D_1=\{a_1, a_2\}$, $D_2=\{b_1, b_2, b_3\}$, $D_3=\{c_1, c_2\}$. Тогда *dom(R)* – все комбинации значений атрибутов из доменов, $\bar{r} = dom(R) - r$ – все комбинации, за исключением тех, что входят в *r*. Активный домен атрибута *B* не содержит $b_3$, поэтому *adom(R,r)* – все комбинации значений, не содержащие $b_3$. Тогда активное дополнение $\tilde{r} = adom(R, r) - r$ – все комбинации из *adom(R,r)* без кортежей из *r*.

Конец примера

## Выбор. Свойства выбора

Пусть теперь *A* – это некоторый атрибут отношения *r*(*R*) и $a \in dom(A)$ – элемент множества значений, которые может принимать отображение *t* на этом атрибуте. Выберем из отношения *r* те кортежи, для которых отображение *t(A)=a*, то есть, на *A* принимает значение *a*, и результат обозначим через $\sigma_{A=a}(r)$. Это унарная операция (применяется к одному отношению), результат которой – новое отношение *r′*(*R*).

***Определение.*** *Выбором* $\sigma_{A=a}(r)$ называется отношение $r'(R) = \sigma_{A=a}(r) \equiv \{ t \in r \mid t(A)=a \}$.

Пример

Пусть отношение *r* – расписание рейсов с атрибутами *номер (№)*, *пункт отправления* (*ПО*), *пункт назначения* (*ПН*), *время вылета* (*ВВ*) и *время прилета* (*ВП*).

*Расписание*

| *(№* | *ПО* | *ПН* | *ВВ* | *ВП)* |
|---|---|---|---|---|
| 119 | Новгород | Чита | 11:30 | 17:30 |
| 94 | Чита | Керчь | 20:50 | 3:40 |
| 117 | Баку | Орёл | 21:50 | 23:50 |
| 216 | Новгород | Москва | 10:00 | 14:00 |
| 217 | Москва | Киев | 16:00 | 20:00 |

$\sigma_{ПО = Новгород}(расписание)$

| *(№* | *ПО* | *ПН* | *ВВ* | *ВП)* |
|---|---|---|---|---|
| 119 | Новгород | Чита | 11:30 | 17:30 |
| 216 | Новгород | Москва | 10:00 | 14:00 |

Конец примера

Пусть *r* и *s* – отношения со схемой *R*; *A, B, C,…* – конечное число атрибутов в *R*, пусть $a \in dom(A)$, $b \in dom(B)$, $c \in dom(C)$,… . Тогда верны следующие утверждения.

***Утверждение 6.1.*** *Операторы выбора коммутативны относительно операции композиции (т.е. результат их применения не зависит от последовательности):*

$$\sigma_{A=a} \circ \sigma_{B=b}(r) \equiv \sigma_{A=a}(\sigma_{B=b}(r)) = \sigma_{B=b}(\sigma_{A=a}(r)) \equiv \sigma_{B=b} \circ \sigma_{A=a}(r).$$

***Доказательство***

$$\begin{aligned}
\sigma_{A=a}(\sigma_{B=b}(r)) &= \sigma_{A=a}(\{ t \in r \mid t(B) = b \}) = \\
&= \{ t \in \{ t' \in r \mid t(B) = b \} \mid t(A) = a \} = \\
&= \{ t \in r \mid t(A) = a, t(B) = b \} = \\
&= \{ t \in r \mid t(B) = b, t(A) = a \} = \\
&= \{ t' \in \{ t \in r \mid t(A) = a \} \mid t'(B) = b \} = \\
&= \sigma_{B=b}(\{ t \in r \mid t(A) = a \}) = \sigma_{B=b}(\sigma_{A=a}(r)).
\end{aligned}$$

Введём следующее обозначение: $\sigma_{A=a} \circ \sigma_{B=b} \equiv \sigma_{A=a,B=b}$. Положим $X = (A, B, C,\ldots)$, а $x = (a, b, c,\ldots)$, тогда оператор выбора можно обозначить $\sigma_{X=x}$. .

***Утверждение 6.2.*** *Операция выбора дистрибутивна относительно бинарных булевых операций:*

$$\sigma_{A=a}(r \gamma s) = \sigma_{A=a}(r)\ \gamma\ \sigma_{A=a}(s), \text{ где } \gamma \in \{\cap, \cup, -\}.$$

***Доказательство***

$$\sigma_{A=a}(r \cap s) = \sigma_{A=a}(\{t \mid t \in r, t \in s\}) =$$
$$= \{t' \in \{t \mid t \in r, t \in s\} \mid t'(A) = a\} =$$
$$= \{t \mid t \in r, t(A) = a\} \cap \{t \mid t \in s, t(A) = a\} =$$
$$= \sigma_{A=a}(\{t \mid t \in r\}) \cap \sigma_{A=a}(\{t \mid t \in s\}) = \sigma_{A=a}(r) \cap \sigma_{A=a}(s).$$

Аналогично доказываются равенства $\sigma_{A=a}(r \cup s)=\sigma_{A=a}(r) \cup \sigma_{A=a}(s)$ и $\sigma_{A=a}(r - s)=\sigma_{A=a}(r) - \sigma_{A=a}(s)$.

***Замечание.*** Операции выбора и активного дополнения не перестановочны (не коммутируют). Можно показать, что $\tilde{\sigma}_{A=a}(r) \subseteq \sigma_{A=a}(\tilde{r})$.

## Проекция. Свойства проекции

Пусть $r$ – отношение со схемой $R$, $X \subseteq R$.

***Определение.*** *Проекцией $\pi_X(r)$ называется отношение $r'(X) = \pi_X(r) \equiv \{t(X) \mid t \in r\}$.*

Это унарная операция, но в отличие от операции выбора, которая выдаёт строки по заданным условиям, она выдаёт столбцы, заголовки которых перечислены в $X$.

Пример

Воспользуемся отношением *расписание* из предыдущего примера:

$\pi_{\{ВВ,\ ВП\}}$(*расписание*) =

| *(ВВ* | *ВП)* |
|---|---|
| 11:30 | 17:30 |
| 20:50 | 3:40 |
| 21:50 | 23:50 |
| 10:00 | 14:00 |
| 16:00 | 20:00 |

$\pi_{ПН}$(*расписание*) =

| *(ПН)* |
|---|
| Чита |
| Керчь |
| Орёл |
| Москва |
| Киев |

Конец примера

Если $Y$ – подмножество $X$, то $\pi_Y(\pi_X(r)) = \pi_Y(r)$. В общем случае, если $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_m$, то $\pi_{X_1} \circ \pi_{X_2} \circ \ldots \circ \pi_{X_m} = \pi_{X_1}$.

***Утверждение 6.3.*** *Операторы проекции и выбора перестановочны относительно композиции:*

$$\pi_X \circ \sigma_{A=a}(r) \equiv \pi_X(\sigma_{A=a}(r)) = \sigma_{A=a}(\pi_X(r)) \equiv \sigma_{A=a} \circ \pi_X(r).$$

***Доказательство***

$$\pi_X(\sigma_{A=a}(r)) = \pi_X(\{t \in r \mid t(A) = a\}) =$$
$$= \{t'(X) \mid t' \in \{t \in r \mid t(A) = a\}\} =$$
$$= \{t(X) \mid t \in r, t(A) = a\} =$$
$$= \sigma_{A=a}(\{t(X) \mid t \in r\}) = \sigma_{A=a}(\pi_X(r)).$$

# Лекция 7. Операции реляционной алгебры (продолжение)

## *Соединение*

***Определение.*** *Пусть r(R) и s(S) – отношения. Соединением r и s называется отношение* $r \| s = q(RS) = \{t(RS) \mid \exists t_r \in r, t_s \in s : t_r = t(R), t_s = t(S)\}$.

Пусть $R \cap S = X \neq \varnothing$. Тогда результат соединения состоит из всех кортежей, в которых $t(X)=t_r(X)=t_s(X)$.

Несколько слов о знаках операций. Мы придерживаемся системы обозначений из [17], за исключением знака соединения, который имеет такой ⋈ вид. Его в наборах символов Word мне найти не удалось, и я использовал ⊗. Позже в [16] я увидел похожее обозначение: ►◄, но там оно символизирует тета-соединение. И там же, далее, приведено обозначение для естественного соединения ||, которое мне показалось наиболее подходящим: обычно оно обозначает конкатенацию. В дальнейшем изложении будем пользоваться именно им.

Пример

Пусть в авиакомпании хранится список самолётов, которые могут использоваться на данном рейсе (отношение *r*), и список пилотов и самолётов, которыми соответствующие пилоты имеют право управлять (отношение *s*). Тогда список пилотов, которые могут быть назначены на эти рейсы, определяется соединением *r*||*s*.

*r*(*Рей Самолёт*)

| *Рей* | *Самолёт* |
|---|---|
| 43 | ТУ-154 |
| 43 | ИЛ-86 |
| 105 | ТУ-154 |

*s*(*Пилот Самолёт*)

| *Пилот* | *Самолёт* |
|---|---|
| Сидоров | ТУ-134 |
| Сидоров | ТУ-154 |
| Борисов | ТУ-154 |
| Петров | ИЛ-86 |

*r*||*s*(*Рей Самолёт Пилот*)

| *Рей* | *Самолёт* | *Пилот* |
|---|---|---|
| 43 | ТУ-154 | Сидоров |
| 43 | ТУ-154 | Борисов |
| 43 | ИЛ-86 | Петров |
| 105 | ТУ-154 | Сидоров |
| 105 | ТУ-154 | Борисов |

Конец примера

Пусть $R=(A_1A_2\ldots A_k)$, $S = (C_1C_2\ldots C_m)$, $R \cap S = \varnothing$. Тогда соединение $r \| s$ называется *декартовым произведением* отношений *r* и *s* (обозначается $r \times s$). Его результат – множество кортежей $t(A_1A_2\ldots A_kC_1C_2\ldots C_m)$, таких, что $t(A_1A_2\ldots A_k) \in r$ и $t(C_1C_2\ldots C_m) \in s$. В нём каждый кортеж $t_r \in r$ соединён с каждым кортежем $t_s \in s$.

Пример

Для отношений *r* и *s* их соединение $r \| s = r \times s$

*r* (*A B*)

| *A* | *B* |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |

*s* (*C D*)

| *C* | *D* |
|---|---|
| $c_1$ | $d_1$ |
| $c_2$ | $d_1$ |
| $c_2$ | $d_2$ |

$r \times s$ (*A B C D*)

| *A* | *B* | *C* | *D* |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ |
| $a_1$ | $b_1$ | $c_2$ | $d_2$ |
| $a_2$ | $b_1$ | $c_1$ | $d_1$ |
| $a_2$ | $b_1$ | $c_2$ | $d_1$ |
| $a_2$ | $b_1$ | $c_2$ | $d_1$ |

Конец примера

## Свойства соединения

***Свойство 1.*** *Имитация выбора.*

С помощью оператора соединения найдём $\sigma_{A=a}(r)$ для отношения $r(R)$. Для этого определим отношение $s(A)$ с одним единственным кортежем $t$ таким, что $t(A) = a$. Тогда $r \parallel s = \sigma_{A=a}(r)$.

***Доказательство***

$$r \parallel s = \{t \mid \exists t_r \in r, t_s \in s, t_r = t(R), t_s = t(A)\} =$$
$$= \{t \mid \exists t_r \in r, t_r = t(R), t(A) = a\} =$$
$$= \{t \mid t(A) = a\} = \sigma_{A=a}(r).$$

***Свойство 2.*** *Обобщённая операция выбора.*

Введём новое отношение $s(A)$ с $k$ кортежами $t_1, t_2, \ldots, t_k$, где $t_i(A) = a_i$ и $a_i \in dom(A)$, $i = 1, 2, \ldots, k$. Тогда $r \parallel s = \sigma_{A=a1}(r) \cup \sigma_{A=a2}(r) \cup \ldots \cup \sigma_{A=ak}(r)$.

***Свойство 3.*** *Коммутативность оператора соединения.*

Из определения следует, что $r \parallel s = s \parallel r$.

***Свойство 4.*** *Ассоциативность оператора соединения.*

Для отношений $q, r, s$ $(q \parallel r) \parallel s = q \parallel (r \parallel s)$. Следовательно, последовательность соединений можно записывать без скобок.

***Свойство 5.*** *Многократные соединения.*

Пусть $r_1(S_1), r_2(S_2), \ldots, r_n(S_n)$ – отношения, $R = S_1 \cup S_2 \cup \ldots \cup S_n$. Обозначим $S$ – последовательность схем $S_1, S_2, \ldots, S_n$. Пусть $t_1, t_2, \ldots, t_n$ – последовательность кортежей, $t_i \in r_i$, $i = 1, 2, \ldots, n$.

***Определение.*** *Кортежи $t_1, t_2, \ldots, t_n$ соединимы на $S$, если существует кортеж $t$ на $R$, что $t_i = t(S_i)$ для каждого $i = 1, 2, \ldots, n$. Кортеж $t$ называется результатом соединения кортежей $t_1, t_2, \ldots, t_n$ на $S$.*

Пример

$r_1$ (*A* *B*)

| | |
|---|---|
| *a1* | *b1* |
| *a1* | *b2* |
| *a2* | *b1* |

$r_2$ (*B* *C*)

| | |
|---|---|
| $b_1$ | $c_2$ |
| $b_2$ | $c_1$ |

$r_3$ (*A* *C*)

| | |
|---|---|
| $a_1$ | $c_2$ |
| $a_2$ | $c_2$ |

Кортежи $\langle a_1 b_1 \rangle$, $\langle b_1 c_2 \rangle$, $\langle a_1 c_2 \rangle$ соединимы с результатом $\langle a_1 b_1 c_2 \rangle$, а кортежи $\langle a_2 b_1 \rangle$, $\langle b_1 c_2 \rangle$, $\langle a_2 c_2 \rangle$ – с результатом $\langle a_2 b_1 c_2 \rangle$:

$r_1 \parallel r_2 \parallel r_3$

| (*A* | *B* | *C*) |
|---|---|---|
| $a_1$ | $b_1$ | $c_2$ |
| $a_2$ | $b_1$ | $c_2$ |

Конец примера

Если в определении принять $n=2$ и если кортежи $t_1$ и $t_2$ соединимы на $S=S_1, S_2$ с результатом $t$, то $t_1=t(S_1)$, $t_2=t(S_2)$, следовательно, $t \in r(S_1) \parallel r(S_2)$. Обратно, если $t \in r(S_1) \| r(S_2)$, то должны существовать $t_1$ и $t_2$ в r($S$) такие, что $t_1=t(S_1)$, $t_2=t(S_2)$, то есть они соединимы на $S$ с результатом $t$. Следовательно, r($S_1$) $\| r(S_2)$ состоит из результатов соединений соединимых на $S$ кортежей $t_1$ и $t_2$ .

***Лемма.*** *Отношение $r_1 \| r_2 \| \ldots \| r_n$ состоит из всех кортежей t, которые являются результатом соединения соединимых на S кортежей $t_i \in r_i$, $i = 1, 2, \ldots, n$.*

Не каждый кортеж каждого отношения может войти в соединение.

***Определение.*** *Отношения $r_1, r_2, \ldots, r_n$ полностью соединимы, если каждый кортеж в каждом отношении является членом списка соединимых на S кортежей.*

Пример

Из предыдущего примера $<a_1\ b_2> \in r_1$, $<b_2\ c_1> \in r_2$ не соединимы. При добавлении к $r_3$ кортежа $<a_1\ c_1>$ отношения становятся полностью соединимыми с результатом

$r_1 \| r_2 \| r_3$

| (A | B | C) |
|---|---|---|
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_1$ | $c_2$ |

Конец примера

***Свойство 6.*** *Проекция соединения.*

Свойство показывает связь проекции и соединения. Похоже, что они взаимнообратны, но это не так.
Пусть *r(R)* и *s(S)* – отношения, $q = r \| s$, *RS* – схема *q*. Пусть $r' = \pi_R(q)$, тогда $r' \subseteq r$ (для любого кортежа *t* из отношения *q* верно $t(R) \in r$, а $r' = \{t(R) / t \in q\}$).
Включение может быть собственным:

*r*

| (A | B) |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |

*s*

| (B | C) |
|---|---|
| $b_1$ | $c_1$ |

*r*||*s*

| (A | B | C) |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |

Может быть равенство $(r = r')$:

*r*

| (A | B) |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |

*s*

| (B | C) |
|---|---|
| $b_1$ | $c_1$ |
| $b_2$ | $c_2$ |

*r*||*s*

| (A | B | C) |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_2$ |

Ясно, что равенство получается при соединении полностью соединимых отношений, но может быть и без этого.
Если $s' = \pi_S(q)$, то $r' = r$ и $s' = s$ тогда и только тогда, когда *r* и *s* – состоят из полностью соединимых кортежей, то есть полностью соединимы.

***Свойство 7.*** *Соединение проекций.*

Поменяем местами проекции и соединения. Пусть *q* – отношение со схемой *RS*, $r = \pi_R(q)$, $s = \pi_S(q)$. Пусть $q' = r \| s$. Если $t \in q$, то $t(R) \in r$, $t(S) \in s \Rightarrow t \in q'$, т.е. $q' \supseteq q$. При $q' = q$ отношение разложимо без потерь на схемы *R* и *S*.

***Свойство 8.*** *Соотношение операций объединения и соединения.*

Пусть $r'$ и *r* – отношения со схемой *R* и *s* – отношение со схемой *S*. Покажем, что $(r \cup r') \| s = (r \| s) \cup (r' \| s)$.

Обозначим левую часть равенства как $q \equiv (r \cup r') \| s$, а правую – $q' \equiv (r \| s) \cup (r' \| s)$. Для кортежа $t \in q$ найдутся кортежи $t_r$ и $t_s$ такие, что $t = t_r \| t_s$, причем $t_r \in r$ или $t_r \in r'$ и

$t_s \in s$. Если $t_r \in r$, то $t \in r \parallel s$, если же $t_r \in r'$, то $t \in r' \parallel s$, то есть $q \subseteq q'$. Чтобы установить включение $q \supseteq q'$, выберем $t \in q'$. Тогда $t \in r \parallel s$ или $t \in r' \parallel s$, следовательно, $t \in (r \cup r') \parallel s$. Из $q \subseteq q'$ и $q \supseteq q'$ следует $q = q'$.

# Лекция 8. Операции реляционной алгебры (продолжение)

В этой лекции рассмотрим более сложные реляционные операторы. Некоторые из них являются обобщением ранее определенных операторов, другие эквивалентны их композиции.

## *Деление*

***Определение.*** *Пусть* $r(R)$ *и* $s(S)$ *– отношения*, $S \subseteq R$. *Положим* $R' = R - S$. *Тогда* $r$, *разделенное на* $s$ *– это отношение* $r'(R')=\{t \mid \forall t_s \in s\ \exists t_r \in r: t_r(R')=t\ \&\ t_r(S)=t_s\}$.

Отношение $r'$ – частное от деления $r$ на $s$, что обозначается $r' = r \div s$. Иначе $r \div s$ – это максимальное подмножество $r'$ множества $\pi_{R'}(r)$, такое, что $r' \parallel s \subseteq r$. Соединение здесь – декартово произведение.

Пример

Дано отношение ***право***, отражающее право пилотирования определенных типов воздушных судов, и два множества типов самолетов, представленных в виде отношений ***q*** и ***s*** с одним атрибутом. Для получения информации о пилотах, имеющих право пилотирования самолетов из этих множеств, используется операция деления.

***право***

| *пилот* | *тип самолета* |
|---|---|
| Иванов | ТУ-134 |
| Иванов | ТУ-154 |
| Иванов | ИЛ-86 |
| Петров | ТУ-134 |
| Петров | ТУ-154 |
| Сидоров | ТУ-134 |
| Сидоров | ТУ-154 |
| Сидоров | ИЛ-86 |
| Сидоров | ЯК-40 |
| Голубев | ТУ-154 |

***q***

| *тип самолета* |
|---|
| ТУ-134 |
| ТУ-154 |
| ИЛ-86 |

***право÷q = q′***

| *пилот* |
|---|
| Иванов |
| Сидоров |

***s***

| *тип самолета* |
|---|
| ТУ-134 |
| ТУ-154 |

***право÷s = s′***

| *пилот* |
|---|
| Иванов |
| Петров |
| Сидоров |

Конец примера

Пример

В сводной ведомости содержатся сведения о положительных отметках студентов, полученных на сессии. Необходимо выяснить, кто их студентов не имеет задолженности за эту сессию, если список необходимых экзаменов задан отношением ***План***, а ведомость – отношением ***Ведомость***. Для этого разделим ***Ведомость*** на ***План*** и спроектируем результат на схему, содержащую единственный атрибут *Студент*.

***Ведомость***

| *Студент* | *Предмет* | *Отметка* |
|---|---|---|
| Иванов И.И. | СППО | 5 |
| Петров П.П. | СППО | 5 |
| Сидоров С.С. | СППО | 3 |
| Иванов И.И. | БД | 4 |
| Сидоров С.С. | БД | 5 |

***План***

| *Предмет* |
|---|
| СППО |
| БД |

***Список=$\pi_{Студент}$(Ведомость÷План)***

| *Студент* |
|---|
| Иванов И.И. |
| Сидоров С.С. |

Конец примера

### *Постоянные отношения.*

При обсуждении соединения мы показали, что результат операции выбора может быть получен при выполнении операции соединения с постоянным отношением. Введём специальный способ записи постоянных отношений.

***Определение.*** *Пусть $A_1,\ldots,A_n$ – различные атрибуты, а $c_i$ является константой из $dom(A_i)$ для $1 \le i \le n$, тогда $\langle c_1 : A_1,\ldots,c_n : A_n\rangle$ – постоянный кортеж $\langle c_1,\ldots,c_n\rangle$ над схемой $A_1A_2\ldots A_n$.*

*Постоянное отношение* над схемой $A_1A_2\ldots A_n$ представляется как множество кортежей. Пусть $c_{ij}$ – константа из $dom(A_i)$ для $1 \le i \le n$ и $1 \le j \le k$, тогда

$$\{\langle c_{11}: A_1, c_{12}: A_2 \ldots c_{1n}: A_n\rangle,$$
$$\langle c_{21}: A_1, c_{22}: A_2 \ldots c_{2n}: A_n\rangle,$$
$$\ldots$$
$$\langle c_{k1}: A_1, c_{k2}: A_2 \ldots c_{kn}: A_n\rangle\}$$

представляет отношение, которое обычно записывалось бы так:

| $(A_1$ | $A_2$ | ... | $A_n)$ |
|---|---|---|---|
| $c_{11}$ | $c_{12}$ | ... | $c_{1n}$ |
| $c_{21}$ | $c_{22}$ | ... | $c_{2n}$ |
| ... | ... | ... | ... |
| $c_{k1}$ | $c_{k2}$ | ... | $c_{kn}$ |

В случае, когда отношение состоит из одного кортежа, фигурные скобки иногда опускаются. Для кортежа с единственным атрибутом опускаются угловые скобки.

***Утверждение.*** *Постоянное отношение с любым числом кортежей k и любым числом атрибутов n может быть построено из постоянных отношений с одним кортежем и одним атрибутом с помощью операторов соединения и объединения.*

### *Переименование атрибутов*

Пример

Отношение *использование* определяет назначение конкретного самолета с заданным бортовым номером на рейс в определенную дату.

***использование***

| *рейс* | *дата* | *номер самолета* |
|---|---|---|
| 12 | 06.12.04 | 134-82 |
| 12 | 07.12.04 | 134-82 |
| 13 | 06.12.04 | 134-82 |
| 26 | 06.12.04 | 86-16 |
| 26 | 07.12.04 | 86-18 |
| 27 | 06.12.04 | 86-16 |
| 27 | 07.12.04 | 86-2 |
| 60 | 06.12.04 | 134-82 |
| 60 | 07.12.04 | 154-6 |

Требуется узнать все пары рейсов, которые используют один и тот же самолет в один и тот же день. Для этого хорошо было бы соединить отношение *использование* с его копией, игнорируя связи по столбцу *рейс*. Но для этого нужно, чтобы атрибут *рейс* в копии назывался по-другому, например, *рейс2*. Переименование атрибутов производится соответствующим оператором.

Конец примера

***Определение.*** *Пусть r – отношение со схемой R, A∈R, B∉R – A, dom(A)=dom(B). Пусть R′ = (R – A)B. Тогда r с A, переименованным в B (обозначается* $\delta_{A \leftarrow B}(r)$*) есть отношение* $r'(R') = \{t' \mid \exists t \in r, t'(R - A) = t(R - A) \;\&\; t'(B) = t(A)\}$.

Пример (продолжение)

Отношение с искомыми парами рейсов:

$s = \pi_{\{\text{рейс, рейс2}\}}(\textit{использование} \parallel \delta_{\{\textit{рейс} \leftarrow \textit{рейс2}\}}(\textit{использование})) =$
$= s(\textit{рейс}, \textit{рейс2})$

| | |
|---|---|
| 12 | 13 |
| 13 | 12 |
| 60 | 12 |
| 12 | 60 |
| 13 | 60 |
| 60 | 13 |
| 12 | 12 |
| 13 | 13 |
| 60 | 60 |
| 26 | 27 |
| 27 | 26 |
| 26 | 26 |
| 27 | 27 |

Конец примера

Пусть $r$ – отношение со схемой $R$,
$A_1, \ldots, A_k \in R$;
$B_1, \ldots, B_k \notin R - (A_1 \ldots A_k)$; (1)
$\forall i$: $dom(B_i) = dom(A_i)$.
Обозначим одновременное переименование атрибутов $A_1, \ldots, A_k$ в $B_1, \ldots, B_k$ как $\delta_{A1,\ldots,Ak \leftarrow B1,\ldots,Bk}(r)$. Благодаря условию (1) оно всегда может быть записано в виде последовательности переименований. Если это условие не выполняется, без введения дополнительных атрибутов такую замену выполнить нельзя. Очевидный пример – обмен $\delta_{A, B \leftarrow B, A}$.

### *Эквисоединение, естественное и θ-соединение*

Мы увидели, что отношения можно соединять по одноимённым атрибутам. Чтобы соединить отношения по различным атрибутам с одинаковыми доменами, требуется выполнить две операции: переименование и соединение. Подобные соединения с переименованиями встречаются очень часто, поэтому эту пару операций разумно представить одной.

***Определение.*** *Пусть* $r(R)$, $s(S)$ *– отношения,* $A_i \in R$ , $B_i \in S$ , $dom(A_i) = dom(B_i)$, $1 \le i \le m$, $A_i \ne B_i$. *Эквисоединением r и s по* $A_1, A_2, \ldots, A_m$ *и* $B_1, B_2, \ldots, B_m$ *называется отношение* $q(RS) = \{t \mid \exists t_r \in r, t_s \in s, t(R) = t_r, t(S) = t_s, t(A_i) = t(B_i), 1 \le i \le m\}$.

Эту операцию будем обозначать следующим образом:

$$r\,[A_1 = B_1, A_2 = B_2, \ldots, A_m = B_m]\,s.$$

Пример

Заданы отношения *маршрут,* в котором указаны аэропорты отправления и назначения авиарейсов, и *приписка*, которое определяет аэропорт, где работает пилот. Следует назначить пилотов на рейсы из аэропорта их приписки. Задача решается эквисоединением по столбцам *пункт отправления* и *аэропорт*.

***маршрут***

| *рейс* | *пункт отправления* | *пункт назначения* |
|---|---|---|
| 84 | Чебоксары | Норильск |
| 109 | Норильск | Омск |
| 117 | Казань | Москва |
| 213 | Норильск | Москва |
| 214 | Москва | Норильск |

***приписка***

| *пилот* | *аэропорт* |
|---|---|
| Алексеев | Норильск |
| Борисов | Казань |
| Воронин | Казань |
| Грушин | Москва |
| Дорофеев | Омск |
| Егоров | Чебоксары |

*маршрут* [*пункт отправления* = *аэропорт*] *приписка* =
= ***назначение***

| *рейс* | *пункт отправления* | *пункт назначения* | *пилот* | *аэропорт* |
|---|---|---|---|---|
| 84 | Чебоксары | Норильск | Егоров | Чебоксары |
| 109 | Норильск | Омск | Алексеев | Норильск |
| 117 | Казань | Москва | Борисов | Казань |
| 117 | Казань | Москва | Воронин | Казань |
| 213 | Норильск | Москва | Алексеев | Норильск |
| 214 | Москва | Норильск | Грушин | Москва |

Конец примера

Заметим, что для однозначного определения операции эквисоединения достаточно условия $A_i \neq B_i$ для всех атрибутов, входящих в сравнение. Однако обычно требуют, чтобы в эквисоединении все атрибуты различались по именам, то есть, чтобы $R \cap S = \varnothing$. Это не сильное ограничение, так как путем переименования атрибутов в $s$ и $r$ можно добиться пустого пересечения их схем.

***Замечание.*** *Если в эквисоединении нет сравнений, то оно совпадает с декартовым произведением:* $r\ [\ ]\ s = r \times s$.

Соединение, определённое ранее, будем называть *естественным*.

***Утверждение.*** *Эквисоединение может быть выражено через переименование и естественное соединение.*

Естественное соединение также может быть выражено через эквисоединение. Например, для отношений $r(A, B, C)$, $s(B, C, D)$, атрибутов $B'$ и $C'$ с $dom(B') = dom(B)$, $dom(C') = dom(C)$: $r \parallel s = \pi_{ABCD}(r\ [B = B', C = C']\ \delta_{B, C \leftarrow B', C'}(s))$.

До сих пор при сравнении значений доменов мы пользовались лишь равенством, но их можно сравнивать, используя и неравенства. В общем случае вводится $\Theta$ – множество символов бинарных операций над парами доменов.

***Определение.*** *Если знак сравнения* $\theta \in \Theta$, *а* $A$ *и* $B$ *– атрибуты, то говорят, что* $A$ *θ-сравним с* $B$, *если* $\theta$ *– бинарное отношение в* $dom(A) \times dom(B)$.

***Определение.*** *Атрибут* $A$ *θ-сравним, если он θ-сравним сам с собой.*

Расширим оператор выбора, используя понятие θ-сравнимости. Пусть $r$ – отношение со схемой $R$, атрибут $A \in R$, $a \in dom(A)$ – константа, $\theta \in \Theta$, $A$ θ-сравним. Тогда расширенный оператор выбора $\sigma_{A\theta a}(r) = \{t \in r \mid t(A)\ \theta\ a\}$. Аналогично этот оператор определяется для случая сравнения между атрибутами, с учетом того, что $B \in R$, $dom(B)=dom(A)$: $\sigma_{A\theta B} = \{t \in r \mid t(A)\theta\ t(B)\}$.

Пример

Отношение *время* определяет время вылета рейса из аэропорта отправления и время его прибытия в аэропорт назначения. Применим оператор выбора для нахождения рейсов, у которых время прибытия не превышает 12 часов, и рейсов, у которых время вылета меньше времени прибытия, по крайней мере, на 2 часа (обозначим соответствующую операцию как <<). Кортежи первой выборки ($s$), пометим символом «*», второй ($q$) – «+».

***время***

| *рейс* | *время вылета* | *время прибытия* | | |
|---|---|---|---|---|
| 84 | 15:00 | 17:30 | | + |
| 109 | 11:40 | 12:00 | * | |
| 117 | 22:00 | 00:30 | * | + |
| 213 | 07:00 | 08:00 | * | |

$s = \sigma_{время\ прибытия \le 12.00}$ (*время*)

$q = \sigma_{время\ вылета << время\ прибытия}$ (*время*)

Конец примера

Эквисоединение – это расширенное соединение для сравнения разных столбцов на равенство. Можно не ограничиваться равенством, а воспользоваться любой операцией $\theta \in \Theta$.

***Определение.*** *Пусть* $r(R)$ *и* $s(S)$ *– отношения, для которых* $R \cap S = \varnothing$, *и пусть* $A \in R$ *и* $B \in S$ *θ-сравнимы для* $\theta \in \Theta$. *Тогда θ-соединением называется отношение* $q(RS) = \{t \mid \exists t_r \in r, t_s \in s, t(R) = t_r, t(S) = t_s, t_r(A)\theta t_s(B)\}$, *которое обозначается* $q(RS) = r\ [A\theta B]\ s$.

Пример

***время_ab***

| *рейс* | *вылет* | *прилёт* |
|---|---|---|
| 60 | 9:40 | 11:45 |
| 91 | 12:50 | 14:47 |
| 112 | 16:05 | 18:15 |
| 306 | 20:30 | 22:25 |
| 420 | 21:15 | 23:11 |

***время_bc***

| *рейс* | *вылет* | *прилёт* |
|---|---|---|
| 11 | 8:30 | 9:52 |
| 60 | 12:25 | 13:43 |
| 156 | 16:20 | 17:40 |
| 158 | 19:10 | 20:35 |

В приведенных отношениях заданы времена вылета и прилета самолетов, совершающих рейсы соответственно из пункта $a$ в пункт $b$ и из пункта $b$ в пункт $c$. Требуется узнать, какие рейсы могут проходить транзитом из города $a$ в $c$ через $b$.

*транзит_ac = время_ab* [*прилёт < вылет′*] $\delta_{N,\ прилёт,\ вылет \leftarrow N',\ прилёт',\ вылет}$ (*время_bc*) =

= ***транзит_ac***

| (*N* | *вылет* | *прилёт* | *N′* | *вылет′* | *прилёт′*) |
|---|---|---|---|---|---|
| 60 | 9.40 | 11.45 | 60 | 12.25 | 13.43 |
| 60 | 9.40 | 11.45 | 156 | 16.20 | 17.40 |
| 60 | 9.40 | 11.45 | 158 | 19.10 | 20.35 |
| 91 | 12.50 | 14.47 | 156 | 16.20 | 17.40 |
| 91 | 12.50 | 14.47 | 158 | 19.10 | 20.35 |

Конец примера

### *Реляционная алгебра. Полнота ограниченного множества операторов*

Обозначим ***U*** – множество атрибутов (*универсум*), ***D*** – множество доменов, *dom* – полная функция из ***U*** в ***D***, ***R*** = $\{R_1, R_2, \ldots, R_p\}$ – множество схем, $R_i \subseteq \mathbf{U}$, $d = \{r_1(R_1), r_2(R_2), \ldots, r_p(R_p)\}$ – множество наборов отношений, Θ – множество бинарных отношений над доменами из ***D***.

***Определение.*** *Реляционная алгебра над* ***U***, ***D***, dom, ***R***, d, *Θ – семиместный кортеж* ***B***=( ***U***, ***D***, dom, ***R***, d, *Θ*, ***O***), *где* ***O*** *– множество операторов объединения, пересечения, разности, активного дополнения, проекции, естественного соединения, деления, переименования, которые используют атрибуты из* ***U,*** *и оператор выбора, использующий операторы из Θ.*

***Теорема***. *Для выражения E над реляционной алгеброй существует выражение E' над ней же, которое определяет ту же функцию и использует лишь операторы (1) постоянных отношений с единственным атрибутом и единственным кортежем, (2) выбора с одним сравнением, (3) естественного соединения, (4) проекции, (5) объединения, (6) разности, (7) переименования.*

***Следствие.*** *Для реляционной алгебры с операцией дополнения в формулировке предыдущей теоремы можно заменить операцию разности (пункт 6) операцией дополнения.*

### *Операторы расщепления и фактора*

Следующие два оператора не относятся к реляционной алгебре, так как в результате их применения из одного отношения получаются два, а для операторов реляционной алгебры результат – одно отношение.

***Определение.*** *Пусть* $\beta(t)$ *– предикат на кортежах над R, тогда расщеплением r по* $\beta$ *называется пара отношений (s, s'), каждое со схемой R, такие, что* $s = \{t \in r \mid \beta(t)\}$ *и* $s' = \{ t \in r \mid \bar{\beta}(t)\}$. *Обозначается эта пара* $SPLIT_\beta(r)$.

Пример
Рассмотрим список студентов, приведённый в примере Лекции 4.

***Студент***

| *Номер студента* | *Группа* | *Номер зачетки* | *ФИО студента* |
|---|---|---|---|
| 1 | 306 | 9708104 | Иванов И.И. |
| 2 | 306 | 9708035 | Петров П.П. |
| 3 | 306 | 9708117 | Сидоров С.С. |
| 1 | 506 | 9508017 | Алексеев А.А. |
| 2 | 506 | 9508027 | Борисов Б.Б. |
| 3 | 506 | 9508037 | Васильев В.В. |

Разобьём его на два по группам. Для этого воспользуемся операцией расщепления с предикатом $\beta(t) = (t(Группа) = 306)$. Тогда $SPLIT_\beta(Студент) = (гр306, гр506)$:

***Гр306***

| *Номер студента* | *Группа* | *Номер зачетки* | *ФИО студента* |
|---|---|---|---|
| 1 | 306 | 9708104 | Иванов И.И. |
| 2 | 306 | 9708035 | Петров П.П. |
| 3 | 306 | 9708117 | Сидоров С.С. |

***Гр506***

| *Номер студента* | *Группа* | *Номер зачетки* | *ФИО студента* |
|---|---|---|---|
| 1 | 506 | 9508017 | Алексеев А.А. |
| 2 | 506 | 9508027 | Борисов Б.Б. |
| 3 | 506 | 9508037 | Васильев В.В. |

Конец примера

Пример демонстрирует возможности оператора при организации работы с распределёнными данными. Если по какой-то причине невыгодно держать все данные на одном обрабатывающем узле (например, пользователь определённой группы данных территориально удалён), их следует разделить с помощью оператора расщепления. Для совместного использования данные объединяются оператором объединения.

Следующий оператор тоже используется для распределения данных, но другим способом. В этом случае на две части делится схема отношения, а для восстановления исходного отношения без потерь в обе получившиеся схемы добавляется уникальный атрибут-метка, который принимает одинаковое значение на кортежах, образованных из общего.

***Определение.*** *Пусть дано отношение $r(R)$ и $B_1, B_2, \ldots, B_k \in R$, $L \notin R$. Тогда фактором будем называть пару следующих отношений: $(s, s') = FACTOR(r; B_1, B_2, \ldots, B_k; L)$, где $s = s((R - B_1B_2\ldots B_k)L)$, $s' = s'(B_1B_2\ldots B_kL)$, причём $s$ и $s'$ соединяются по $L$ без потерь.*

Действие последнего оператора рассмотрим на примере.

Пример

Задан список абитуриентов, сдающих вступительные экзамены. Для обеспечения секретности каждый абитуриент на каждом экзамене получает шифр. Проверяющий видит только шифр и работу, но не фамилию или номер экзаменационного листа абитуриента.

Чтобы исключить доступ ко всей информации, разделим отношение на два, добавив в каждое одинаковую метку: *Дело(метка, Группа, Номер экз.листа, ФИО)* и *Шифр(Шифр, метка)*. Воспользуемся оператором *FACTOR*(*Абитуриент*; *шифр; метка*) = *(Дело, Шифр).*

***Абитуриент***

| *Шифр* | *Группа* | *Номер экз.листа* | *ФИО* |
|---|---|---|---|
| 31415 | 08-01 | 20081104 | Иванов И.И. |
| 92653 | 08-01 | 20081205 | Петров П.П. |
| 58979 | 08-01 | 20081007 | Сидоров С.С. |
| 27182 | 08-02 | 20081230 | Алексеев А.А. |
| 81828 | 08-02 | 20081231 | Борисов Б.Б. |
| 45904 | 08-02 | 20081765 | Васильев В.В. |

***Дело***

| *Группа* | *Номер экз.листа* | *ФИО* | *метка* |
|---|---|---|---|
| 08-01 | 20081104 | Иванов И.И. | 1 |
| 08-01 | 20081205 | Петров П.П. | 2 |
| 08-01 | 20081007 | Сидоров С.С. | 3 |
| 08-02 | 20081230 | Алексеев А.А. | 4 |
| 08-02 | 20081231 | Борисов Б.Б. | 5 |
| 08-02 | 20081765 | Васильев В.В. | 6 |

***Шифр***

| *метка* | *Шифр* |
|---|---|
| 1 | 31415 |
| 2 | 92653 |
| 3 | 58979 |
| 4 | 27182 |
| 5 | 81828 |
| 6 | 45904 |

Конец примера

Следующий пример применения оператора фактора не столь очевиден. Мы его используем для получения более компактной базы данных, которая будет, кроме того, более удобной для сопровождения.

Пример

Рассмотрим список отдыхающих в некотором доме отдыха. Во время заезда администратор интересуется у отдыхающих, курят они или храпят. Его цель – для минимизации конфликтов разместить курильщиков с курильщиками, храпящих – с храпящими.

***список***

| *отдыхающий* | *курит* | *храпит* |
|---|---|---|
| Иванов | да | да |
| Петров | да | нет |
| Сидоров | нет | нет |
| Борисов | да | да |
| Васин | нет | нет |
| Алексеев | нет | нет |
| Григорьев | нет | да |

Получившийся список довольно неудобен. Мало того, что он велик, так при попытке учесть ещё какое-нибудь обстоятельство придётся реструктурировать очень большую таблицу. Тогда разделим исходное отношение на два таким образом, чтобы в первом были записаны все комбинации свойств отдыхающих, обозначенные целыми числами (метками), а во втором вместо столбцов со свойствами появился столбец с метками. Набору свойств соответствует метка, взятая из соответствующего кортежа первого отношения. Это достигается применением оператора *FACTOR*(*список*; *курит, храпит; метка*) = *(свойства*, *новый_список*).

***новый_список***

| *отдыхающий* | *метка* |
|---|---|
| Иванов | 1 |
| Петров | 2 |
| Сидоров | 4 |
| Борисов | 1 |
| Васин | 4 |
| Алексеев | 4 |
| Григорьев | 3 |

***свойства***

| *метка* | *курит* | *храпит* |
|---|---|---|
| 1 | да | да |
| 2 | да | нет |
| 3 | нет | да |
| 4 | нет | нет |

Действительно, по определению оператора фактора получившиеся отношения должны быть соединимы по метке и не более того, что мы и наблюдаем. Но в первом примере одному кортежу первого отношения соответствовал единственный кортеж второго, а здесь нет. Для достижения полученного эффекта мы использовали правило: если атрибуты $B_1, B_2, \ldots, B_k \in R$ из определения оператора не содержат ключ, одинаковые кортежи $t(B_1B_2\ldots B_k)$ помечаются одинаковыми метками. В противном случае одинаковыми метками помечаются кортежи $t(R\text{-}B_1B_2\ldots B_k)$.

Конец примера

# Лекция 9. Язык структурных запросов SQL

При изучении данной темы потребуется приводить примеры, которые могут стать довольно громоздкими, если в каждом приводить свои таблицы (отношения). Поэтому везде, где это не оговорено специально, будем использовать таблицы продавцов и покупателей некоторой продукции, а также таблицу заказов.

***Продавец***

| *Ном_прод* | *Имя_прод* | *Город* | *Комиссия* |
|---|---|---|---|
| 11 | Иванов | Москва | 0,12 |
| 12 | Петров | Тула | 0,13 |
| 14 | Сидоров | Москва | 0,11 |
| 17 | Борисов | Киров | 0,15 |
| 13 | Титов | Пенза | 0,10 |

***Покупатель***

| *Ном_пок* | *Имя_пок* | *Город* | *Значимость* |
|---|---|---|---|
| 21 | Комов | Москва | 100 |
| 22 | Емелин | Омск | 200 |
| 23 | Мохов | Тула | 200 |
| 24 | Попов | Рязань | 300 |
| 26 | Окулов | Москва | 100 |
| 28 | Глинка | Тула | 300 |
| 27 | Зимин | Омск | 100 |

***Заказ***

| *Ном_зак* | *Сумма* | *Ном_пок* | *Ном_прод* | *Дата* |
|---|---|---|---|---|
| 301 | 18,7 | 28 | 17 | 03.10 |
| 303 | 767,2 | 21 | 11 | 03.10 |
| 302 | 1900,1 | 27 | 14 | 03.10 |
| 305 | 5160,4 | 23 | 12 | 03.10 |
| 306 | 1098,1 | 28 | 17 | 03.10 |
| 309 | 1713,2 | 22 | 13 | 04.10 |
| 307 | 75,7 | 24 | 12 | 04.10 |
| 308 | 4723,0 | 26 | 11 | 05.10 |
| 310 | 1309,9 | 24 | 12 | 06.10 |
| 311 | 3891,8 | 26 | 11 | 06.10 |

Здесь имена атрибутов обозначают следующее:

- *ном_прод* – уникальный номер продавца;
- *имя_прод* – имя (фамилия) продавца;
- *город* – название города, где размещается продавец или покупатель;
- *комиссия* – комиссионные, которые берет продавец;
- *ном_пок* – уникальный номер покупателя;
- *имя_пок* – имя (фамилия) покупателя,
- *значимость* – степень интереса, который проявляют продавцы к данному покупателю;
- *ном_зак* – номер заказа;
- *сумма* – сумма, на которую оформлен заказ;
- *дата* – дата заказа (день и месяц через точку).

### *Начальные понятия*

Язык, предназначенный для работы с реляционными базами данных – *SQL* (язык структурных запросов – *Structured Query Language*) – был призван продемонстрировать возможность эффективной работы с множествами, представленными отношениями. Возможности языка должны быть достаточными для реализации всех операторов реляционной алгебры. Язык должен быть непроцедурным – предполагалось, что таким образом облегчается работа пользователя, не умеющего программировать.

Большинству этих задач разработанный язык отвечает, однако не всем и не в той степени, как рассчитывали авторы. Во-первых, непонятно, какой смысл вкладывался в термин «структурный». То, что получилось, имеет очень слабое отношение к устоявшемуся понятию «структурное программирование». Далее, может ли язык работать с множествами? Да, и языковые средства для этого достаточно хороши. Однако об эффективности, особенно на первых порах внедрения языка, следует говорить осторожно. Первые реализации были столь громоздки, что при появлении СУБД на персональных ЭВМ об *SQL* даже речи не было. Более того, в течение значительного периода времени его считали умирающим языком и немного жалели о его гибели. Автору в это время приходилось читать лекции по базам данных, на которых слушатели отдавали дань уважения этому языку, но относились к нему как к исторической реликвии. Но все-таки языку удалось устоять, чему в немалой степени способствовал рост производительности ЭВМ, а также более эффективные методы реализации языка. Кроме того, уже существовал большой контингент специалистов по базам данных, владеющим языком и, что очень важно, действовал стандарт *SQL*.

Возвращаясь к оценке языка, заметим, что он действительно позволяет реализовать все операторы реляционной алгебры, но форма представления заметно отличается от алгебраических выражений. И, наконец, непроцедурность. В то время, когда создавался язык, непроцедурные языки были очень модными. Считалось, что запись решения задач естественно представлять в виде непроцедурных соотношений, что избавляло от необходимости перечислять в нужной последовательности действия, приводящие к результату. Предполагалось, что любой постановщик задачи в состоянии сформулировать, что же он хочет получить. Значит, если ему предоставить адекватный язык, он сможет достаточно легко записать нужные соотношения. Практика показала, что это далеко не так. Порой пользователь лучше объясняет, что он делает, чем зачем. Даже для специалистов непроцедурность зачастую порождает значительные трудности, так как сложный запрос к базе данных трудно понять и еще труднее отладить: он обычно не делится на простые операторы. Да и переход от привычного процедурного языка к непроцедурному тоже не так прост.

### *Стандарт* ANSI

Как уже упоминалось, для языка *SQL* существует стандарт, на самом деле – серия последовательных стандартов. В курсе лекций мы будем обращаться к стандарту *ANSI*. Как всегда, промышленные реализации по разным причинам отклоняются от стандартов, причем, не всегда в худшую сторону. Стандарт *ANSI* регламентирует не все особенности реализации языка, кроме того, есть распространенные, но не входящие в стандарт варианты операторов. В подобных случаях будем приводить как версию стандарта, так и промышленную.

#### Типы данных

Типы данных в различных версиях *SQL* могут заметно различаться, что связано с набором типов данных, принятом в конкретной СУБД. В стандарте *ANSI* рекомендуется использовать символьные (например, *CHAR, VARCHAR*) и числовые (*INT, DEC*) типы данных. В СУБД *FoxPro* используется широкий спектр типов данных (различные

формы символьных числовых типов, типы даты и времени и т.п.). Все они отражены в соответствующей реализации *SQL*.

### Интерактивный и встроенный SQL

В стандарте предусмотрено существование двух форм языка: интерактивной и встроенной. Первая форма предполагает работу непосредственно в среде *SQL* и не требует наличия какой-либо другой программной среды. Считается, что запросы формулируются в режиме диалога и выполняются интерпретатором языка. Во втором случае язык каким-то способом встраивается в другую языковую систему и общается с пользователем через нее. То есть, интерфейс обеспечивает среда другого языка, а *SQL* выполняет запросы, поступающие из программы. В чистом виде это реализовано, например, в среде *Delphi*, а *FoxPro* даже включает операторы *SQL* в базовый язык.

### Синтаксис

Кратко и неформально структуру языка можно представить следующим образом. Язык состоит из *команд*, оканчивающихся точкой с запятой. Команды состоят из последовательности *фраз*, разделенных пробелами. Фразы состоят из *ключевого слова*, за которым через пробелы следуют *аргументы*.

*Текст ::= команда; ... команда;*
*Команда ::= фраза ... фраза*
*Фраза ::= Ключевое_слово аргумент аргумент ... аргумент*

### Подразделы SQL

Согласно рекомендации *ANSI*, *SQL* состоит из следующих разделов: Язык Определения Данных (ЯОД или *DDL*), Язык Манипулирования Данными (ЯМД или *DML*), Язык Управления Данными (ЯУД или *DCL*). Последний раздел может включаться в ЯМД. Первый раздел служит для определения схем данных (таблиц), второй – для манипуляций данными, которые выполняются через запросы к базе данных.

## *Простейшие действия*

Определение таблиц – это разовая работа, изменение структуры – эпизодическая, а манипулирование данными – деятельность постоянная. И если изменение содержания базы данных большой сложности не представляет, то выборка необходимой информации может быть довольно замысловатой. Именно на запросы расходуется наибольшее время работы с БД. Поэтому основу языка *SQL* составляют запросы к базе данных. Запросы определяются командой ***Select***.

Простейшие запросы – отображение содержимого всей таблицы, проекция, перестановка атрибутов, выбор. Команда для них имеет следующий формат:

***Select*** [***Distinct***] <*список атрибутов*> ***From*** <*таблица*> [***Where*** <*условие*>]*;*

Здесь <*список атрибутов*> – схема (перечень атрибутов) отношения, которое будет результатом выполнения команды. В список может включаться символ «*», который обозначает всю схему исходной таблицы с именем <*таблица*>. В результирующую выборку попадают атрибуты лишь из тех строк, для которых <*условие*> истинно. Фраза ***Where*** не обязательная, если она отсутствует, в выборку попадают атрибуты из всех строк. Заметим, что в результирующей выборке могут возникнуть повторяющиеся строки, если список атрибутов не содержит ключ. Для удаления дублирующихся строк служит вариант ***Distinct***.

Пример

Выбрать содержимое всей таблицы продавцов (представлены два варианта):

***Select** * **From** Продавцы;*
***Select** ном_прод, имя_прод, город, комиссия **From** Продавец;*

Получить проекцию таблицы продавцов на атрибуты *имя_прод, город* с удалением возможных дублирующихся строк:

***Select Distinct** имя_прод, город **From** Продавец;*

Получить проекцию таблицы заказов на подсхему (*дата, ном_пок, ном_зак*), с другим, по сравнению с исходной схемой, порядком атрибутов:

***Select** дата, ном_пок, ном_зак **From** Заказ;*

Выбрать продавцов из Москвы:

***Select** имя_прод, город **From** Продавец **Where** город='Москва';*

Конец примера

Предикат, определяющий условия выбора, может содержать обычные операции отношения (<, <=, =, <>, >=, >), операцию ***between … and …*** (истина, когда первый операнд не меньше второго и не больше третьего) и логические операции (***and***, ***or***, ***not***). Для задания множества служат круглые скобки, ограничивающие его определение. Множество может определяться как простым перечислением элементов, так и запросом. Последний вариант будет рассматриваться позже. Для определения принадлежности элемента множеству служит операция ***in***. Для символьных значений допускается выбор по маске. Символ «%» маскирует любую цепочку в искомой строке, символ «_» – лишь один символ, фраза ***like*** определяет маску.

Пример

Выбрать продавцов из Москвы, у которых комиссионные меньше 0,1:

***Select** Имя_прод, Комиссия **From** Продавец*
***Where** Город='Москва' **and** Комиссия<0,1;*

Выбрать продавцов как из Москвы, так и из Тулы (два варианта):

***Select** * **From** Продавец*
***Where** Город='Москва' **or** Город='Тула';*

***Select** * **From** Продавец*
***Where** Город **in** ('Москва', 'Тула');*

Выбрать продавцов, у которых комиссионные в интервале от 0,1 до 0,12, причем, в первом случае интервалы включаются, во втором – нет:

***Select** * **From** Продавец*
***Where** Комиссия **between** 0.1 **and** 0.12;*

***Select** * **From** Продавец*
***Where** Комиссия **between** 0.1 **and** 0.12 **and not** Комиссия **in** (0.1, 0.12);*

Выбрать продавцов, у которых первая или вторая буква фамилии – «М»:

***Select** * **From** Продавец*
***Where** Имя_прод **like** 'М%' **or** Имя_прод **like** '_м%';*

Конец примера

Согласно требованиям к реляционным базам данных, атрибуты должны иметь возможность принимать пустое значение. В *SQL* оно обозначается как ***null***. Пустое значение может принимать поле любого типа. Для проверки атрибута на пустоту служит операция ***is null***. Сравнение с пустым значением неопределенно (*unknown*): ***a=null***. Не-

определенное значение возникает и в результате проверки предиката принадлежности: *a* ***in (null).*** Обычно неопределенное значение интерпретируется как *ложь*, но не всегда: ***not*** *ложь – истина*, а ***not*** *unknown – unknown*.

Использование ***not*** в синтаксисе *SQL* более свободно, чем в обычных языках программирования: допустимо и (***not*** *a* ***is null)***, и (*a* ***is not null)***, и ***(not*** *a* ***in*** *(...)),* и *(a* ***not in*** *(...)).*

### *Функции агрегирования*

Функции агрегирования возвращают единственное (скалярное) значение для группы кортежей. Всего различают пять таких функций:

- *COUNT* – количество непустых строк или значений атрибутов, отличных от ***null,*** удовлетворяющих заданному условию;
- *SUM* – сумма значений атрибута;
- *AVG* – среднее значение атрибута;
- *MAX* – максимальное значение атрибута;
- *MIN* – минимальное значение атрибута.

В команде ***Select*** они используются в списке выбираемых полей наряду с именами атрибутов. Их аргументы – имена атрибутов, причем, для *SUM* и *AVG* допустимы аргументы только числового типа.

Рассмотрим подробнее функцию *COUNT*. Она подсчитывает количество всех непустых атрибутов, даже если их значение повторяется. Для исключения повторений используется вариант ***Distinct.*** Вариант *COUNT(*)* подсчитывает количество строк в выборке, включая и пустые, и повторяющиеся. Применять ***Distinct*** здесь нельзя. Вариант ***All*** позволяет считать все непустые значения атрибута, включая повторяющиеся.

Агрегатные функции применимы не только к атрибутам, но и к выражениям, содержащим атрибуты. В этом случае вариант ***Distinct*** запрещен.

Пример

Подсчитать количество продавцов, которые выполняли заказы:

***Select*** *COUNT(****Distinct*** *ном_прод)* ***From*** *Заказ;*

Подсчитать количество продавцов:

***Select*** *COUNT(*)* ***From*** *Продавец;*

Подсчитать количество непустых сумм:

***Select*** *COUNT(****All*** *сумма)* ***From*** *Заказ;*

Подсчитать общую сумму заказов в тысячах рублей:

***Select*** *SUM(сумма/1000)* ***From*** *Заказ;*

Конец примера

### *Группировка*

Нередко при работе с базой данных требуется выполнять какие-то действия с группой записей, объединенных общим свойством. Для этого существует возможность группировки, реализуемая фразой ***Group By***. Записи группируются по одинаковым значениям указанного атрибута. Если при этом в запросе участвует агрегатная функция, она выполняется для каждой группы.

Пример

Выбрать наибольшие суммы заказов для каждого продавца:

***Select*** *MAX(сумма), ном_прод* ***From*** *Заказ*
***Group By*** *ном_прод;*

| *Сумма* | *Ном_прод* |
|---|---|
| 1900,1 | 14 |
| 5160,4 | 12 |
| 1098,1 | 17 |
| 1713,2 | 13 |
| 4723,0 | 11 |

Конец примера

Фраза ***Group By*** может содержать более одного атрибута, по которым группируются записи. В этом случае первый атрибут в списке определяет самую внешнюю группу, второй – группу внутри первой и т.п.

При работе с группами данных может возникнуть необходимость отбирать группы по какому-нибудь признаку, например, определить общие суммы заказов по дням и среди них отобрать те, которые превышают некоторое число. Если бы дело касалось записей, задача легко решалась бы фразой ***Where***: ***…Where*** *сумма >…* . Но к группе она неприменима. Для отбора групп по некоторому признаку используется фраза ***Having***.

Пример

Выбрать дни, в которые сумма заказов превышает 5000:

***Select*** *дата, Sum(сумма)* ***From*** *Заказ*
***Group By*** *дата* ***Having*** *Sum(сумма)>5000;*

Конец примера

Фраза ***Having*** может содержать не только агрегатную функцию, но и любой атрибут. Единственное требование – чтобы этот атрибут в каждой группе имел одинаковое для группы значение.

Пример

Выбрать наибольшие заказы, выполняемые продавцами, номера которых 12 или17:

***Select*** *ном_прод, MAX(сумма)* ***From*** *Заказ*
***Group By*** *ном_прод* ***Having*** *ном_прод* ***in*** *(12, 17);*

Конец примера

И, наконец, естественный запрет на использование вложенных агрегатных функций. Например, для выяснения дня, в который была наибольшая сумма заказов, можно, казалось бы, записать:

***Select*** *дата, MAX(SUM(сумма))* ***From*** *Заказ* ***Group By*** *дата;*

***Group By*** предполагает, что групп будет столько, сколько различных дат. С другой стороны, *SUM(сумма)* применяется к каждой группе, а *MAX(SUM(сумма))* для всех групп выработает единственное значение, что противоречит предыдущему утверждению.

### *Возможности форматирования*

Возможности форматирования отображаемых данных в *SQL* довольно слабые. Можно вместо атрибута использовать выражения или константы, а также упорядочивать и группировать записи.

Если вместо атрибута использовать выражение или константу, соответствующий столбец в выборке, согласно стандарту, становится безымянным (в *FoxPro* столбец получает специальное имя). В случае константы все записи в этой позиции будут иметь

значение, равное этой константе. Нельзя задать выборку с единственным столбцом, определенным константой.

Пример

***Select*** *Имя_прод, город, Комиссия*100, '%'* ***From*** *Продавец;*

| *Имя_прод* | *Город* | | |
|---|---|---|---|
| Иванов | Москва | 12 | % |
| Петров | Тула | 13 | % |
| Сидоров | Москва | 11 | % |
| Борисов | Киров | 15 | % |
| Титов | Пенза | 10 | % |

В данном случае название столбца, обозначающего процент комиссии, названия не имеет, но в конкретных СУБД он по умолчанию именуется. Чтобы задать ему разумное имя, следует переименовать соответствующее выражение:

***Select*** *Имя_прод, город, Комиссия*100* ***As*** *Процент, '%'* ***From*** *Продавец;*

Конец примера

Для упорядочивания записей в выборке используется фраза ***Order by***, в которой, как и в случае группировки, задается список атрибутов. Процесс упорядочивания начинается с самого правого атрибута списка и заканчивается самым левым. Каждый атрибут может быть снабжен признаком ***Asc*** для возрастающего порядка или ***Desc*** для убывающего. По умолчанию записи упорядочиваются по возрастанию значения указанного атрибута. Атрибут из списка ***Order by*** должен быть указан в выборке.

Вместо имени в списке можно задавать номер атрибута из списка выбора, если, например, столбец задан агрегатной функцией. Однако при этом возникают неприятности при отладке программы: любое изменение списка выбора чревато ошибкой в списке ***Order by***. Гораздо лучше переименовать выбираемые данные так, чтобы имена столбцов были понятными. Операция переименования реализуется конструкцией

*<выбираемое выражение>* ***As*** *<имя>*.

Упорядочивание может сочетаться с группировкой, в этом случае ***Order by*** выполняется последним.

Особый случай возникает, когда атрибут принимает значение ***null***. Стандарт упорядочивания не определяет его положение в списке, в некоторых реализациях считается, что это значение наименьшее, в других – что наибольшее.

Пример

***Select*** *ном_зак, сумма, дата* ***From*** *Заказ*
***Order By*** *дата, сумма* ***Desc*** *;*

| *Ном_зак* | *Сумма* | *Дата* |
|---|---|---|
| 305 | 5160,4 | 03.10 |
| 302 | 1900,1 | 03.10 |
| 306 | 1098,1 | 03.10 |
| 303 | 767,2 | 03.10 |
| 301 | 18,7 | 03.10 |
| 309 | 1713,2 | 04.10 |
| 307 | 75,7 | 04.10 |
| 308 | 4723,0 | 05.10 |
| 311 | 3891,8 | 06.10 |

| | | |
|---|---|---|
| 310 | 1309,9 | 06.10 |

***Select*** *ном_прод, MAX(сумма)* ***As*** *макс_сумма* ***From*** *Заказ*
***Group By*** *ном_прод* ***Order By*** *ном_прод;*

| *Ном_прод* | *макс_сумма* |
|---|---|
| 11 | 4723,0 |
| 12 | 5160,4 |
| 13 | 1713,2 |
| 14 | 1900,1 |
| 17 | 1098,1 |

***Select*** *Ном_прод, Max(сумма)* ***From*** *Заказ*
***Group By*** *Ном_прод* ***Order By*** *2* ***Decs****;*

| *Ном_прод* | |
|---|---|
| 12 | 5160,4 |
| 11 | 4723,0 |
| 14 | 1900,1 |
| 13 | 1713,2 |
| 17 | 1098,1 |

Конец примера

# Лекция 10. Язык структурных запросов SQL (продолжение)

### *Соединение*

В *SQL* реализуются различные варианты эквисоединения. Известно, что с помощью переименования и эквисоединения можно реализовать естественное соединение. Но в *SQL* к переименованию прибегать не обязательно: допускается именование атрибута с указанием имени таблицы, к которой он относится. Таким образом, можно обеспечить уникальность имен даже в случае непустого пересечения схем соединяемых отношений. Соответствующая конструкция – *<имя таблицы>.<имя атрибута>*.

Остановимся на единственном варианте эквисоединения, отвечающем операции реляционной алгебры, различные варианты оператора соединения ***Join***, входящего в *SQL*, рассматривать не будем. Напомним, что эта операция записывается как $r[A_1 = B_1, A_2 = B_2,\ldots, A_m = B_m]s$, где $r$ и $s$ – соединяемые отношения, а $A_1, A_2,\ldots, A_m$ и $B_1, B_2,\ldots, B_m$ – атрибуты, по равенству которых производится соединение. В *SQL* соответствующий оператор выглядит так:

***Select*** *<список атрибутов>* ***From*** *r, s*
***Where*** $A_1 = B_1$ ***and*** $A_2 = B_2$ ***and***… ***and*** $A_m = B_m$***;***

Вместо знака равенства может стоять знак любой операции сравнения, лишь бы она была определена на соответствующих доменах. В этом случае говорим о $\theta$-соединении.

Рассматриваемый вид соединения основан на ссылочной целостности. Если для какого-либо значения одного из сравниваемых атрибута нет равного ему значения второго, запись в выборку не включается. Таким образом, если рассматривать один из атрибутов как родительский ключ, а другой как внешний, ссылающийся на него, необходимым условием включения записи в выборку будет существование значения родительского ключа, равного любому внешнему. Это и есть условие ссылочной целостности. В качестве родительского ключа чаще всего выбирается первичный.

Пример

Подобрать продавцов и покупателей, живущих в одном городе:

***Select*** *имя_пок, имя_прод, Покупатель.город* ***From*** *Покупатель, Продавец*
***Where*** *Покупатель.город=Продавец.город;*

| *Имя_пок* | *Имя_прод* | *Город* |
|---|---|---|
| Комов | Иванов | Москва |
| Комов | Сидоров | Москва |
| Мохов | Петров | Тула |
| Глинка | Петров | Тула |
| Окулов | Иванов | Москва |
| Окулов | Сидоров | Москва |

Заметим, что в списке выбираемых атрибутов *имя_пок* и *имя_прод* однозначно определяют таблицы, из которых выбираются. Атрибут *город* есть в обеих таблицах, поэтому следует уточнить, атрибут какой таблицы выбирается. В данном случае это безразлично, так как их значения равны.

В следующем примере *ном_пок* в таблице *Покупатель* играет роль первичного (родительского) ключа, а в таблице *Заказ* – внешнего.

***Select*** *имя_пок, ном_зак* ***From*** *Покупатель, Заказ*
***Where*** *Заказ.ном_пок=Покупатель.ном_пок;*

Так как на один внешний ключ в нашем случае приходится ровно один родительский, количество записей в выборке будет равно количеству записей в таблице *Заказ*. Если бы какое-то значение внешнего ключа не было найдено среди родительских, записей стало бы меньше (нарушение ссылочной целостности). Наоборот, если бы для одного значения внешнего ключа нашлось более одного значения родительского, записей стало бы больше (нарушение условия однозначности первичного ключа).
Конец примера

Соединение может применяться не только к двум таблицам. Таблиц может быть как больше, так и меньше, то есть одна таблица может соединяться сама с собой. Во втором случае возникает сложность, связанная с тем, что соединяемые таблицы должны иметь разные имена, иначе одноименные атрибуты различить невозможно. Для выхода из этого положения в SQL таблице может быть присвоено локальное, временное имя таблицы, которое используется наряду с основным. Это имя называется *переменной области определения*, или *переменной корреляции*, или *псевдонимом*. Таблица может иметь несколько псевдонимов. Существует два формата определения псевдонима:

***Select*** *<список атрибутов>* ***From*** *r пн1, s пн2 …*

***Select*** *<список атрибутов>* ***From*** *r* ***As*** *пн1, s* ***As*** *пн2 …*

Здесь *пн1* и *пн2* – псевдонимы таблиц *r* и *s* соответственно.

Пример
Найти заявки покупателей из городов, не совпадающих с городами продавцов. В этом случае соединяются три таблицы:

```
Select ном_зак, имя_пок From Заказ, Покупатель, Продавец
    Where Заказ.ном_пок=Покупатель.ном_пок
        and Заказ.ном_прод=Продавец.ном_прод
        and Покупатель.город<>Продавец.город;
```

Найти покупателей, имеющих попарно равную значимость. В этом случае одна таблица соединяется с собой же:

```
Select a.имя_пок As пок1, b. имя_пок As пок1, a.значимость
    From Покупатель a, Покупатель b
    Where a.значимость=b.значимость;
```

| *пок1* | *пок2* | *Значимость* |
|---|---|---|
| Комов | Окулов | 100 |
| Комов | Комов | 100 |
| Комов | Зимин | 100 |
| Окулов | Окулов | 100 |
| Окулов | Комов | 100 |
| Окулов | Зимин | 100 |
| Зимин | Зимин | 100 |
| Зимин | Окулов | 100 |
| Зимин | Комов | 100 |
| Емелин | Мохов | 200 |
| Емелин | Емелин | 200 |
| Мохов | Мохов | 200 |
| Мохов | Емелин | 200 |
| Попов | Попов | 300 |
| Попов | Глинка | 300 |
| Глинка | Попов | 300 |
| Глинка | Глинка | 300 |

Очевидно, что такой результат вряд ли может считаться удовлетворительным. Нужно избавиться от повторов. Для этого существует следующий нехитрый прием:

```
Select a.имя_пок, b.имя_пок, a.значимость From Покупатель a, Покупатель b
    Where a.значимость=b.значимость;
        and a.имя_пок < b.имя_пок;
```

| *пок1* | *пок2* | *Значимость* |
|---|---|---|
| Комов | Окулов | 100 |
| Зимин | Окулов | 100 |
| Зимин | Комов | 100 |
| Емелин | Мохов | 200 |
| Глинка | Попов | 300 |

Конец примера

### *Вложенные запросы*

В *SQL* множество значений может задаваться не только перечислением входящих в него элементов, но и запросом. Так же, как и список элементов, запрос заключается в круглые скобки. Полученное множество используется так же, как и рассмотренное ранее. Множество содержит атомарные элементы, поэтому формирующий его запрос должен выбирать только один атрибут. В результате запроса множество может быть пустым, содержать один элемент или более одного. Если множество содержит один элемент, оно может рассматриваться как обычное значение, и для него определены соответствующие операции сравнения. Для множества, содержащего более одного элемента, эти операции, естественно, недопустимы, определена лишь операция принадлежности элемента множеству. Заметим, что в результате запроса может образоваться несколько одинаковых значений и, как и в случае обычного запроса, по умолчанию все они включаются в выборку. То есть, на самом деле речь идет не о множестве, а о мультимножестве (комплекте). Использование варианта ***Distinct*** приводит к исключению повторяющихся элементов.

Запрос, формирующий множество, внешне выглядит как обычный, его называют вложенным запросом. Вложенный запрос сам может иметь вложенный запрос.

Пример

Найти все заказы, которые выполняет продавец Сидоров:

```
Select * From Заказ
    Where ном_прод=(Select ном_прод From Продавец
                        Where имя_прод='Сидоров');
```

Запрос корректен, если есть ровно один продавец Сидоров. Если такого нет – результат не определен, если их более одного – результат ошибочен.
Найти все заказы, сумма которых больше, чем средняя за 4 октября:

```
Select * From Заказ
    Where сумма>(Select AVG(сумма) From Заказ Where дата='04.10');
```

Найти все заказы для продавцов из Москвы:

```
Select * From Заказ
    Where ном_прод in (Select ном_прод From Продавец
                        Where город='Москва');
```

Такой же результат можно получить, используя операцию соединения:

```
Select a.* From Заказ a, Продавец b
```

***Where** a.ном_прод = b.ном_прод **and** город='Москва';*

Найти комиссионные всех продавцов, обслуживающих покупателей из Москвы:

***Select** комиссия **From** Продавец*
***Where** ном_прод **in** (**Select** ном_прод **From** Заказ*
***Where** ном_пок **in** (**Select** ном_пок **From** Покупатель*
***Where** город='Москва'));*

Определить количество покупателей, имеющих значимость, превышающую среднюю для покупателей из Тулы:

***Select** значимость, COUNT(**Distinct** ном_пок) **From** Покупатель*
***Having** значимость > (**Select** AVG(значимость) **From** Покупатель*
***Where** город='Тула')*
***Group By** значимость;*

Конец примера

Соединение и вложенные запросы, приведенные в примере, выполняются разными способами, поэтому, несмотря на то, что выборка получается одинаковой, время, затраченное на выполнение запроса, может существенно разниться. Это же касается и необходимой памяти. Выбор варианта неоднозначен, он зависит как от размеров таблиц, включая промежуточные, так и от используемой СУБД.

### *Связанные запросы*

Для реализации вложенного запроса может потребоваться информация из таблиц внешнего запроса. Запросы такого типа называются *связанными*.

Пример

Выбрать всех покупателей, которые сделали заказы 3 октября:

***Select** * **From** Покупатель a*
***Where** '03.10' **in** (**Select** дата **From** Заказ b **Where** a.ном_пок=b.ном_пок);*

Конец примера

При вычислении простого вложенного запроса внутренний подзапрос вычисляется автономно до выполнения внешнего запроса. Связанный запрос так выполнен быть не может: выполнение внутреннего подзапроса зависит от состояния таблиц, объявленных во внешнем запросе. Следовательно, внутренний подзапрос должен выполняться для каждой строки внешней таблицы, от которой он зависит. Итак, сформулируем правила выполнения связанных запросов.

1. Выбрать очередную строку внешней таблицы, от которой зависит внутренний подзапрос, она называется *строкой-кандидатом*.
2. Сохранить ее под псевдонимом, указанном во внешней фразе ***From***.
3. Выполнить подзапрос. Используемые в подзапросе значения строки-кандидата называются *внешними ссылками*.
4. Оценить результат внешнего запроса на основании результата подзапроса.
5. Повторить процедуру для следующих строк внешней таблицы.

Как и в случае вложенных запросов, связанный запрос можно заменить соединением. Здесь тоже нужно внимательно отнестись к выбору варианта, от этого может существенно зависеть производительность.

Пример

Реализовать предыдущий пример, используя соединение:

```
Select Distinct a.* From Покупатель a, Заказ b
    Where a.ном_пок=b.ном_пок and дата='03.10';
```

Очевидно, ***Distinct*** в этом случае необходим, если хотя бы один покупатель сделал более одного заказа. В результате эта операция потребует значительного времени, что снизит эффективность запроса.

Выбрать имена и номера продавцов, обслуживающих более одного покупателя:

```
Select ном_прод, имя_прод From Продавец a
    Where 1<(Select COUNT(Distinct ном_пок) From Заказ
        Where ном_прод=a.ном_прод);
```

Конец примера

Связанный запрос во внутреннем подзапросе может содержать ту же таблицу, что и во внешнем, то есть таблица связывается сама с собой. В этом случае одна и та же таблица в разных подзапросах играет разные роли, но обработка идет по приведенным ранее правилам. Разумеется, одновременно с выборкой строки-кандидата фиксируется и состояние таблицы, чтобы восстановить его после выполнения подзапроса.

Пример

Для каждого покупателя выбрать все заказы на сумму, большую средней суммы его заказов:

```
Select * From Заказ a
    Where сумма>(Select AVG(сумма) From Заказ b
        Where a.ном_пок=b.ном_пок);
```

Конец примера

Так же, как и в случае вложенного запроса, связанный подзапрос может использоваться во фразе ***Having***, если условие накладывается на группу записей.

### *Предикаты, определенные на подзапросах*

В состав логических выражений *SQL* могут входить предикаты, определенные на подзапросах: признак того, что подзапрос не пуст (***Exists***), признак того, что все элементы удовлетворяют некоторому условию (***All***) и признак того, что существует хотя бы один элемент, удовлетворяющий некоторому условию (***Any***, ***Some***).

Функция ***Exists*** истинна, если ее аргумент (подзапрос) содержит хотя бы один элемент, в противном случае она ложна. Легко видеть, что в подзапросе этой функции использование агрегатных функций бессмысленно. Рассмотрим применение функции ***Exists*** на примерах.

Пример

Выбрать всех покупателей из Тулы, если хотя бы один из них сделал заказ:

```
Select * From Покупатель
    Where город='Тула' and Exists
        (Select * From Заказ Where ном_пок in
            (Select ном_пок From Покупатель Where город='Тула'));
```

Заметим, что здесь в подзапросе используется конструкция ***Select*** *… Использование ее в ***Exists*** – это единственный случай корректного употребления варианта «*» в подзапросе.

Выбрать номера всех продавцов, у которых более одного покупателя:

```
Select Distinct ном_прод From Заказ a
```

```
Where Exists(Select * From Заказ b Where a.ном_прод=b.ном_прод
    and a.ном_пок<>b.ном_пок);
```

В данном случае ***Exists*** используется в сочетании со связанным подзапросом. Дополним предыдущий пример соединением: выберем имена всех продавцов, у которых более одного покупателя:

```
Select Distinct a.ном_прод, имя_прод From Заказ a, Продавец c
    Where Exists(Select * From Заказ b
            Where a.ном_прод=b.ном_прод and a.ном_пок<>b.ном_пок)
        and a.ном_прод=c.ном_прод;
```

Конец примера

Предикат ***All*** имеет вид ***All*** *<переменная> θ (<подзапрос>)*, где *θ* – операция сравнения. Он принимает истинное значение, если каждое значение подзапроса удовлетворяет условию, определяемому операцией *θ*. Естественно, этот предикат редко может применяться для операции равенства: в этом случае все элементы выборки должны быть равны между собой. Неравенство – более содержательная операция, она обозначает, что левая часть не равна ни одному из элементов выборки. Правда, этот предикат легко реализуется операцией ***in***. Более интересны операции «больше», «меньше» и т.п. Рассмотрим применение предиката ***All*** на примере.

Пример

Выбрать всех покупателей, значимость которых выше значимости любого покупателя из Тулы:

```
Select * From Покупатель
    Where значимость>All(Select значимость From Покупатель
        Where город ='Тула');
```

Этот же запрос можно сформулировать с ***Exists***:

```
Select * From Покупатель a
    Where not Exists(Select * From Покупатель b
        Where a.значимость<=b.значимость and b.город='Тула');
```

Конец примера

Предикат ***Any*** (синоним – ***Some***) имеет вид ***Any*** *<переменная> θ (<подзапрос>)*, где *θ* – операция сравнения. Он принимает истинное значение, если хотя бы одно значение подзапроса удовлетворяет условию, определяемому операцией *θ*. Этот предикат чаще используется для операции равенства, чем для «больше», «меньше» и т.п. Рассмотрим применение предиката ***Any*** на примере.

Пример

Выбрать всех покупателей, живущих в городе, где есть продавец:

```
Select * From Покупатель Where город= Any(Select город From Продавец);
```

Этот же запрос можно сформулировать с ***in***:

```
Select * From Покупатель Where город in (Select город From Продавец);
```

С ***Exists*** этот запрос выглядит так:

```
Select * From Покупатель a Where
    Exists(Select * From Продавец b Where a.город=b.город);
```

Конец примера

Любой запрос как с ***Any***, так и с ***All***, может быть выражен через ***Exists***, но в этом случае часто требуется более глубокая вложенность запросов и нередко встречаются связанные подзапросы.

Уточним, как ведут себя предикаты ***Any, All*** и ***Exists*** в особых случаях. Для пустой выборки ***All*** принимает значение *истина*, ***Any*** и ***Exists*** – *ложь*. При сравнении с ***null*** предикаты ***Any*** и ***All*** принимают неопределенное значение, ***Exists*** никогда неопределенное значение не принимает.

### *Объединение*

Операция объединения реализуется фразой ***Union***, которая объединяет два независимых подзапроса. Эта операция объединяет два множества в одно, значит, элементы исходных множеств должны быть однотипными. Понятно, что в каждом подзапросе должно быть одинаковое количество столбцов, и они должны быть сравнимы. С точки зрения стандарта *ANSI* сравнимость сводится к тому, чтобы тип и размер столбцов из каждой пары совпадали. Другое ограничение связано с допустимостью ***null***-значений. Если они запрещены в одном подзапросе, они должны быть запрещены и в другом. Нельзя использовать ***Union*** в подзапросах, а в объединяемых выборках – агрегатные функции. В конкретных реализациях могут быть и другие ограничения.

При объединении из результата автоматически исключаются тождественные строки, в отличие от команды ***Select***. Чтобы их оставить, следует использовать вариант ***Union All***.

Для получения требуемого порядка строк в выборке используют, как обычно, фразу ***Order By***, она ставится единственный раз, причем, в конце.

Пример

Выбрать номера покупателей, значимость которых выше 200 или которые сделали заказ на сумму более 3000:

***Select*** *ном_пок* ***From*** *Покупатель* ***Where*** *значимость>200*
***Union***
***Select*** *ном_пок* ***From*** *Заказ* ***Where*** *сумма>3000* ***Order By*** *ном_пок;*

Конец примера

Объединение – операция над двумя операндами, поэтому для объединения более чем двух выборок используют скобочные конструкции:

***(Select*** *<выборка 1>* ***Union [All] Select*** *< выборка 2>)*
***Union [All]***
***(Select*** *< выборка 3>* ***Union [All] Select*** *< выборка 4>) ...*

Очевидно, что различный порядок выполнения действий при использовании ***Union*** и ***Union All*** может привести к разным результатам.

### *Изменение базы данных*

Понятие «изменение базы данных» имеет, по крайней мере, две стороны: изменение содержания существующей БД и изменение ее структуры, в том числе – создание и уничтожение таблиц.

#### Изменение содержания

Добавление данных в таблицу выполняется командой ***Insert***:

***Insert Into*** *<таблица>* ***Values*** *(<знач$_1$>, <знач$_2$>, ...<знач$_N$>);*

Здесь номер каждого значения соответствует номеру атрибута в схеме отношения. Согласно стандарту *ANSI*, пустое значение (***null***) не может быть записано в табли-

цу непосредственно. Однако оно возникает, если воспользоваться ключевой формой записи, при которой указывается два списка: список атрибутов и список их значений. В этом случае могут быть указаны не все атрибуты:

***Insert Into*** *<таблица> (<атр$_1$>, <атр$_2$>, …<атр$_k$>)*
***Values*** *(<знач$_1$>, <знач$_2$>, …<знач$_k$>);*

Вместо прямого указания значений можно использовать запрос. Как и ранее, существует два варианта: с заполнением всех атрибутов без их перечисления и заполнение части атрибутов с их указанием в списке.

***Insert Into*** *<таблица>*
***Select … From … Where …;***

***Insert Into*** *<таблица> (<атр$_1$>, <атр$_2$>, …<атр$_k$>)*
***Select … From … Where …;***

Очевидные замечания.

1. Таблица уже должна существовать.
2. Полученные значения по смыслу должны соответствовать схеме таблицы или списку атрибутов.
3. Типы (домены) выбираемых значений должны соответствовать типам (доменам) атрибутов таблицы.
4. Добавленные записи не должны нарушать уникальность ключа.

Следует обратить внимание на одну важную особенность добавления в *FoxPro*, а, возможно и в других СУБД. Там есть удобные варианты, позволяющие не перечислять значения, а загрузить их из переменных памяти или массива: ***Insert Into*** *<таблица>* ***From Memvar***. Если один из атрибутов объявлен как автоматически наращиваемый (*autoincrement*), при добавлении данных регистрируется ошибка: этот атрибут считается доступным только для чтения. И тогда действительно приходится перечислять все значения. Это неудобно, так как при добавлении в схему базы данных одного атрибута или замене его имени потребуется проверить и исправить все места, где может быть изменение. В противном случае изменение производилось бы автоматически, правда, вероятность ошибки была бы больше.

И ещё одна возможность: в качестве значения *<знач$_i$>* может быть задано любое допустимое выражение.

Пример

Добавить заказ в таблицу заказов:

***Insert Into*** *Заказ* ***Values*** *(337, 5500, 28, 17, '07.10');*

Добавить итоги в таблицу итогов, если она заранее создана:

***Insert Into*** *Итоги (дата, итог)*
***Select*** *дата, SUM(сумма)* ***From*** *Заказ* ***Group By*** *дата;*

Конец примера

Исключение строк из таблицы производится командой ***Delete***. Существует два варианта команды: для очистки таблицы целиком и для удаления найденных строк.

***Delete From*** *<таблица>;*
***Delete From*** *<таблица>* ***Where*** *<условие>;*

Изменение значений полей производится командой ***Update***, которая имеет, как и команда удаления, две модификации. В первом варианте меняются значения атрибутов во всей таблице, во втором – в пределах выборки:

***Update*** *<таблица>*

***Set*** *<атр$_1$>=<знач$_1$>, <атр$_2$>=<знач$_2$>, …<атр$_k$>=<знач$_k$>;*
***Update*** *<таблица>*
***Set*** *<атр$_1$>=<знач$_1$>, <атр$_2$>=<знач$_2$>, …<атр$_k$>=<знач$_k$>*
***Where*** *<условие>;*

Как и при добавлении, в команде изменения в качестве значения может быть задано выражение. Кроме того, все команды изменения содержимого базы данных в условии могут использовать подзапросы.

Пример

Увеличить значимость всех покупателей из Тулы на 10:

***Update*** *Покупатель* ***Set*** *значимость=значимость+10*
***Where*** *город='Тула';*

Увеличить комиссионные всем продавцам, имеющим более трех покупателей, на 0,1:

***Update*** *Продавец* ***Set*** *комиссия=комиссия+0.1*
***Where*** *3<(****Select*** *COUNT(****Distinct*** *ном_пок)* ***From*** *Заказ*
***Where*** *Заказ.ном_прод=Продавец.ном_прод);*

Конец примера

### Изменение структуры

Следующая группа команд относится к созданию таблиц, их удалению, изменению их структур, а также к созданию и изменению индексов.

Создание таблицы производится следующей командой (фрагменты в квадратных скобках [] могут отсутствовать):

***Create Table*** *<таблица>*
*(<атр$_1$> <тип$_1$> [<(размер$_1$)>] [<ограничения$_1$>],*
*<атр$_2$> <тип$_2$> [<(размер$_2$)>] [<ограничения$_2$>],*
*. . .*
*<атр$_N$> <тип$_N$> [<(размер$_N$)>] [<ограничения$_N$>]*
[, ***Primary Key*** *(<первичный ключ>)];*
[, ***Foreign Key*** *(<внешний ключ>)*
***References*** *<таблица2> <родительский ключ>*]*);*

Здесь

- *<таблица>* – имя создаваемой таблицы,
- *<атр>* – имя атрибута,
- *<тип>* – тип атрибута,
- *<(размер)>* – размер атрибута,
- *<ограничения>* – перечень ограничений, наложенных на атрибут,
- *<первичный ключ>* – перечень атрибутов, входящих в первичный ключ таблицы,
- *<внешний ключ>* – перечень атрибутов, входящих во внешний ключ таблицы,
- *<таблица2>* – таблица, на которую ссылается данная,
- *<родительский ключ>* – атрибуты в *таблице2*, на которые ссылается внешний ключ.

Ограничения на атрибут накладываются для контроля целостности. Они могут быть разными: запрет на пустоту (***not null***), указание на уникальность значения (***unique***), на то, что этот атрибут – первичный ключ (***primary key***) или внешний ключ (***references*** *<таблица2>*(*<родительский ключ>*)), на необходимость удовлетворять условиям (***check***(*<предикат>*)), на значение по умолчанию (***default***=<значение>).
Первичный ключ описывается отдельно, если в его состав входит более одного атрибута. Список атрибутов, составляющих первичный ключ, записывается через запятую. То

же относится и к внешнему ключу, состоящему из нескольких атрибутов. Только здесь нужно обеспечить сравнимость соответствующих атрибутов из внешнего и родительского ключей. Обязательно нужно обеспечить ссылочную целостность, то есть, на каждое значение внешнего ключа должно быть соответствующее значение родительского.

Пример

Создать таблицы, которые служили для примеров, в среде *FoxPro*. Указание на СУБД нужно для определения типов данных. Несмотря на то, что в современных версиях реализованы все стандартные типы *SQL*, работать со специфическими типами удобнее.

```
Create Table Продавец
        (ном_прод       Numeric(2)      primary key,
         имя_прод       Char(40)        not null,
         город          Char (20)       default='Москва',
         комиссия       Numeric(4,2)    check(комиссия<1));
Create Table Покупатель
        (ном_пок        Numeric(2)      primary key,
         имя_пок        Char(40)        not null,
         город          Char (20)       default='Москва',
         значимость     Numeric(3));
Create Table Заказ
        (ном_зак        Numeric(3)      primary key,
         сумма          Numeric(7,1)    not null,
         ном_пок        Numeric(2)      references Покупатель(ном_пок),
         ном_прод       Numeric(2)      references Продавец(ном_прод),
         дата           Date            not null);
```

Конец примера

Изменение структуры таблицы выполняется командой ***Alter Table***, которая не поддерживается стандартом *ANSI*. Тем не менее, в реализациях она применяется. Сразу заметим, что применение ее на рабочей таблице рискованно, лучше создать новую таблицу и загрузить в нее данные из старой. С помощью команды ***Alter Table*** можно добавлять и удалять атрибуты, изменять их описание, изменять описание таблицы. Формат команды для добавления атрибута:

***Alter Table*** *<таблица>* ***Add*** *<атр> <тип>* [*<(размер)>*] [*<ограничения>*]*;*

Удаление таблицы может быть выполнено тогда, когда она пустая, то есть у нее предварительно были удалены все данные. Команда удаления:

***Drop Table*** *<таблица>;*

Для ускорения поиска в таблицах *SQL* предоставляет возможность пользоваться индексами. Создание индекса производится следующей командой:

***Create Index*** *<имя индекса>* ***On*** *<таблица> (<*$атр_1$*>, <*$атр_2$*>, ...<*$атр_k$*>);*

Если создается уникальный индекс, вместо ***Create Index*** используется вариант ***Create Unique Index***. Удаление индекса производится командой

***Drop Index*** *<имя индекса>;*

На этом краткое знакомство с языком *SQL* и предоставляемыми им возможностями заканчивается. Подробно его можно изучить по литературе, приведенной в библиографии. Для знакомства с конкретной реализацией следует обратиться к специальной литературе.

# Лекция 11. Понятие о нормальных формах

В лекции дано краткое описание нормальных форм представления реляционных баз данных. Понятия, связанные с нормальными формами 2 и 3, будут уточняться в последующих лекциях, форма Бойса-Кодда, форма 4 и форма 5, помимо данной лекции не обсуждаются. Назначение этой лекции – дать по возможности содержательное представление о теме, чтобы была более ясной цель дальнейших формальных рассуждений.

Важнейшие цели, которым служит база данных – это снижение избыточности данных и повышение надежности хранения информации. Любое априорное знание об ограничениях на данные может служить этим целям. Один из способов формализации этих знаний – установление зависимости между данными, которая отражает их семантику. Семантическая информация выражается множеством функциональных зависимостей схемы. Считается, что функциональная зависимость имеет место, если значение кортежа на одном множестве атрибутов единственным образом определяет их на другом. Определим это понятие более точно.

***Определение.*** *Пусть R – схема отношения, $X,Y \subseteq R$. Множество атрибутов Y функционально зависит от X тогда и только тогда, когда в любой момент времени для каждого из различных значений Y существует только одно из различных значений X. Другими словами, для любого r(R), $t_i, t_j \in r$, $t_i(Y) \neq t_j(Y) \Rightarrow t_i(X) \neq t_j(X)$.*

Эквивалентный термин: множество *X* определяет *Y*. Обозначение – $X \rightarrow Y$. Левую часть функциональной зависимости *X* называют *детерминантом*.

Пример

Рассмотрим отношение, заданное следующей схемой:

*график (Пилот, Рейс, Дата, Время).*

Ясно, что допустимо не любое сочетание значений атрибутов. Их зависимость задается следующими ограничениями:

- для каждого рейса определено лишь одно время вылета;
- для атрибутов (*Пилот, Дата, Время*) определен лишь один рейс;
- для атрибутов (*Рейс, Дата*) определен единственный пилот.

Таким образом, задано множество функциональных зависимостей:

$Рейс \rightarrow Время$

$(Пилот, Дата, Время) \rightarrow Рейс$

$(Рейс, Дата) \rightarrow Пилот$

Конец примера

Некоторые функциональные зависимости могут быть нежелательны в конкретной схеме, так как они при модификации базы данных приводят к трудностям, называемым аномалиями модификации (аномалиями добавления, изменения и удаления). Для приведения схемы в корректный вид используется замена одного множества отношений другим, сохраняющим ее эквивалентность. Такое преобразование составляет суть процесса нормализации. В результате исходное небольшое число таблиц с «большой» схемой, обладающее непривлекательными свойствами, заменяется б**о**льшим числом таблиц с «меньшей» схемой, этими свойствами не обладающих. Говорят, что полученные отношения удовлетворяют некоторой нормальной форме.

Нормальные формы, в которых находятся отношения, составляют иерархию, в которой формы с большими номерами не обладают некоторыми нежелательными свойствами, характерными для форм с меньшими номерами. В теории нормальных форм для реляционных БД рассматривается шесть уровней нормализации: 1НФ – 5НФ и форма Бойса-Кодда (промежуточная между 3НФ и 4НФ). Каждый из следующих

уровней ограничивает типы допустимых функциональных зависимостей отношения. Функциональные зависимости отношения описывают его семантику. Уровень нормализации зависит от семантики отношения.

Надо заметить, что процесс нормализации не всегда сопровождает проектирование данных. Чаще всего в процессе построения информационной модели проектировщик, руководствуясь естественным порядком построения отношений, строит их сразу в третьей нормальной форме. Дейт в [6] приводит убедительные рассуждения по этому поводу. Более того, он утверждает, что отношения, полученные при проектировании, будут сразу в пятой нормальной форме, если проектировщик не злонамерен. В этом с ним солидарна Атре [2], утверждающая, что вполне достаточно владеть навыками проектирования отношений в 3НФ. Тем не менее, нормализация отношений требуется на этапе сопровождения (развития) программной системы, когда выявляются не известные ранее функциональные зависимости, в результате чего отношения теряют нормальную форму.

### *1 нормальная форма (1НФ)*

Согласно определению отношения, все его атрибуты атомарны, то есть не могут быть разделены семантически на более мелкие элементы. Отношение, обладающее этим свойством, называется нормализованным или, что то же самое, находящимся в первой нормальной форме (1НФ).

***Определение.*** *Отношение находится в 1НФ, если все значения его атрибутов атомарны, то есть, для каждого отношения* $r(R)$*, если* $A \in R$*,* $adom(A,r)$ *атомарен* **.**

Пример

Пусть для отношения со схемой *рейс (Номер, Пункт назначения, Вылет)* атрибут *Вылет* определен как пара *(День, Время)*:

| ***рейс*** | *(Номер,* | *Пункт назначения,* | *Вылет)* |
|---|---|---|---|
| | 1 | Владивосток | пн, 9:40 |
| | | | ср, 10:20 |
| | 632 | Уфа | пн, 6:00 |
| | | | вт, 6:00 |

В этом случае легко реализовать запросы типа «Выдать все рейсы до Уфы», в отличие от запроса «Выдать все рейсы, вылетающие утром». С точки зрения второй задачи отношение не находится в 1НФ. Преобразование очевидно: отношение заменяется другим со схемой

*рейс (Номер, Пункт назначения, День, Время).*

Конец примера

### *2 нормальная форма (2НФ)*

Широко распространённая ошибка при проектировании баз данных – попытка объявить в качестве первичного ключа суперключ: дескать, лишний атрибут в ключе не повредит. Такая практика на деле приводит к значительным неприятностям.

***Определение.*** *Функциональная зависимость* $X=(A_1,A_2,...,A_k) \rightarrow B$ *полная, если* $B$ *зависит от всех* $A_i$ *из* $X$*. Если существует* $X' \rightarrow B$*, где* $X'$ *– собственное подмножество* $X$*, функциональная зависимость неполная.*

***Определение.*** *Отношение находится во 2НФ, если оно находится в 1НФ и каждый непервичный атрибут функционально полно зависит от ключа.*

Определение запрещает в качестве ключа использовать суперключ, если отношение находится во 2НФ.

Пример

Задано отношение со схемой *поставки (Поставщик, Товар, Цена)*, для которого определены следующие ограничения:

- Товар могут поставлять разные поставщики.
- Цена одинаковых товаров одинакова.
- Поставщик может поставлять разные товары.

Эти ограничения определяют следующие функциональные зависимости:

$$Поставщик, Товар \rightarrow Цена$$
$$Товар \rightarrow Цена$$

Здесь налицо неполная функциональная зависимость цены от ключа.
**Аномалия включения**: новый товар не включается в БД без поставки.
**Аномалия удаления**: поставки прекращаются – удаляются сведения о товаре.
**Аномалия обновления**: изменение цены влечет полный пересмотр.
Преобразование:

*Поставки (поставщик, товар)*
*Цена товара (товар, цена)*

Конец примера

### *3 нормальная форма (3НФ)*

Соответствие отношения 2НФ не гарантирует от других аномалий. Неприятные явления могут обнаружиться, если есть функциональная зависимость атрибута от непервичных.

***Определение.*** *Атрибут A транзитивно зависит от X, если существует Y такой, что выполняется условие* $X \rightarrow Y \;\&\; \neg(Y \rightarrow X) \;\&\; Y \rightarrow A \;\&\; X \rightarrow A, A \notin XY$.

Условие $\neg(Y \rightarrow X)$ означает, что Y – множество заведомо не первичных атрибутов.

***Определение.*** *Отношение находится в 3НФ, если оно находится в 1НФ и в нем отсутствует транзитивная зависимость атрибутов от первичных атрибутов.*

Пример

Решается задача, связанная с определением складов для отделений больницы. Задача ставится руководством отделения, для которого важно, чтобы за его отделением был закреплен склад определенного объема, причем только один. Тогда для отношения со схемой *хранение (Отделение, Склад, Объем)* существует единственная функциональная зависимость:

$$Отделение \rightarrow Склад$$

Через некоторое время выяснилось, что нужно следить и за состоянием складов безотносительно их принадлежности к отделению. Это порождает вторую функциональную зависимость:

$$Склад \rightarrow Объем$$

Таким образом, отношение оказалось не в 3НФ (существует транзитивная зависимость объема от отделения через склад: *Отделение* → *Склад* & ¬*Склад* → *Отделение* & *Склад* → *Объем* & *Отделение* → *Объем*).

**Аномалия включения**: нет отделения, получающего товар с этого склада – нет сведений об объеме.

**Аномалия удаления**: отделение перестает получать товар – нет данных о складе.

**Аномалия обновления**: изменение объема склада влечет полный пересмотр.

Преобразование:

*хранение (Отделение, Склад)*
*объем склада (Склад, Объем)*

Конец примера

### *Нормальная форма Бойса-Кодда (НФБК)*

Некоторые исследователи считают НФБК частным случаем 3НФ, но большинство, всё-таки, выделяют её как самостоятельную. Существует несколько эквивалентных определений НФБК, далее приводится одно из них.

***Определение.*** *Отношение находится в НФБК, если оно находится в 1НФ и в нем отсутствует зависимость первичных атрибутов от непервичных.*

Пример

В реализации программных проектов принимают участие программисты, которые разрабатывают отдельные модули каждого проекта. Для любого проекта каждый модуль разрабатывается одним программистом, каждый программист работает только в одном проекте. Необходимо создать базу данных для контроля выполнения проектов, в которой фиксируются незавершённые работы. Очевидно, что существует функциональная зависимость

*Проект, Модуль* → *Программист*

Естественная схема отношения такова:

*участие(Проект, Модуль, Программист)*

Но из условия единственности проекта, в котором участвует программист, следует и вторая функциональная зависимость:

*Программист* → *Проект*

Отношение находится в 3НФ, но у него есть аномалии модификации.

**Аномалия включения**: программист не попадает в БД до участия в проекте.

**Аномалия удаления**: последний модуль, который разрабатывал программист, закончен – программист исчез.

**Аномалия обновления**: увольняется программист и заменяется новым – нужен просмотр всего отношения.

**Преобразование:**

*состав_проекта (Проект, Модуль)*
*разработчик (Программист, Проект, Модуль)*

Конец примера

### *4 нормальная форма (4НФ)*

Другая распространённая ошибка при проектировании баз данных, кроме использования суперключа – наличие в одном отношении атрибутов, связанных отношением «один ко многим» или, что ещё хуже – «многие ко многим». Изначально эта связь может быть не очевидна, но по мере уточнения семантики отношения проявляется.

***Определение.*** *A многозначно определяет B в R (или B многозначно зависит от A), если каждому значению A соответствует множество (возможно, пустое) значений B, не зависимых от других атрибутов из R. Обозначение:* $A \rightarrow\rightarrow B$.

Пример

Задано отношение *преподаватель(Ид-преп, Дети, Курсы, Должность)*, которое связывает уникальный идентификатор (код) преподавателя с его семейными обстоятельствами (наличием детей) и служебным положением (читаемые курсы). Будем считать, что атрибуты *Ид-преп* и *Дети* находятся в отношении 1:M, а *Ид-преп* и *Курсы* – в отношении M:N. Здесь наличие детей и читаемые курсы – независимые атрибуты, то есть присутствуют многозначные зависимости *Ид-преп* $\rightarrow\rightarrow$ *Дети* и *Ид-преп* $\rightarrow\rightarrow$ *Курсы*.
Конец примера

***Определение.*** *Отношение находится в 4НФ, если оно находится в 1НФ и в нем отсутствует нефункциональные многозначные зависимости. Другое определение – для любой нетривиальной зависимости* $X \rightarrow\rightarrow Y$ *множество атрибутов* $X$ *содержит ключ).*

Пример
Зависимость между преподавателем, детьми и курсами из предыдущего примера приводит к тому, что при появлении нового ребенка приходится добавлять столько кортежей, сколько курсов читает этот преподаватель, а при добавлении курса следует добавить столько кортежей, сколько у преподавателя детей.

Преобразование:

*R1(Ид-преп, Дети)*
*R2(Ид-преп, Курсы)*
*R3(Ид-преп, Должность)*

Конец примера

### *5 нормальная форма (5НФ) – проекция/соединение*

***Определение.*** *Отношение находится в 5НФ, если оно находится в 1НФ и любая зависимость по соединению определяется возможными ключами отношения.*

Зависимость по соединению говорит о том, что отношение может быть восстановлено без потерь соединением некоторых его проекций. То есть, если $R$ – схема отношения $r(R)$, $R_1, R_2, \ldots, R_k \subseteq R$, $R_1 \cup R_2 \cup \ldots \cup R_k = R$, то $\pi_{R1}(r) \parallel \pi_{R2}(r) \parallel \ldots \parallel \pi_{Rk}(r) = R$.

В данном случае отношение не разбивается на какие-то другие. Оно просто может быть разбито на более удобные отношения и восстановлено обратно, то есть, его проекции состоят из полностью соединимых кортежей. Пример – факторизация отношения при работе с распределёнными базами данных. В тот момент, когда разделённое оператором фактора отношение будет соединено, мы должны быть уверены в его корректности: потеря данных или получение неверных их комбинаций недопустимы.

Исследование 5НФ достаточно сложно, в работе [13] говорится об отсутствии ясной её интерпретации и сомнительной практической применимости. Подробнее про 5НФ можно посмотреть в [16, 17].

Пример

Рассмотрим отношение *R1*. Оно находится в 4НФ: нет многозначных зависимостей, посмотрим, находится ли оно в 5НФ. Если мы обнаружим не полностью соединимые

проекции, значит, оно не в 5НФ. Рассмотрим проекции *R2(A,B)* и *R3(B, C)*. Для них условие *R2 || R3 = R1* выполняется. Но в соединении проекций *R4(A,C)* и *R3* уже появляются лишние кортежи, например, <a1, b2, c1>. Значит, исходное отношение не находится в 5НФ.

***R1*** *(A, B, C)*

| | | |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a2 | b1 | c2 |
| a3 | b1 | c1 |
| a3 | b1 | c2 |
| a3 | b2 | c1 |
| a3 | b2 | c2 |

***R2*** *(A, B)*

| | |
|---|---|
| a1 | b1 |
| a2 | b1 |
| a3 | b1 |
| a3 | b2 |

***R3*** *(B, C)*

| | |
|---|---|
| b1 | c1 |
| b1 | c2 |
| b2 | c1 |
| b2 | c2 |

Конец примера

# Лекция 12. Проектирование данных

### *Процессы проектирования*

Проектирование программных систем складывается, в основном, из проектирования процессов, данных, событий, интерфейсов и выходных документов. В силу специфики курса мы рассматриваем лишь проектирование данных – процесс разработки структуры базы данных в соответствии с требованиями заказчиков. В ходе разработки проекта нужно ответить на следующие вопросы:

- что представляют собой требования заказчиков и в какой форме они выражены;
- как они преобразуются в структуру базы данных;
- как часто и каким образом структура базы данных должна перестраиваться.

В настоящее время рассматриваются три уровня абстракции для определения структуры данных: концептуальный (точка зрения заказчика), логический (точка зрения разработчика) и физический (точка зрения администратора БД). В соответствии с этим рассматриваются три уровня модели и три шага проектирования. Некоторые источники (например, [19]) утверждают, что ни физической модели, ни шага физического проектирования на самом деле нет. Обсуждение этого взгляда проведем, когда уточним содержание всех трех видов абстракции.

Концептуальный уровень – наиболее общее представление об информационном содержании предметной области. Представляется в виде концептуальной модели (КМ), которая часто называется концептуальной схемой или информационной структурой. КМ обладает высокой степенью стабильности, она проблемно-ориентирована и не зависит от конкретной СУБД, операционной системы и аппаратного обеспечения. Ее поведение должно быть полностью предсказуемо.

Концептуальное представление оперирует основными элементарными данными предметной области, называемыми *сущностями*. Сущности описываются *атрибутами*. Данные могут находиться в некотором отношении друг с другом: образовывать ассоциации. Эти ассоциации называются *связями*. Концептуальная модель должна поддерживать согласованность связей в пределах уровня детализации.

Обычно для концептуального представления используется модель *«Сущность-Связь» (ER-модель)*, введенная Ченом [20], которая графически выражается *ER*-диаграммами. Существуют различные модификации представления (нотации) диаграмм. Ранее уже приводились сведения о *ER*-модели. Добавим, что, согласно предложению Чена, не только сущности, но и связи могут иметь атрибуты, выражающие их свойства. Представление модели внешне напоминает структуру базы данных и служит для отображения на логическую модель.

Логический уровень представления оперирует такими понятиями, как *запись*, *компоненты* записи, *связи* между записями. Соответствующая ему модель называется *логической* (ЛМ), она представляет собой отображение концептуальной модели в среду конкретной СУБД. Иногда [19] рассматривают не конкретную СУБД, а только ее класс (модель) – иерархическую, сетевую или реляционную. Особенности этих моделей рассматривались ранее.

Физический уровень демонстрирует физическое хранение данных. На этом уровне используются такие понятия, как *физические блоки*, *файлы*, *хранимые записи*, *указатели*. Взаимосвязи между хранимыми записями, возникающие в процессе их группировки, а также индексные структуры тоже рассматриваются на уровне физической модели. С точки зрения чистой базы данных можно абстрагироваться от физической модели представления данных, но знать ее полезно для достижения более высокой производительности системы. Если есть возможность влиять на физическую модель и есть представление о способах оптимизации работы в ее рамках, следует этим

воспользоваться, особенно для распределенных данных. Правда, реально почти никогда этого не делают.

Есть и другая классификация уровней представления данных. Согласно стандарту *ANSI / SPAC,* изложенному, в частности, в [18], архитектура БД представлена трехуровневой моделью с *внешним*, *концептуальным* и *внутренним* уровнями. В отличие от предыдущей модели, это не модель проектирования, а модель оперирования данными.

Внешний уровень – описание на языке пользователя структуры данных, вида и формы их представления, а также описание операций манипулирования данными. Считается, что для описания предметной области используется несколько внешних моделей. Данный уровень содержит черты как КМ, так и ЛМ, описанных ранее.

Концептуальный уровень – наиболее общее представление об информационном содержании предметной области. Определение совпадает с приведенным ранее.

Внутренний уровень – организованная совокупность структурированных данных, отображение концептуальной модели в конкретную среду хранения. Легко видеть, что это понятие объединяет ранее определенные логическую и физическую модели.

Если рассмотреть эти два подхода к описанию представления данных, можно прийти к заключению, что первый более прагматичен. В нем предметная область рассматривается как единое целое, а не как совокупность проектных требований, называемых внешними моделями. Реально проектные требования редко можно назвать полноценной моделью, так как любая модель должна давать на каком-то уровне адекватное представление о предметной области. Полученные же требования зачастую выступают как совокупность представлений о ней разных групп пользователей. Такая ситуация возникает в тех случаях, когда аналитик считает, что пользователи формулируют свои знания как локальные модели, совокупность которых и должна составлять требуемую модель. Ошибочность этого утверждения хорошо иллюстрируется известной индийской сказкой об исследовании слона пятью слепцами, в ходе которого они предложили свои локальные модели слона (исследователь хобота считал, что слон похож на канат, хвоста – на метелку, бока – на стену, уха – на лист, ноги – на колонну). Реально пользователь часто не в состоянии построить даже локальную информационную модель, а про глобальные связи между ними и говорить не приходится. Есть и другое соображение. Популярное в настоящее время направление проектирования – перепроектирование производственных процессов (реинжиниринг бизнес-процессов – *BPR*) – отрицает такой подход в силу того, что он консервирует существующую технологию и не дает выделить цель производства. А раз невозможно выделить общую цель производства, результат данного этапа исследования нельзя считать моделью. Но можно, введя понятие типа пользователя (эксперта), рассматривать соответствующую внешнюю модель как точку зрения этого эксперта на предметную область. В этом случае концептуальная модель представляется как единое целое, дополненное совокупностью точек зрения экспертов. Любопытно мнение о возможности адекватного представления модели, приведенное в [13], где утверждается, что оно невозможно в принципе.

Теперь рассмотрим два основных уровня проектирования: концептуальное и логическое – с точки зрения первого подхода.

### *Концептуальное проектирование*

На этапе концептуального проектирования определяются информационные потребности и локальные представления предметной области. Выявляется роль, назначение, взаимосвязь данных, проводится их глобальная спецификация. Результат этапа – описание объектов данных и их взаимосвязи без указания способа их физической организации. Структура данных представляется концептуальной схемой, содержащей набор сущностей, связей и атрибутов. Различаются две важных стадии концептуального проектирования: анализ данных и организация их хранения.

Содержание первой стадии – сбор полной и точной информации о данных предметной области. Заметим, что речь идет о первоначальном сборе информации, в процессе проектирования, как правило, выясняются дополнительные обстоятельства. Нередко для проведения данной работы прибегают к одному из двух методов (или к обоим): анкетированию и работе с экспертами. В первом случае пользователю предлагается анкета, в которой он должен дать свое представление о данных. Во втором случае среди пользователей выбирается группа экспертов, которые все и излагают. Оба этих метода страдают большими недостатками. Анкетирование никогда не гарантирует полноту и достоверность информации: к этому процессу пользователи относятся как к досадной помехе в основной деятельности, поэтому стараются не столько дать точные сведения, сколько поскорее «отбиться». Работа с экспертами несколько лучше, но требует затрат квалифицированного труда. Кроме того, эксперты, будучи специалистами в своей области, не всегда способны достаточно внятно и полно изложить суть проблемы в целом. Тем не менее, не стоит пренебрегать этими методами, но за основу лучше взять метод, основанный на личном участие разработчиков в исследовании. Иногда он называется этнографическим. Суть его сводится к тому, что исследователь (аналитик) наблюдает производственный процесс непосредственно на рабочем месте пользователя. Здесь, в частности, следует изучить набор выходных (в первую очередь) и входных документов и проследить их движение и модификацию на каждом этапе технологического процесса. Во время такого исследования выявляются особенности принятого документооборота, его недостатки и делаются попытки, как это предлагает *BPR*, оптимизировать его с точки зрения машинной обработки. Заметим, что нужно обращать внимание не только на официально утверждённые формы, но и на внутреннюю документацию, в том числе и на ручные записи.

Вторая стадия сводится к разработке графического представления полученной информации в виде схемы, которая включает, в частности, результирующие данные с формирующими их процессами и исходные со ссылкой на использующие процессы. На этом же этапе уточняется степень важности данных, выявляются и фиксируются связи между ними. К данной работе, наряду с проектировщиком, полезно привлекать администратора баз данных и представителей пользователя.

### *Логическое проектирование*

Роль логическое проектирования – отображение КМ в выбранную модель данных. На этом этапе необходимо определить отношения и атрибуты, выделить ключи. На ряд атрибутов могут быть наложены ограничения, которые выражаются в функциональных зависимостях между ними. Если выбрана реляционная модель данных, в процессе проектирования следует так определять отношения, чтобы атрибуты в каждом из них функционально полно зависели от ключей и не было транзитивной зависимости атрибутов в отношении. В результате должна сформироваться логическая схема БД, находящаяся в третьей нормальной форме. Эта схема, разумеется, не окончательная, в процессе проектирования она может неоднократно корректироваться, в результате чего нормализованность может нарушиться. В этом случае добавляется специальный этап нормализации схемы. Основное назначение этапа нормализации – получение схемы, эквивалентной данной, но не обладающей некоторыми отрицательными свойствами, связанными с функциональными зависимостями.

#### Обеспечение целостности и достоверности данных

При обсуждении реляционной модели говорилось, что модель содержит правила целостности, которые определяют множество непротиворечивых состояний базы данных и множество изменений ее состояний. Эти правила обеспечиваются контролем ограничения целостности.

Ограничения целостности служат для защиты данных от некорректных изменений. Различают статические ограничения, отражающие множество корректных состояний БД, и динамические, определяющие правильные переходы из состояния в состояние. Соответственно, ограничения целостности обеспечиваются вызовом при модификации данных программ статического или динамического арбитража.

Статический арбитраж контролирует выполнение основных правил, характерных для реляционных баз данных:

- уникальность кортежей во всех отношениях;
- уникальность и непустоту первичного ключа;
- уникальность возможных ключей;
- зависимость между атрибутами:
- формат атрибутов;
- ограничение значений атрибутов.

Динамический арбитраж обеспечивает корректность базы данных при выполнении операций добавления, удаления и изменения данных. Для этого необходимо описание соответствующих условий. Каждое из них может затрагивать одну таблицу или более. Кроме того, динамический арбитраж обеспечивает корректность в особых состояниях базы данных, которые описаны условными ограничениями.

Процедуры контроля исходных данных по заданным ограничениям могут быть как внешними по отношению к базам данных, так и встроенными, если СУБД допускает возможность триггеров и хранимых процедур.

Триггер – это процедура, связанная с одной таблицей базы данных. Роль триггера – обеспечение контроля добавления, удаления или изменения записи. Выполнение контроля может потребовать обращения к другим таблицам базы данных, тогда в работу включаются триггеры этих таблиц, возникает каскад триггеров. Для обеспечения корректности подобной деятельности нередко используют механизм транзакций.

Хранимые процедуры определяются для базы данных в целом. Обычно они служат для реализации однотипной деятельности с базой данных и обеспечивают уменьшение программного кода и сокращение транзакции.

### Понятие транзакции

Классическая транзакция – последовательность операций изменения базы данных или выборки, которая воспринимается СУБД как атомарное действие. Дисциплина транзакций включает различные функции для поддержки приложений, основанных на коммуникациях. Системы обработки транзакций охватывают базы данных, сети, операционные системы. Понятие обработки транзакций применимо к любой компьютерной среде, но чаще говорят о средах с крупномасштабными центрами обработки информации, такими, как резервирование билетов, крупные больницы, торговые центры и т.п. Транзакция объединяет несколько действий, которые рассматриваются как единое целое. Это значит, что должно быть обозначено начало транзакции и её завершение, которое может быть как успешным, так и неуспешным. Во втором случае транзакция обязана вернуть базу данных в исходное состояние – откатить транзакцию.

Принято считать, что любая транзакция основана на наборе принципов, называемом *ACID*:

- атомарность (*Atomicity*) – выполнение действий по принципу «все или ничего»;
- целостность (*Consistency*) – корректные преобразования состояний системы;
- изолированность (*Isolation*) – данные, для которых нарушения целостности возникли в процессе выполнения транзакции, невидимы до фиксации;
- долговременное сохранение (*Durability*) – результаты зафиксированной транзакции сохраняются даже при аппаратных или программных сбоях.

Эти принципы поддерживают многие модели транзакций – от простейших (плоских) до сложных (вложенных и многозвенных).

Плоские транзакции обладают единственным уровнем управления для произвольного числа элементарных действий. Транзакция откатывается, если хотя бы один компонент не завершается. Для распределенной транзакции это не приемлемо: вероятность отказа велика, транзакцию приходится повторять заново. Поэтому используется модификация, основанная на контрольных точках, чтобы при откате повторять работу с текущей точки.

В модель многозвенных транзакций каждый этап вычислений фиксируется как субтранзакция, она же и откатывается при неудаче.

Вложенная транзакция представляет собой иерархию транзакций, управляемую транзакцией верхнего уровня. Для нее существуют три правила управления:

- правило фиксации – фиксация делает результаты видимыми только для родительского уровня;
- правило отката – откат корневой транзакции ведет к откату всей иерархии;
- правило видимости – родительская транзакция видит результаты дочерней после их фиксации, а дочерняя результаты родительской – всегда, соседние одноуровневые результаты друг друга не видят.

Последнее правило позволяет выполнять субтранзакции параллельно. Вложенные транзакции представляют собой гибкие средства управления субтранзакциями, но они сложны в реализации и не всегда необходимы. В ряде случаев достаточно многозвенной модели.

#### Контроль полномочий

Контроль полномочий (разграничение прав доступа) служит защитой от несанкционированного доступа к данным. Для него обычно используется система паролей, отражающая статусы пользователей. Обычно рассматриваются право на вход в систему, которое регламентирует определённые действия в её среде, право на доступ к данным для чтения и право на доступ к данным на изменение. В последнем случае рассматривается возможность манипуляции с данными общего пользования (справочно-нормативная информация), изменения (удаления) данных и их добавления. При обмене данными через каналы связи также производится идентификация и аутентификация источника (приёмника) данных. Контроль полномочий позволяет в определённой степени гарантировать защиту данных от несанкционированного доступа и изменения.

### *Средства создания модели*

Создание *ER*-модели обычно сопровождается ее графическим представлением. Различные нотации *ER*-диаграмм поддерживается специальными средствами проектирования программных систем (*CASE*-средствами). В некоторых случаях подобные средства включаются в инструментальную среду создания баз данных (*FoxPro, Access, Oracle* и т.п.), иногда они существуют как отдельный продукт. Последний вариант интересен тем, что нет привязки к конкретной СУБД, то есть можно определять концептуальную модель, которая может быть отображена в логическую после того, как будет выбрана СУБД. Рассмотрим один из наиболее популярных средств такого рода – *ERwin*.

*CASE*-средство *ERwin* уже достаточно долго присутствует на рынке инструментальных средств, ориентированных на базы данных. Существует несколько версий этого продукта. *ERwin* позволяет создавать модели двух уровней: концептуального и логического. Правда называются они довольно своеобразно: концептуальный называется логическим, а логический – физическим. Представление диаграмм соответствует стандарту *IDEF1X*. Основные элементы модели следующие: сущности, атрибуты сущностей, домены, связи (четыре вида), индексы. Современные версии *ERwin* не допускают

атрибуты связей. Среди атрибутов выделяются первичные ключи. Кроме первичных, можно отметить возможные (альтернативные) ключи, а также ключи поиска. Внешние ключи формируются автоматически при определении связей, причем, в качестве родительского ключа выбирается первичный.

Сильная сторона *ERwin* заключается в относительной простоте и удобстве работы с продуктом, что немаловажно в случае сжатых сроков разработки (а это случается постоянно). Разумеется, поддерживаются различные уровни документирования модели: описания сущностей, атрибутов, связей, диаграммы, модели в целом и т.п. Есть возможность получить пакет документации о модели с использованием внутреннего генератора отчетов.

После выбора СУБД появляется возможность привязки к типам данных, принятых в этой СУБД. В терминах *ERwin*, эта работа производится в физической модели. Можно уточнить типы данных, индексы, а также задать хранимые процедуры, обеспечивающие корректность базы данных. Заметим, что в этой модели несколько меняется терминология: сущности называются таблицами, а атрибуты – столбцами. И, наконец, завершающая работа – автоматическая генерация пустой базы данных по разработанной модели. Базу данных следует проверить, особенно индексы, заполнить тестовыми данными и протестировать. Обычно для создания работоспособной базы данных требуется несколько итераций, в ходе которых изменять следует исключительно модель, а не созданную базу данных.

## *Работа со средством создания моделей данных ERwin*

Данный раздел не входит в базовый курс, он может быть полезен для выполнения лабораторных работ. Тем более, что мне пришлось потратить целую лекцию на изложение этого материала. Тема и изложение восходит, в основном, к [15].

### Сущности, атрибуты, ключи

Логическая модель *ERwin* имеет несколько уровней отображения диаграммы: уровень сущностей, уровень атрибутов, уровень определений, уровень первичных ключей и уровень пиктограмм. Переключиться между первыми тремя уровнями можно с использованием кнопок панели инструментов. Переключиться на другие уровни отображения можно при помощи контекстного меню.

Различают три уровня логической модели, отличающихся по глубине представления информации о данных:

- диаграмма сущность-связь (*Entity Relationship Diagram, ERD*);
- модель данных, основанная на ключах (*Key Based model, KB*);
- полная атрибутивная модель (*Fully Attributed model, FA*).

Диаграмма сущность-связь представляет собой модель данных верхнего уровня. Она включает сущности и связи, отражающие основные понятия предметной области. В нее включаются сущности и связи между ними, которые удовлетворяют основным требованиям, предъявляемым к ИС. Диаграмма сущность-связь может включать связи «многие ко многим» и не включать описание ключей.

Модель данных, основанная на ключах – более подробное представление данных. Она включает описание всех сущностей и первичных ключей, которые соответствуют предметной области.

Полная атрибутивная модель – наиболее детальное представление структуры данных: представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи.

Построение модели данных предполагает определение сущностей и атрибутов: необходимо определить, какая информация будет храниться в конкретной сущности или атрибуте. Сущности должны иметь наименование с четким смысловым значением,

именоваться существительным в единственном числе, не носить «технических» наименований. Именование сущности в единственном числе облегчает чтение модели.

Каждая сущность должна быть полностью определена с помощью текстового описания в закладке *Definition*. Закладки *Note, Note2, Note3, UDP* (*User Defined Properties* – свойства, определенные пользователем) служат для внесения дополнительных комментариев и определений к сущности.

Каждый атрибут хранит информацию об определенном свойстве сущности. Для описания атрибутов следует выбрать в контекстном меню пункт *Attribute Editor*. В появившемся диалоге кнопка *New* даёт возможность указать имя атрибута, имя соответствующей ему в физической модели колонки и домен. Домен атрибута используется при определении типа колонки на уровне физической модели. Закладка *Definition* позволяет записывать определения отдельных атрибутов. Закладка *UDP* служит для задания значений свойств, определяемых пользователем.

Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Согласно синтаксису *IDEF1X* имя атрибута должно быть уникально. Каждый атрибут должен быть определен (закладка *Definition*). Иногда определение атрибута можно дать через описание домена. Например, оценка студента – это целое число в диапазоне от 2 до 5.

В *ERwin* домен может определяться и использоваться как в логической, так и в физической модели. На логическом уровне домены можно описать без конкретных физических свойств, на физическом уровне они автоматически получают специфические свойства, которые можно изменить вручную. Так, домен «Возраст» может иметь на логическом уровне тип *Number*, на физическом уровне колонкам домена будет присвоен тип *INTEGER*.

Для создания нового домена в диалоге *Domain Dictionary Editor* следует

- по кнопке *New* вызвать диалог *New Domain*;
- выбрать родительский домен из списка *Domain Parent*. Новый домен наследует все свойства родительского домена. Эти свойства в дальнейшем можно переопределить;
- набрать имя домена в поле *Logical Name*. Можно указать имя домена на физическом уровне в поле *Physical Name*. По умолчанию оно принимает значение логического имени.

Домен может быть описан в закладке *Definition*, снабжен комментарием в закладке *Note* или свойством, определенным пользователем в закладке *UDP*. На физическом уровне диалог *Domain Dictionary Editor* позволяет редактировать физические свойства домена.

На физическом уровне сущностям соответствуют таблицы, а атрибутам – столбцы. Выбрав в контекстном меню пункты *Table Editor* или *Column Editor*, можно вызвать редакторы для задания свойств таблиц и столбцов.

*ERwin* автоматически создает имена таблиц и столбцов на основе имен соответствующих сущностей и атрибутов, учитывая максимальную длину имени и другие синтаксические ограничения, накладываемые СУБД. При генерации имени таблицы или столбца по умолчанию все пробелы автоматически преобразуются в символы подчеркивания, а длина имени обрезается до максимальной длины, допустимой для выбранной СУБД. Все изменения, сделанные в *Table Editor* или *Column Editor*, не отражаются на именах сущностей и атрибутов.

Рассмотрим множество типов ключей в модели *IDEF1X*. Общее для них – обеспечение доступа к данным. Но роли их в модели различны.

Первичный ключ (*primary key*) – это атрибут или группа атрибутов, однозначно идентифицирующая экземпляр сущности. Атрибуты первичного ключа на диаграмме не требуют специального обозначения, они находятся в списке атрибутов выше гори-

зонтальной линии. Чтобы сделать новый атрибут первичным, при внесении его в диалоге *Attribute Editor* нужно включить флажок *Primary Key* в нижней части закладки *General*. Уже существующий атрибут вносится в состав первичного ключа либо на диаграмме, либо в том же диалоге установкой для него этого же флажка *Primary Key*.

Выбор первичного ключа может оказаться непростой задачей, решение которой может повлиять на эффективность будущей системы. В одной сущности могут оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются возможными или потенциальными ключами (*candidate key*). Каждая сущность должна иметь, по крайней мере, один возможный ключ. Если сущность имеет только один возможный ключ, он и становится первичным. Некоторые сущности могут иметь более одного возможного ключа. Тогда один из них становится первичным, а остальные - альтернативными ключами. Альтернативный ключ (*Alternate Key*) – это возможный ключ, не ставший первичным. *ERwin* позволяет выделить атрибуты альтернативных ключей, и при генерации схемы БД по этим атрибутам будет генерироваться уникальный индекс.

Часто бывает необходимо обеспечить доступ к нескольким экземплярам сущности, объединенным каким-либо одним признаком. Для повышения производительности в этом случае используются неуникальные индексы. *ERwin* позволяет на уровне логической модели назначить атрибуты, которые будут участвовать в неуникальных индексах. Эти атрибуты называются инверсионными входами (*Inversion Entries*). Это атрибут или группа атрибутов, которые не определяют экземпляр сущности, но используются для обращения к экземплярам сущности. *ERwin* генерирует неуникальный индекс для каждого инверсионного входа.

Имена индексов формируются автоматически, но их можно изменить вручную.

Внешние ключи (*Foreign Key*) создаются автоматически, когда связь соединяет сущности: связь образует ссылку на атрибуты первичного ключа в дочерней сущности, и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибуты внешнего ключа обозначаются символом (*FK*) после своего имени.

### Связи

Связь – это логическое соотношение между сущностями. В ссылочных моделях баз данных связь реализуется непосредственной ссылкой экземпляра одной сущности на экземпляр другой. В реляционной модели реализуется ассоциативная связь, согласно которой по состоянию экземпляра одной сущности (кортежа) можно догадаться, какие экземпляры другой сущности с ним связаны. Тем не менее, на уровне модели «сущность-связь» эта разница никак не сказывается, на что специально указывал Чен.

Каждая связь должна именоваться глаголом. Имя связи выражает некоторое ограничение и облегчает чтение диаграммы, например:

- Каждый ***Студент*** <*входит в*> ***Группу***;
- Каждая ***Группа*** <*содержит*> ***Студента***.

Связь показывает, какие именно студенты входят в группу и какая именно группа содержит студента.

В модели «сущность-связь» определены независимые и зависимые сущности, причём, степень зависимости отображается на диаграмме. В стандарте *IDEF1X* на диаграмме выделяется только сильная зависимость, которая определяется идентифицирующей связью. Слабо зависимая сущность по виду не отличаются от независимой, но её можно определить по наличию неидентифицирующей связи. Связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. При установлении связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав дочерней сущности. Эта операция называет-

ся миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ (*FK*).

Когда проводится идентифицирующая связь, *ERwin* автоматически преобразует дочернюю сущность в зависимую, которая изображается прямоугольником со скругленными углами. Экземпляр зависимой сущности определяется только через отношение к родительской сущности. При установлении идентифицирующей связи миграцией атрибутов производится в состав первичного ключа дочерней сущности.

Если атрибут мигрирует в качестве внешнего ключа более чем на один уровень, то на первом уровне отображается полное имя внешнего ключа (имя роли + базовое имя атрибута), на втором и более – только имя роли (см. далее).

При установлении неидентифицирующей связи дочерняя сущность выглядит как независимая, а атрибуты первичного ключа родительской сущности мигрируют в состав непервичных атрибутов дочерней сущности.

Идентифицирующая связь показывается на диаграмме сплошной линией с жирной точкой на дочернем конце связи, неидентифицирующая – пунктирной.

Для редактирования свойств связи следует из контекстного меню выбрать пункт *Relationship Editor*. В закладке *General* появившегося диалога можно задать мощность, имя и тип связи.

Мощность связи (*Cardinality*) служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней. Различают четыре типа мощности:

- общий случай, когда одному экземпляру родительской сущности соответствуют 0, 1 или более экземпляров дочерней сущности, не помечается;
- символом ***P*** помечается случай, когда одному экземпляру родительской сущности соответствуют один или более экземпляров дочерней сущности (исключено нулевое значение);
- символом ***Z*** помечается случай, когда одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения);
- целым числом помечается случай точного соответствия, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

По умолчанию символ, обозначающий мощность связи, не показывается на диаграмме. Для его отображения следует в контекстном меню выбрать *Display Options/Relationship* и затем включить вариант *Cardinality*. Это же касается и имени связи: для его отображения следует в контекстном меню выбрать пункт *Display Options/Relationship* и затем включить вариант *Verb Phrase*.

Имя связи (*Verb Phrase*) – глагольная фраза, характеризующая отношение между родительской сущностью и дочерней.

В закладке *Definition* можно дать более полное определение связи для того, чтобы в дальнейшем иметь возможность на него ссылаться.

В закладке *Rolename/RI Actions* можно задать имя роли и правила ссылочной целостности. Имя роли – это синоним атрибута внешнего ключа, который показывает, какую роль играет атрибут в дочерней сущности. Обязательно имя роли используется в том случае, когда два или более атрибутов одной сущности имеют одинаковую область значений, но разный смысл. Другой случай применения имени роли – рекурсивная связь, когда одна и та же сущность выступает одновременно как родительская и дочерняя. При задании рекурсивной связи атрибут должен мигрировать в качестве внешнего ключа в состав непервичных атрибутов той же сущности. Он не может появиться дважды в одной сущности под одним именем, поэтому обязательно должен получить имя роли.

Связь «многие ко многим» возможна только на уровне логической модели данных. Такая связь обозначается сплошной линией с двумя точками на концах. Связь «многие ко многим» должна именоваться двумя фразами в обе стороны. Это облегчает чтение диаграммы.

При переходе к физическому уровню *ERwin* автоматически преобразует связь «многие ко многим», добавляя новую таблицу и устанавливая две новые связи «один ко многим» от старых таблиц к новой. При этом имя новой таблице формируется автоматически как «***Имя1_Имя2***». Но автоматического решения проблемы связи «многие ко многим» не всегда достаточно. В ряде случаев дополнительная таблица требует новых атрибутов либо в состав ключа, либо во множество непервичных. По сути, это ни что иное, как реализация концепции атрибутов связи в модели Чена.

Рассмотрим типы зависимых сущностей.

- Характеристическая – зависимая сущность, которая связана только с одной родительской и хранит информацию о её характеристиках.
- Ассоциативная – связанная с несколькими родительскими сущностями. Содержит информацию о связях сущностей.
- Именующая – частный случай ассоциативной, не имеющей собственных атрибутов, только атрибуты родительских сущностей, мигрировавших в качестве внешнего ключа.
- Категориальная – дочерняя сущность в иерархии наследования.

Иерархия наследования (или иерархия категорий) представляет собой особый тип объединения сущностей, которые разделяют общие характеристики. Например, в вузе обучаются бюджетные и платные студенты. Из их общих свойств можно сформировать обобщенную сущность (родовой предок) ***Студент***, чтобы представить информацию, общую для всех студентов. Специфическая для каждого типа информация может быть расположена в категориальных сущностях (потомках) ***бюджет*** и ***платный***. Обычно иерархию наследования создают, когда несколько сущностей имеют общие по смыслу атрибуты, либо когда сущности имеют общие по смыслу связи. Для каждой категории можно указать дискриминатор – атрибут родового предка, который показывает, как отличить одну категориальную сущность от другой.

Иерархии категорий делятся на два типа – полные и неполные. В полной категории одному экземпляру родового предка обязательно соответствует экземпляр в каком-либо потомке. Если категория еще не выстроена полностью и в родовом предке могут существовать экземпляры, которые не имеют соответствующих экземпляров в потомках, то такая категория будет неполной.

### Индексы

В таблице базы данных строки обычно хранятся в том порядке, в котором их разместили. Многие реляционные СУБД имеют организацию памяти, при которой физически таблица может храниться фрагментарно в разных областях диска. Хотя такой способ хранения и позволяет быстро вводить новые данные, но для того, чтобы найти нужную строку, придется просмотреть всю таблицу. При её объеме в миллионы строк простой перебор ведет к катастрофическому падению производительности. Чтобы решить проблему поиска данных, СУБД использует особый объект, называемый индексом. Индекс указывает на строки, в которых хранится конкретное значение колонки. Поскольку значения (статьи) в индексе хранятся в определенном порядке, просматривать нужно гораздо меньший объем данных. Индекс создаётся для тех колонок, по которым часто производится поиск.

При генерации схемы физической БД *ERwin* автоматически создает отдельный индекс на основе первичного ключа каждой таблицы, а также на основе всех альтернативных ключей, внешних ключей и инверсионных входов, поскольку эти колонки наи-

более часто используются для поиска данных. Можно отказаться от генерации индексов по умолчанию, и для повышения производительности создать собственные индексы. На уровне логической модели индекс можно создать неявно, включая колонки в состав альтернативных ключей и инверсионных входов.

Изменяются характеристики существующего индекса или создаётся новый в редакторе *Index Editor*. В нём можно изменить имя индекса, изменить его определение так, чтобы он принимал уникальные или дублирующиеся значения, или изменить порядок сортировки данных.

Подробнее проблемы физического доступа к данным, в частности, индексного доступа, обсуждаются в Лекции 13.

### Генерация схемы базы данных

Процесс генерации физической схемы БД из модели данных в *ERwin* называется прямым проектированием (*Forward Engineering*). При генерации физической схемы *ERwin* включает необходимые возможности, доступные при определении таблиц в выбранной СУБД. Для генерации БД следует выбрать пункт меню *Tools/Forward Engineering*. Появляется соответствующий диалог. Закладка *Options* служит для задания вариантов генерации объектов БД. Для задания генерации какого-либо объекта следует выбрать его в левом списке закладки, после чего включить соответствующий вариант в правом списке. В закладке *Summary* отображаются все варианты, заданные в закладке *Options*.

Кнопка *Generate* запускает процесс генерации схемы. Возникает диалог связи с БД и начинает выполняться генерация. При этом возникает диалог *Generate Database Schema,* в котором можно выбрать дополнительные параметры генерации.

Результат работы – сгенерированная база данных или сообщения об ошибках. По ходу работы можно задать вариант сохранения текста программы создания базы данных на *SQL*. Может случиться, что некоторые действия, например, создание индексов, не смогли быть выполнены во время генерации. Тогда, пользуясь полученными тестами, можно сделать это позже.

### Отчёты

Для генерации отчетов в *ERwin* имеется специальный инструмент – *Report Browser*. Он позволяет выполнять предопределенные отчеты, сохранять результаты их выполнения, создавать собственные отчеты, печатать и экспортировать их в распространенные форматы. Каждый отчет может быть настроен индивидуально, данные в нем могут быть отсортированы и отфильтрованы.

Диалог *Report Browser* вызывается кнопкой в панели инструментов *ERwin*. В верхней левой части диалога расположено окно, отображающее дерево отчетов. Отчеты группируются в папки. Отчет может включать несколько результирующих наборов данных, каждое из которых генерируется при очередном выполнении отчета. По умолчанию *Report Browser* содержит предварительно определенные отчеты, позволяющие представить информацию об основных объектах модели данных. Для выполнения отчета достаточно дважды щелкнуть по нему в дереве отчетов или щелкнуть по соответствующей кнопке на панели инструментов. Результат выполнения отчета будет отображен в правом окне диалога *Report Browser*.

Для создания нового отчета следует выбрать пункт меню *File/New ERwin Report* или щелкнуть по кнопке на панели инструментов. Появляется диалог *ERwin Report Editor*. В поле *Name* следует внести имя отчета. Категория отчета (*Category*) указывает на тип объектов модели, по которым будет создаваться отчет (атрибуты, сущности, домены, связи и т.д.). Закладки *Definition* и *Note* служат соответственно для внесения определения и комментария к отчету. Закладка *Options* отображает информацию, которая будет включена в отчет. После щелчка по кнопке *OK* отчет будет добавлен в список

отчетов диалога *Report Browser*. Для выполнения отчета нужно либо дважды щелкнуть по его имени в списке, либо щелкнуть по кнопке в палитре инструментов.

Редактор *Edit ERwin Report* даёт возможность изменить любой существующий отчет.

Полученный после выполнения отчета набор данных можно отформатировать, распечатать, экспортировать или сохранить в виде представления. Для экспорта следует выбрать в контекстном меню пункт *Export result set*. Допустимые форматы экспорта:

- *CSV* – текстовый файл;
- *HTML*;
- *DDE* – экспорт в *MS Word* или *MS Excel*;
- *RPTwin* – экспорт в специализированный генератор отчетов.

# Лекция 13. Методы хранения данных и доступа к ним

Производительность программной системы во многом зависит от методов доступа к данным. Порой недели труда над оптимизацией алгоритма, дающей выигрыш по времени вдвое, пропадают из-за неудачного метода работы с данными, увеличивающего время на два порядка. В реальной жизни программист зачастую вынужден использовать конкретную СУБД, избранную для реализации проекта, поэтому возможности маневрировать методами доступа у него небольшие. Но если есть возможность выбора СУБД, знание используемых в ней алгоритмов работы с данными может быть полезным. Кроме того, владение методами хранения данных и доступа к ним позволяет для повышения эффективности время от времени прибегать к их собственной реализации.

Проблема эффективного доступа к данным достаточно сложна, для ее изучения требуется отдельный курс. Цель данной лекции – лишь ознакомление с нею на примере некоторых часто встречающихся методов. В нее включены последовательный, прямой, индексно-последовательный, индексно-произвольный методы, метод инвертированных списков, метод хеширования (перемешанные таблицы). Не рассматриваются методы, основанные на деревьях. Кроме того, не рассматриваются проблемы, связанные с физическим размещением данных в оперативной и внешней памяти.

Для оценки методов доступа и хранения используются понятия *эффективности доступа* и *эффективности хранения*.

***Определение.*** *Эффективность доступа – отношение числа логических обращений к числу физических при выборке элемента данных.*

***Определение.*** *Эффективность хранения – отношение числа информационных байтов к числу физических при хранении.*

Например, если на одно логическое обращение требуется два физических, то эффективность доступа 0,5. Если на 10 байт информации требуется одна двухбайтовая ссылка, эффективность хранения 10/12.

В этой лекции используется понятие ключа поиска. Под ним подразумевается любой набор значений атрибутов, который должен быть найден. Он не обязательно однозначно определяет строку, но в некоторых методах однозначность подразумевается. Далее в лекции словом «ключ» будет обозначаться именно ключ поиска.

## *Непосредственный доступ*

### Последовательный метод

В этом методе предполагается физическое расположение записей в логической последовательности. Для выборки записи необходимо просмотреть все предшествующие ей. Очевидно, что эффективность доступа линейно зависит от длины файла. Как правило, время доступа в этом случае недопустимо велико. С другой стороны, для этого метода характерна очень высокая эффективность хранения. Кроме того, алгоритм доступа к данным крайне прост. Следовательно, метод не может применяться там, где необходим быстрый доступ к данным большого объема. Но его можно использовать в тех случаях, когда по характеру задачи следует выбирать записи последовательно (например, полное копирование данных), а также при очень небольших объемах данных в силу простоты алгоритма доступа.

### Прямой метод

Для прямого метода необходимо взаимно однозначное соответствие между ключом и адресом записи. В этом случае некоторая адресная функция (возможно, тривиальная)

формирует адрес, по которому выбирается запись. Это наиболее эффективный метод по времени доступа, эффективность доступа всегда равна единице. Эффективность хранения зависит от плотности размещения ключей. Если это справочная (неизменяемая) таблица, ключи могут располагаться достаточно плотно. В общем случае этот метод довольно расточителен по памяти.

Прямой метод применяется в случаях, когда время – наиболее ценный ресурс, например, при организации таблиц операционной системы, а также в тех случаях, когда характер задачи предполагает плотное хранение данных с доступом по номеру, например, в задачах вычислительной математики. Но уже для работы с разреженными матрицами он неэффективен, для этого существуют специальные методы.

### *Индексные методы*

В основе индексных методов доступа лежит создание вспомогательной структуры – *индекса*, содержащего *ключи поиска* и ссылки на физические адреса данных. Термин «ключ поиска» не обязательно подразумевает его уникальность, это просто атрибут (комбинация атрибутов), который должен удовлетворять критерию поиска. Если ключ поиска уникален, его называют *первичным* ключом, в противном случае говорят о *вторичном* ключе. Впрочем, вторичный ключ может быть и уникальным, просто это не обязательно. В зависимости от вида ключа поиска различаются *первичные* и *вторичные* индексы.

Доступ к данным производится в два этапа. Вначале в индексе (индексном файле) находятся требуемые значения ключей, затем из основного файла по ссылке извлекается требуемая информация. Разумеется, ни эффективность доступа, ни эффективность хранения при использовании этих методов не могут достигать единицы, но производительность системы в целом может стать достаточно высокой. Для ее увеличения обычно требуют, чтобы индекс целиком размещался в оперативной памяти.

Индексы могут быть устроены по-разному. Если поиск и выборка производится по комбинации атрибутов (*индексному выражению*), соответствующий индекс называется *составным*. Индекс, построенный на иерархии ссылок, называется *многоуровневым*. Индекс, который содержит ссылки не на все записи, а на некоторый диапазон, называется *неплотным*. *Плотный* индекс содержит ссылки на все записи. Элемент индекса часто называют *статьей*.

Существует множество индексных методов доступа. Рассмотрим три из них: индексно-последовательный, индексно-произвольный и метод инвертированных списков.

#### Индексно-последовательный метод

В индексно-последовательном методе информационный файл размещается по блокам одинакового размера, начальная часть блока заполняется информационными записями, конечная часть остается свободной. Строится индекс, статья которого содержит указатель на блок, а в качестве ключа индекса выбирается значение ключа первой или последней (это предпочтительнее) записи соответствующего блока. Индексы группируются в индексный файл, который упорядочен по значению ключа. Таким образом, индекс ссылается на группу записей, которые расположены в логическом порядке, то есть в данном методе используется неплотный индекс.

Поиск производится следующим образом. В оперативную память загружается индекс, в нем выбирается ссылка на диапазон ключей, в котором предположительно находится искомая запись. Затем в память загружается нужный блок, в нем последовательным методом ищется нужный ключ.

Неплотность индекса дает возможность уменьшить количество его записей по сравнению с объемом базы кратно размеру блока. Но индекс все равно может стать слишком большим и не помещаться в память. Тогда есть два пути: либо увеличить

блок, либо как-то реорганизовать индекс. Блок увеличить можно лишь в ограниченных пределах: во-первых, он должен помещаться в память, во-вторых – поиск в нем ключа не должен заметно сказываться на производительности. С индексом можно поступить интереснее: его можно рассматривать как информационный файл и, в свою очередь, проиндексировать. Таким образом, получается иерархия индексов, каждый элемент которой способен разместиться в память. Начальную загрузку для этого метода делают из сортированного файла.

Добавление записи в информационный файл производится в свободное место выбранного блока. Если свободного места нет, запись либо размещают в дополнительный блок, связанный с выбранным, который называется областью переполнения, либо делят блок пополам, формируя два новых. В первом случае процесс формирования блока проще, но зона последовательного поиска увеличивается на величину блока. Во втором случае процесс деления занимает довольно много времени, но время последующего поиска увеличивается незначительно.

Эффективность доступа зависит от размера индексов и числа уровней их иерархии. Кроме того, на нее влияет размер блоков, наличие в них свободных мест, наличие областей переполнения.

Эффективность хранения в основном зависит от объема свободного места в блоках и от величины индексов.

Пример

Список студентов с оценками по трем предметам размещается в файле, разделенном на блоки размером в четыре записи (справа). Ссылки на блоки хранятся в индексном файле (слева), в качестве ключа выбираются последние фамилии блока. Обратите внимание, что после начальной загрузки в каждом блоке стало по две записи, отсюда и такие значения ключей. Затем информация дописывалась. Второй блок оказался переполненным, образовался блок расширения (в нашем случае – 2.1).

| *Фамилия* | *Блок* |
|---|---|
| Верещак | *1* |
| Кравчук | *2* |
| Мальцев | *3* |
| Харьков | *4* |

| *Блок* | *Фамилия* | *Имя* | *Оц1* | *Оц2* | *Оц3* |
|---|---|---|---|---|---|
| *1* | Агоштон | Екатерина | 4 | 3 | 4 |
| | Верещак | Олеся | 5 | 4 | 4 |
| | Баранова | Виктория | 3 | 5 | 3 |
| | | | | | |
| *2* | Ершова | Татьяна | 3 | 3 | 4 |
| | Кравчук | Татьяна | 5 | 5 | 5 |
| | Заборников | Андрей | 5 | 3 | 5 |
| | Живова | Ирина | 4 | 3 | 4 |
| *3* | Лукша | Роман | 5 | 4 | 3 |
| | Мальцев | Михаил | 5 | 3 | 5 |
| | | | | | |
| | | | | | |
| *4* | Новикова | Наталья | 3 | 4 | 4 |
| | Харьков | Юрий | 4 | 4 | 4 |
| | Рустамова | Камилла | 5 | 5 | 5 |
| | | | | | |
| *2.1* | Зотова | Алина | 5 | 5 | 5 |
| | Корниенко | Павел | 4 | 5 | 5 |
| | | | | | |
| | | | | | |

Конец примера

**Индексно-произвольный метод**

В отличие от предыдущего, этот метод основан на использовании плотного индекса. В этом случае число статей индекса равно количеству информационных записей. Суть метода состоит в следующем. Для информационной структуры (файла) формируется индекс, который содержит значения ключей поиска и ссылки на соответствующие записи. При поиске записи вначале в индексе выбирается статья с искомым ключом, затем по ссылке выбирается непосредственно требуемая запись. Поиск однозначен, если он производится по первичному или другому уникальному индексу. В случае вторичного ключа результат поиска – выборка из записей с равными ключами.

Как и в индексно-последовательном методе, нужно стремиться к тому, чтобы весь индекс размещался в памяти. Но в данном случае, в силу плотности индекса, ситуация хуже из-за большего его размера. Более того, иногда он может превышать размер информационного файла. Уменьшение области поиска достигается, например, построением многоуровневого индекса. Ключи обычно бывают упорядоченными для последующего дихотомического поиска, но не исключаются и другие алгоритмы. Естественно, упорядоченность записей в информационном файле не существенна, однако иногда она позволяет заметно сократить время работы. Например, выдача отчета по всему файлу с сортировкой по ключу поиска приведет к последовательному просмотру статей индекса, но к хаотичному выбору записей в случае их сильного перемешивания по этому ключу. Это, в свою очередь, приводит к «дерганью» головки дисковода, что заметно увеличивает время доступа. Решение проблемы – сортировка по ключу поиска. К замедлению поиска приводит и дублирование значений ключей, следовательно, этот метод наиболее эффективен для первичных индексов.

Итак, эффективность доступа во многом зависит от способа поиска статьи индекса, то есть от способа его организации. Кроме того, на него могут оказывать влияние некоторые свойства ключей (случайное расположение в файле, повторяемость).

Эффективность хранения зависит от размера индекса.

Пример

Список студентов из предыдущего примера размещается в информационном файле (справа), ссылки на записи хранятся в индексном файле (слева), в качестве ключа выбираются фамилии.

| *Фамилия* | *№* |
|---|---|
| Агоштон | 11 |
| Баранова | 14 |
| Верещак | 3 |
| Ершова | 12 |
| Живова | 10 |
| Заборников | 6 |
| Зотова | 7 |
| Корниенко | 9 |
| Кравчук | 1 |
| Лукша | 2 |
| Мальцев | 4 |
| Новикова | 13 |
| Рустамова | 5 |
| Харьков | 8 |

| *№* | *Фамилия* | *Имя* | *Оц1* | *Оц2* | *Оц3* |
|---|---|---|---|---|---|
| 1 | Кравчук | Татьяна | 5 | 5 | 5 |
| 2 | Лукша | Роман | 5 | 4 | 3 |
| 3 | Верещак | Олеся | 5 | 4 | 4 |
| 4 | Мальцев | Михаил | 5 | 3 | 5 |
| 5 | Рустамова | Камилла | 5 | 5 | 5 |
| 6 | Заборников | Андрей | 5 | 3 | 5 |
| 7 | Зотова | Алина | 5 | 5 | 5 |
| 8 | Харьков | Юрий | 4 | 4 | 4 |
| 9 | Корниенко | Павел | 4 | 5 | 5 |
| 10 | Живова | Ирина | 4 | 3 | 4 |
| 11 | Агоштон | Екатерина | 4 | 3 | 4 |
| 12 | Ершова | Татьяна | 3 | 3 | 4 |
| 13 | Новикова | Наталья | 3 | 4 | 4 |
| 14 | Баранова | Виктория | 3 | 5 | 3 |

Конец примера

### Инвертированные списки

Два предыдущих метода ориентировались, в основном, на поиск записей с уникальным значением ключа. Однако нередко возникает задача выбора группы записей по определенным параметрам, каждый из которых не уникален. Более того, записей с каким-то фиксированным значением параметра может быть очень много. Это характерно, например, для библиотечного поиска, когда требуется подобрать книгу с заданным годом издания, автором, издательством и т.п. Для подобных задач существуют специальные методы, наиболее популярный из которых – метод инвертированных списков или *инвертированный* метод.

Считается, что поиск может проводиться по значениям любых полей (вторичных ключей) или их комбинации. Для каждого вторичного ключа создается индекс. В нем на каждое значение ключа формируется список указателей на записи файла с этим значением. Это не обязательно физическая ссылка, допускается и первичный ключ. Таким образом, инвертированный индекс группируется по именам полей, которые в свою очередь группируются по значениям. При поиске записи с заданным значением ключа выбирается нужный индекс, в нем каким-то способом (например, индексно-произвольным) выбирается статья с этим значением, затем выбирается весь список ссылок на записи с искомым значением. Дальнейший выбор записей с одинаковым значением вторичного ключа производится по ссылкам, содержащимся в выбранном списке.

Легко видеть, что поиск по комбинации значений полей сводится к выбору соответствующих списков и их пересечению (операция ***И***) или объединению (операция ***ИЛИ***). Действительно, в пересечении списков содержатся ссылки на записи, удовлетворяющие обоим критериям, а в объединении – хотя бы одному. Критерии могут включать как условия на один ключ, так и на разные. При этом можно использовать не только равенство, но и другие операции отношения. Например, для выбора книг Пушкина, изданных в 1949 году, следует взять пересечение списков «автор = Пушкин» и «год издания = 1949». Выбор книг, изданных позже 2005 года производится по объединению списков, определенных отношением «год издания > 2005».

Более эффективна работа со списками при использовании метода битовых карт, который тоже предназначен для работы с вторичными ключами. Во многом он эквивалентен предыдущему, только вместо списков используются битовые шкалы, длина которых равна количеству информационных записей. Наличие единицы в позиции $N$ означает, что в $N$-й записи значение соответствующего ключа совпадает с искомым значением, наличие нуля – нет. Очевидно, что объединение всех битовых шкал для всех значений заданного ключа дает шкалу, состоящую из одних единиц. В этом методе вместо работы со списками выполняются логические операции с битовыми шкалами. Для выбора искомых записей следует пробежать результирующую битовую шкалу и отобрать записи, номера которых равны позициям шкалы, содержащим единицы. Этот метод хорош, когда количество различных значений вторичных ключей, а значит, и шкал, невелико. В этом случае среднее количество единиц в шкале достаточно велико, что повышает эффективность работы. Заметим, что в предыдущем методе в таких условиях удлиняются списки, что, наоборот, снижает эффективность. Особенно удобно использовать битовые карты при задании сложных условий выбора: операции над битовыми шкалами гораздо проще и быстрее, чем со списками. Понятно, что с ростом количества информационных записей и количества различных вторичных ключей эффективность этого метода падает, особенно эффективность хранения.

Достоинства метода – независимость от объема файла при выборе данных по произвольным значениям ключа, отбор списка записей по сложным условиям без обращения к файлу. Особенно эффективно применение инвертированных списков при выборке данных по совокупности критериев, если атрибуты имеют сравнительно не-

большой диапазон значений. Недостаток – большие затраты времени на создание и обновление инвертированных индексов, причем, время зависит от объема данных. Этот метод обычно используется лишь для поиска. Для начальной загрузки данных и обновления используют другие методы.

Эффективность доступа зависит от эффективности поиска в индексе, но в любом случае она ниже 0,5 (доступ к индексу и доступ к записи файла). Для повышения эффективности следует размещать индексы в оперативной памяти.

Эффективность хранения зависит от метода хранения индекса, от числа инвертируемых полей и от множества значений каждого вторичного ключа (от длины инвертированного списка).

Пример

В приведенном примере в информационном файле (середина) размещается список студентов с оценками по трём предметам. Левый прямоугольник символизирует индекс, в котором находится единственный вторичный ключ – «Оц1». Каждому его значению соответствует список, в котором перечислены соответствующие номера записей информационного файла. Справа от информационного файла такая же конструкция для ключа «Оц2». Выбор всех, получивших «5» по первому предмету, сводится к нахождению в индексе соответствующего значения ключа «Оц1» и загрузки записей, указанных в списке. Если нужно найти тех, кто получил «4» или «5», следует найти и объединить соответствующие списки. Подобные действия выполняются и для другого вторичного ключа. Тогда для выборки тех, кто получил пятерки по обоим предметам, следует в соответствующих индексах найти списки для требуемых значений и взять их пересечение.


| № | Фамилия | Имя | Оц1 | Оц2 | Оц3 |
|---|---|---|---|---|---|
| 1 | Агоштон | Екатерина | 4 | 3 | 4 |
| 2 | Баранова | Виктория | 3 | 5 | 3 |
| 3 | Верещак | Олеся | 5 | 4 | 4 |
| 4 | Ершова | Татьяна | 3 | 3 | 4 |
| 5 | Живова | Ирина | 4 | 3 | 4 |
| 6 | Заборников | Андрей | 5 | 3 | 5 |
| 7 | Зотова | Алина | 5 | 5 | 5 |
| 8 | Корниенко | Павел | 4 | 5 | 5 |
| 9 | Кравчук | Татьяна | 5 | 5 | 5 |
| 10 | Лукша | Роман | 5 | 4 | 3 |
| 11 | Мальцев | Михаил | 5 | 3 | 5 |
| 12 | Новикова | Наталья | 3 | 4 | 4 |
| 13 | Рустамова | Камилла | 5 | 5 | 5 |
| 14 | Харьков | Юрий | 4 | 4 | 4 |




Конец примера

### *Хеширование*

Этот метод называется еще методом перемешанных таблиц. Он представляет собой расширение метода прямого доступа на случай отсутствия взаимно однозначного соответствия между ключом и адресом записи. Существует адресная функция (хеш-функция), которая по ключу формирует адрес, однако, не исключено, что один и тот же адрес выделится разным ключам. Эта ситуация называется *коллизией*, а соответствующие ключи – *синонимами*. Алгоритм хеширования включает в себя механизм разрешения коллизий. Эффективность данного метода доступа во многом зависит от эффективности этого механизма. Кроме того, существенно влияет распределение ключей и размер таблицы. Чем больше размер таблицы по отношению к информационным стро-

кам, тем меньше обычно вероятность коллизий, тем выше эффективность доступа. Простейшая реализация метода заключается в том, что исходя из предположения о равномерном распределении значения ключей, функция хеширования отображает их равномерно на множество допустимых адресов. Простейший способ разрешения коллизий следующий. Если при попытке размещения по указанному адресу выясняется, что там уже что-то лежит, последовательно ищется первое свободное место, при прохождении через конец таблицы указатель возвращается на начало. Если свободная запись найдена – хорошо, в противном случае считается, что таблица переполнена. Аналогично ищутся данные при выборке. Если по указанному адресу есть данные, проверяется их ключ. При несовпадении регистрируется коллизия, которая разрешается, как указано ранее. Если данных нет, поиск неудачен. Этот алгоритм прост, но неэффективен по времени при заполнении таблицы более чем наполовину. Кроме того, при неравномерном распределении ключей этот алгоритм приводит к локальным сгущениям записей и увеличению числа коллизий при относительно свободной таблице. Если есть априорные сведения о распределении ключей, можно построить хеш-функцию, отображающую их опять же равномерно. Это заметно повысит эффективность даже для простого алгоритма разрешения коллизий.

Теперь рассмотрим метод хеширования более подробно. Пусть $M$ – число записей в таблице, $k \in K$ – ключ из множества допустимых значений ключа $K$, $a$ – адрес в таблице, $h(k)$ – хеш-функция, которая по значению ключа формирует адрес: $a=h(k)$, то есть отображает множество ключей $K$ на интервал адресов $[0, M-1]$. Будем считать, что $M$ больше количества возможных записей. Так как рассматриваемый метод не предполагает взаимно однозначного соответствия ключей и адресов, допускается существование таких $k_1$ и $k_2$, что при $k_1 \neq k_2$ выполняется $h(k_1) = h(k_2)$. Эта ситуация, как уже было сказано, называется коллизией, а $k_1$ и $k_2$ называются синонимами.

Эффективность доступа при методе хеширования зависит от распределения ключей, от равномерности распределения адресов хеш-функцией, что влечет уменьшение числа коллизий, и от алгоритма разрешения коллизий.

Эффективность хранения зависит от соотношения между возможным количеством ключей и реальным размером таблицы. Она хуже при слабо заполненных таблицах, обеспечивающих высокую эффективность доступа.

### Методы хеширования

***Метод деления***

В этом методе хеш-функция принимает значение, равное остатку от деления значения ключа на $M$: $h(k)=k \bmod M$. Большое значение для эффективности распределения ключей имеет выбор числа $M$. Очевидно, например, что если оно является степенью системы счисления, значением функции будут младшие разряды ключа. Лучший вариант для уменьшения группировки ключей, если $M$ – простое число.

***Метод умножения***

Функция принимает следующее значение: $h(k)=\lfloor M \times \{k\alpha\} \rfloor$, где $M=2^m$, $\alpha$ – произвольное дробное число, $\{k\alpha\}$ – дробная часть произведения, $\lfloor x \rfloor$ – ближайшее целое, не превосходящее $x$. В качестве $\alpha$ лучше брать достаточно длинный отрезок иррационального числа. Удобно использовать «золотое сечение»: $\alpha = (\sqrt{5}- 1) / 2$. При $\alpha = 1/M$ метод эквивалентен методу деления. К данному методу близок метод середины квадратов: в качестве значения функции берутся средние двоичные разряды числа $k^2$. Однако он хуже метода умножения.

***Преобразование системы счисления***

В основе метода лежит преобразование ключа из системы счисления с основанием $p$ в систему счисления с основанием $q$ ($p<q$) при ограничении $s$ на порядок результата:

$$k = a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p + a_0, \quad h(k) = a_s q^s + a_{s-1} q^{s-1} + \ldots + a_1 q + a_0.$$

Трудоемкость этого метода больше, чем двух предыдущих.

***Деление многочленов***

Пусть $k$ записывается как $k=2^n b_n + 2^{n-1} b_{n-1} + \ldots + 2b_1 + b_0$, $M=2^m$. Представим ключ в виде многочлена $k(t)=b_n t^n + b_{n-1} t^{n-1} + \ldots + b_1 t + b_0$ и определим остаток от деления его на многочлен $c(t)=t^m + c_{m-1} t^{m-1} + \ldots + c_1 t + c_0$. Этот остаток, представленный в двоичной системе, и будет значением функции $h(k)$. Если $c(t)$ – простой неприводимый многочлен, то при условии близких, но неравных ключей $k_1 \neq k_2$ выполняется $h(k_1) \neq h(k_2)$. Этот метод обладает свойством сильного рассеивания ключей.

Изложенные методы рассматривались в предположении, что $k$ – целое положительное число. Но их несложно распространить и на символьный ключ. Достаточно лишь разбить его на отрезки достаточной длины и сложить их двоичные представления. Недостаток этого метода – он слабо чувствителен к порядку символов. Избавиться от этого недостатка позволит циклический сдвиг ключа.

## Методы разрешения коллизий

Рассмотрим теперь некоторые методы разрешения коллизий. Предложенный вначале простой метод нехорош тем, что вызывает вторичные сгущения ключей в таблице и тем самым существенно замедляет доступ. Значит, нужно подобрать функцию, которая рассеивает ключи при появлении коллизии.

Методы разрешения коллизий делятся на два класса: метод цепочек и метод открытой адресации. Метод цепочек в свою очередь делится на методы внутренних и внешних цепочек. Из методов открытой адресации рассмотрим линейное и квадратичное опробование и повторное хеширование.

***Метод внешних цепочек***

Для реализации метода внешних цепочек вместо ключа в строку хеш-таблицы размещается ссылка на начало линейного однонаправленного списка, в который записывается ключ и связанные с ним данные. Список располагается в памяти, внешней по отношению к таблице. При размещении элемента адресная функция указывает на строку хеш-таблицы. Если строка пуста, в нее записывается адрес начала списка, а в первое его звено помещаются данные. Если строка непуста, регистрируется коллизия, синоним записывается в очередное звено списка. При поиске ключа проверяется соответствующая строка таблицы. Если она пуста, поиск неудачен, в противном случае ключ ищется в списке.

***Метод внутренних цепочек***

Метод внутренних цепочек отличается тем, что список строится внутри самой хеш-таблицы. Для этого строка таблицы дополняется полем для ссылки на элемент этой же таблицы. При возникновении коллизии в таблице каким-то способом ищется свободное место, в него размещаются данные, а ссылка на это место записывается в строку, на которую указывала хеш-функция. При каждой следующей коллизии список удлиняется аналогичным образом. Этот метод имеет ту неприятную особенность, что построенные таким способом цепочки имеют тенденцию срастаться. Предположим, хеш-функция обрабатывает ключ, который не имел синонимов, но при попытке записать данные в таблицу выясняется, что там уже что-то есть. В чем дело? Оказывается, при разрешении какой-то предыдущей коллизии это место было использовано как свободное на тот момент. Тогда текущие данные запишутся в тот же список, хотя ключ и не синоним тому, который этот список породил. Итак, два независимых списка срослись, что увеличило время поиска как по первой группе синонимов, так и по второй. Этого можно избежать, переместив чужой ключ на свободное место, но такая работа требует затрат. Этот метод называется еще *методом срастающихся цепочек*.

***Метод линейного опробования***

Это простейший метод открытой адресации, который уже приводился в качестве примера. При появлении коллизии последовательно ищется свободное место, и если оно находится, туда записывается ключ (строка данных). При достижении границы таблицы адрес циклически переносится на другой ее конец. В случае записи отсутствие свободного места говорит о переполнении таблицы. При поиске необходимо в строке-кандидате проверять ключ. Если он совпадает с искомым – поиск успешный, в противном случае поиск продолжается среди синонимов. Выбор пустой строки или обнаружение, что вся таблица пройдена, обозначает, что ключ не найден.

В общем случае поиск свободного места производится с некоторым шагом ***c***, формула вычисления очередного адреса:

$$h_i(k)=h_0(k)+ci,$$

где ***k*** – значение ключа, $h_0(k)$ – начальное значение хеш-функции, ***c*** – постоянный шаг, ***i*** – номер итерации. Заметим, что этот метод приводит к локальным сгущениям синонимов, особенно в случаях срастания двух или более соседних групп. Чтобы улучшить ситуацию, выбирают ***c*** и ***M*** взаимно простыми, причем, будет лучше, если ***c*** достаточно велико. Понятно, что вероятность сгущениям синонимов существенно снижается, если таблица заполнена слабо.

***Метод квадратичного опробования***

Этот метод похож на предыдущий, но в формулу добавляется нелинейный член:

$$h_i(k)=h_0(k)+ci+di^2,$$

где обозначения те же, что и раньше, ***d*** – константа. Благодаря нелинейности удается существенно уменьшить число проб, но для достаточно плотно заполненной таблицы избежать срастания групп различных синонимов не удается.

***Метод двойного хеширования***

В этом методе при возникновении коллизии для вычисления следующего адреса используется еще одна хеш-функция:

$$h_i(k)=h_0(k)+ig(k),$$

где $g(k)$ – хеш-функция, возможно, похожая на $h_0(k)$, но не эквивалентная ей. Например, если ***M*** – простое число, можно использовать следующие функции:

$$h_i(k)=(k \bmod M)+i(1+k \bmod (M-2)),$$

$$h_i(k)=h_0(k)+i(M-h_0(k))$$ и т.п.

Во втором примере $h_0(k)$ и $g(k)$ зависимы. Это может ускорить вычисления, но если $h_0(k)$ приводила к сгущениям, то и $h_i(k)$ будет обладать тем же свойством. Для уменьшения подобных неприятностей вводится некоторая независимая величина ***r***, которая управляет последовательностью проб. Тогда после $h_i(k)$ будет следовать не $h_{i+1}(k)$, а $h_{i+r}(k)$. Если $h_0$ и ***g*** – независимые функции, вероятность коллизии будет $1/M^2$.

### Методы удаления и переразмещения (рехеширования)

Представим, что нам потребовалось из хеш-таблицы удалить некоторый ключ. Очевидно, что если использовался метод открытой адресации, сделать это впрямую невозможно: ключ может входить в цепочку синонимов, и его удаление приведет к разрушению этой цепочки. Тогда можно вместо удаления ключа «стереть» его, заменив значением, выходящим за границы области определения этого ключа. При поиске это значение будет пропускаться, а при вставке это место будет рассматриваться как пустое. В этом случае каждая позиция хеш-таблицы будет в одном из трех состояний: свободная,

занятая, удаленная. Понятно, что однажды занятая позиция не может стать свободной, поэтому с исчерпанием ресурса свободных мест время поиска будет расти.

Для метода внешних цепочек удаление вполне возможно, оно эквивалентно удалению из линейного однонаправленного списка. С методом внутренних цепочек и похожим на него методом линейного опробования тоже можно провести подобную работу в силу определенности цепочек. Но в этих случаях для ликвидации разрыва цепочки потребуется сдвинуть (переписать) хвост списка. Во время сдвига надо будет выяснить, не принадлежит ли очередной ключ к чужому синониму, если это так – обойти его, не сдвигая. Для удобства удаления можно потребовать, чтобы каждый список синонимов начинался с того адреса, на который указывает хеш-функция. Тогда при записи первого ключа нужно освободить соответствующую строку, если она была занята чужим синонимом. Подобная работа может занять определенное время, так что подобные ограничения корректны в тех случаях, когда количество вставок существенно меньше, чем количество поисков.

Одна из проблем метода хеширования – выбор размера таблиц. Если выбрать большую таблицу, будет неэффективно использоваться память, если маленькую – будет заметно расти время доступа с увеличением процента заполнения таблицы. В простейшем случае работы с хеш-таблицами, который приводился в примере, заполнение таблицы более, чем на половину приводило к резкому росту времени поиска, работа пользователей практически была парализована. Были и другие наблюдения за ростом времени, но все, по сути, сводилось к одному: при достижении некоторого критического процента заполнения работа с таблицей становилась невозможной. К сожалению, нет возможности простого увеличения размера таблицы в силу того, что функции хеширования зависят от этого размера. Тогда следует создать новую таблицу большего размера и переписать в нее содержимое старой. Эта процедура требует значительного времени, ее, чаще всего, невозможно выполнить в оперативном режиме, но другого выхода нет. Эта процедура называется *переразмещением* или *рехешированием*.

### Анализ метода

Каким бы ни был выбор хеш-функции, возможны коллизии. Это связано с вероятностными законами, лежащими в основе метода. Метод может показать хорошие результаты в одном случае и быть неприемлемым в другом. Тем не менее, возможно провести его анализ для определения некоторых характеристик.

Предположим, что существует хеш-функция, равномерно распределяющая ключи в таблице. Доказано, что в этом случае среднее число проб $C'_n$, необходимое для вставки ключа в таблицу размера $M$, содержащую $n$ ключей, для методов открытой адресации минимально. Обозначим $\alpha = n/M$ – коэффициент заполненности таблицы, $C$ – среднее число проб для поиска существующего ключа, $C'$ – среднее число проб для вставки ключа при больших $n$ и $M$. Тогда для случая отсутствия коллизий

$$C' = 1/(1-\alpha) + O(1/M)$$

$$C = (1/\alpha)\ \ln(1/(1-\alpha)) + O(1/M)$$

При наличии коллизий эти величины зависят от метода их разрешения. Исключение составляет метод двойного хеширования, характеристики которого практически совпадают с приведенными.

| | | |
|---|---|---|
| *Метод внешних цепочек* | $C' = \alpha + e^{-\alpha}$ | $C = 1 + \alpha/2$ |
| *Метод внутренних цепочек* | $C' = 1 + 1/4(e^{2\alpha} - 1 - 2\alpha)$ | $C = 1 + \alpha/4 + 1/(8\alpha)\ (e^{2\alpha} - 1 - 2\alpha)$ |
| *Линейное опробование* | $C' = \frac{1}{2}(1 + 1/(1-\alpha))$ | $C = \frac{1}{2}(1 + 1/(1-\alpha)^2)$ |
| *Двойное хеширование* | $C' = 1/(1-\alpha)$ | $C = (1/\alpha)\ \ln(1/(1-\alpha))$ |

Здесь подразумевается наличие слагаемого $O(1/M)$.

Возникает вопрос, существует ли идеальная функция хеширования, распределяющая ключи без коллизий? Существует теорема, утверждающая, что такую функцию для заранее известного множества ключей построить можно, но объем вычислений с ростом количества ключей резко растет и при их количестве, превышающем 20, становится неприемлемым. Для некоторых задач такой вариант допустим, но при поиске в больших таблицах во внешней памяти он бесполезен.

# Лекция 14. Функциональные зависимости

Последующие лекции будут посвящены более подробному изучению нормальных форм на базе работы Мейера [17]. В силу краткости курса приводится изложение лишь тех разделов, которые дают представление о способах минимизации действий при контроле соблюдения функциональных зависимостей в базе данных. В идеальном варианте должен рассматриваться алгоритм синтеза базы данных по множеству функциональных зависимостей, но реально для его внятного изложения необходимо больше времени. Интересующихся отсылаю к работе [17]. Краткость курса не позволяет давать достаточно сложные доказательства утверждений, которые приводятся в лекциях. По той же причине совершенно не обсуждаются многозначные зависимости и соответствующие нормальные формы.

Теория нормальных форм базируется на понятии функциональных зависимостей (ФЗ). Общее представление о них дано в обзорной лекции по нормализации. Теперь дадим определение функциональной зависимости на некотором отношении, а затем придём к функциональным зависимостям на схеме, о чём и говорилось в обзорной лекции.

***Определение.*** *Пусть r – отношение со схемой R; $X,Y \subseteq R$. Отношение r удовлетворяет функциональной зависимости $X \to Y$, если $\pi_Y(\sigma_{X=x}(R))$ имеет не более, чем один кортеж для каждого X-значения x, то есть, для $t_1, t_2 \in r$ : $(t_1(X)=t_2(X)) \Rightarrow (t_1(Y)=t_2(Y))$.*

Помимо обычных *X* и *Y*, рассмотрим варианты с пустыми подмножествами атрибутов. Будем считать, что $X \to \varnothing$ тривиально удовлетворяет любому отношению, а $\varnothing \to Y$ удовлетворяет отношениям, в которых *Y*-значения всех кортежей совпадают.

Пользуясь приведенным определением, для заданного отношения *r* и множеств атрибутов $X,Y \subseteq R$ можно построить алгоритм, проверяющий существование ФЗ $X \to Y$. Такой алгоритм приведен в [17] (алгоритм *SATISFIES*). Суть его в том, что кортежи отношения группируются по *X*-значениям, при этом все *Y*-значения для одинаковых *X*-значений должны быть одинаковыми.

Алгоритм *SATISFIES*

Вход: отношение *r*, ФЗ $X \to Y$.

Выход: **истина**, если *r* удовлетворяет $X \to Y$, **ложь** в противном случае.

1. Отсортировать *r* по *X*-столбцам, группируя равные значения.
2. Если каждая группа *X*-кортежей имеет одинаковые *Y*-значения, возвратить **истину**, в противном случае – **ложь.**

Пример

Дан график распределения пилотов по рейсам. Проверим ФЗ *Рейс→Время* и *Время→Рейс*.

***Исходный график***

| *Пилот* | *Рейс* | *Дата* | *Время* |
|---|---|---|---|
| Иванов | 83 | 09.01.2000 | 10:00 |
| Иванов | 16 | 10.01.2000 | 13:00 |
| Петров | 81 | 08.01.2000 | 05:00 |
| Петров | 31 | 12.01.2000 | 18:00 |
| Петров | 83 | 11.01.2000 | 10:00 |
| Сидоров | 83 | 13.01.2000 | 10:00 |
| Сидоров | 16 | 12.01.2000 | 13:00 |
| Федоров | 81 | 09.01.2000 | 05:00 |
| Федоров | 81 | 13.01.2000 | 05:00 |
| Федоров | 41 | 15.01.2000 | 13:00 |

***Группировка по рейсу***

| *Пилот* | *Рейс* | *Дата* | *Время* |
|---|---|---|---|
| Иванов | 83 | 09.01.2000 | 10:00 |
| Петров | 83 | 11.01.2000 | 10:00 |
| Сидоров | 83 | 13.01.2000 | 10:00 |
| Петров | 81 | 08.01.2000 | 05:00 |
| Федоров | 81 | 09.01.2000 | 05:00 |
| Федоров | 81 | 13.01.2000 | 05:00 |
| Федоров | 41 | 15.01.2000 | 13:00 |
| Петров | 31 | 12.01.2000 | 18:00 |
| Иванов | 16 | 10.01.2000 | 13:00 |
| Сидоров | 16 | 12.01.2000 | 13:00 |

***Группировка по времени***

| *Пилот* | *Рейс* | *Дата* | *Время* |
|---|---|---|---|
| Петров | 81 | 08.01.2000 | 05:00 |
| Федоров | 81 | 09.01.2000 | 05:00 |
| Федоров | 81 | 13.01.2000 | 05:00 |
| Иванов | 83 | 09.01.2000 | 10:00 |
| Петров | 83 | 11.01.2000 | 10:00 |
| Сидоров | 83 | 13.01.2000 | 10:00 |
| Иванов | 16 | 10.01.2000 | 13:00 |
| Сидоров | 16 | 12.01.2000 | 13:00 |
| Федоров | 41 | 15.01.2000 | 13:00 |
| Петров | 31 | 12.01.2000 | 18:00 |

Легко видеть, что в первом случае функциональная зависимость есть, во втором – нет.
Конец примера

### *Аксиомы вывода*

Для отношения *r(R)* в любой момент существует некоторое семейство ФЗ, которому оно удовлетворяет, причем, одно его состояние может удовлетворять данной ФЗ, другое – нет. Требуется выявить семейство ФЗ *F*, которому удовлетворяют все допустимые состояния *r*, то есть определить семейство *F*, заданное на схеме *R*.

Множество ФЗ, применимых к *r(R)*, конечно, поэтому можно найти все ФЗ, которым удовлетворяет *r* (например, применяя алгоритм, рассмотренный ранее). Но это достаточно долгий процесс, который зависит от мощности множества ФЗ и от количества атрибутов, составляющих эти зависимости. Возникает вопрос, можно ли для данного множества функциональных зависимостей найти семантически эквивалентное, но меньшее. Иногда это оказывается возможным. В дальнейших лекциях, в основном, обсуждается эта проблема.

***Определение.*** *Будем говорить, что множество ФЗ F влечет ФЗ* $X \to Y$ ($F \models X \to Y$), *если каждое отношение, удовлетворяющее всем зависимостям из F, удовлетворяет и* $X \to Y$.

В 1974 году Армстронг предложил ряд правил, пользуясь которыми можно по заданному множеству функциональных зависимостей *F* построить такое множество *G*, что если отношение удовлетворяет всем ФЗ $f \in F$, оно удовлетворяет и всем $g \in G$. Эти правила, устанавливающие определённые свойства функциональных зависимостей, названы аксиомами вывода. Рассмотрим эти аксиомы.

*F1. Рефлексивность*

$$X \to X$$

*F2. Пополнение (расширение левой части)*

$$(X \to Y) \Rightarrow (XZ \to Y)$$

Пример 1

*r (A B C D)*

| | | | |
|---|---|---|---|
| *a1* | *b1* | *c1* | *d1* |
| *a2* | *b2* | *c1* | *d1* |
| *a1* | *b1* | *c1* | *d2* |
| *a3* | *b3* | *c2* | *d3* |

Здесь $(A \to B) \Rightarrow \{ AB \to B, AC \to B, AD \to B, ABC \to B, ABD \to B \}$

Конец примера

*F3. Аддитивность*
Позволяет объединить две ФЗ с одинаковыми левыми частями.

$$(X \to Y, X \to Z) \Rightarrow (X \to YZ)$$

Пример 2

Для отношения из примера 1: $(A \to B, A \to C) \Rightarrow (A \to BC)$
Конец примера

*F4. Проективность*
В некоторой степени, обратная *F3*.

$$(X \to YZ) \Rightarrow (X \to Y)$$

*F5. Транзитивность*

$$(X \to Y, Y \to Z) \Rightarrow (X \to Z)$$

Пример 3

Для отношения из примера 1: $(A \to B, B \to C) \Rightarrow (A \to C)$
Конец примера

*F6. Псевдотранзитивность*

$$(X \to Y, YZ \to W) \Rightarrow (XZ \to W)$$

Эта система аксиом избыточна. Например, $F6 \Rightarrow F5$ (для $Z=\varnothing$), $(F1, F2, F3, F5) \Rightarrow F6$. Но она полна, то есть любая ФЗ, которая следует из $F$, может быть получена применением аксиом *F1-F6*.

Можно доказать [17], что *{F1, F2, F6}* – полное подмножество аксиом: $(F1, F2, F6) \Rightarrow F3$, $(F1, F2, F6) \Rightarrow F4$, $F6 \Rightarrow F5$. Например, докажем *F4*. Пусть $X \to YZ$, тогда из *(F1)*: $Y \to Y$ и (F2): $YZ \to Y$. По *(F6)*: $X \to Y$. Утверждение доказано.

Подмножество независимых аксиом *{F1, F2, F6}* носит название *аксиом Армстронга*. Впрочем, иногда [10, 16] аксиомами Армстронга называют все шесть аксиом.

***Определение.*** *Пусть F – множество ФЗ для r(R). Замыкание F (обозначается $F^+$) – это наименьшее множество, содержащее F, и такое, что при применении к нему аксиом Армстронга нельзя получить ни одной ФЗ, не принадлежащей F.*

***Определение.*** *Два множества ФЗ F и G над одной и той же схемой называются эквивалентными $F \equiv G$, если $F^+ = G^+$.*

Если $F \models X \to Y$, то либо $X \to Y \in F$, либо её можно получить путём последовательного применения аксиом вывода к $F$. Эта последовательность аксиом называется выводом $X \to Y$ из $F$.

***Определение.*** *Последовательность $P$ функциональных зависимостей называется последовательностью вывода на $F$, если каждая ФЗ из $P$ либо принадлежит $F$, либо следует из предыдущих ФЗ в $P$ после применения к ним одной из аксиом вывода.*

Пример 4

$F = \{AB \to E, AG \to J, BE \to I, E \to G, GI \to H\}$. Последовательность вывода определяется неоднозначно, например для $AB \to GH$ она может выглядеть так (справа указана аксиома и номера элементов последовательности вывода, к которым она применяется):

*1. $AB \to E$;*
*2. $AB \to AB$ (F1: 1);*
*3. $AB \to B$ (F4: 2);*

*4. AB→BE (F3: 1, 2);*
*5. BE→I;*
*6. AB→I (F5: 4, 5);*
*7. E→G;*
*8. AB→G (F5:1, 7);*
*9. AB→GI (F3: 6,8);*
*10. GI→H;*
*11. AB→H (F5: 9, 10);*
*12. AB→GH (F3: 8, 11).*

Очевидно, что эта последовательность будет, в частности, последовательностью вывода для других ФЗ, например, *AB→GI*.

Конец примера

***Определение.*** *Используемое множество в последовательности вывода P – множество ФЗ из* $F$*, принадлежащее* $P$.

### *B-аксиомы и RAP-последовательности вывода*

Кроме аксиом Армстронга, часто рассматривается другая полная система аксиом, которая называется *B-аксиомами*.

B1. Рефлексивность (*Reflexivity*)

$$X \to X$$

B2. Накопление (*Accumulation*)

$$(X \to YZ,\ Z \to CW) \Rightarrow (X \to YZC)$$

B3. Проективность (*Projectivity*)

$$(X \to YZ) \Rightarrow (X \to Y)$$

Пример 5

Пусть *F* – множество ФЗ из примера 5. Приведём последовательность вывода для *AB→GH*, использующую только *B*-аксиомы:

1. *EI→EI* (B1);
2. *E→G*;
3. *EI→EGI* (B2);
4. *EI→GI* (B3);
5. *GI→H*;
6. *EI→GHI* (B2);
7. *EI→GH* (B3);
8. *AB→AB* (B1);
9. *AB→E*;
10. *AB→ABE* (B2);
11. *BE→I*;
12. *AB→ABEI* (B2);
13. *AB→ABEGI* (B2);
14. *AB→ABEGIH* (B2);
15. *AB→GH* (B3).

Конец примера

Можно показать [17], что аксиомы Армстронга выводятся из *B*-аксиом. Из полноты системы аксиом Армстронга следует и полнота системы *B*-аксиом.

***Определение.*** *Последовательность вывода $X \to Y$ из $F$, полученная при помощи B-аксиом называется RAP-последовательностью (по первым буквам названия B-аксиом), если она удовлетворяет следующим условиям:*

1) *первая ФЗ – это $X \to X$;*
2) *последняя ФЗ – это $X \to Y$;*
3) *каждая ФЗ (за исключением первой и последней) либо принадлежит $F$, либо имеет вид $X \to Z$ и получена с помощью аксиомы* B2.

Пример 6

Пусть $F$ – множество ФЗ из примера 5. Выпишем RAP-последовательность вывода $AB \to GH$ из $F$:

1. $AB \to AB$ (B1);
2. $AB \to E$;
3. $AB \to ABE$ (B2);
4. $BE \to I$;
5. $AB \to ABEI$ (B2);
6. $E \to G$;
7. $AB \to ABEGI$ (B2);
8. $GI \to H$;
9. $AB \to ABEGIH$ (B2);
10. $AB \to GH$.

Конец примера

***Теорема.*** *Пусть $F$ – множество ФЗ. Если существует последовательность вывода из $F$ для $X \to Y$, то существует RAP-последовательность вывода из $F$ для $X \to Y$.*

### *Ориентированный ациклический граф вывода*

Ориентированный (*directed*) ациклический (*acyclic*) граф (*DA*-граф) – это орграф, не имеющий циклов. Помеченный *DA*-граф – это *DA*-граф, каждой вершине которого поставлен в соответствие некоторый элемент из множества меток $L$.

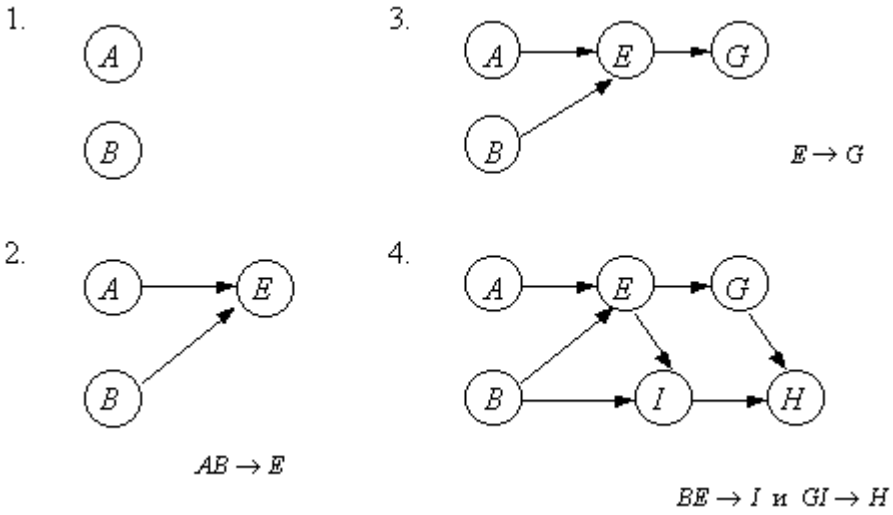
***Определение.*** *Пусть $F$ – множество ФЗ над схемой $R$. DA-граф вывода над $F$ – это DA-граф, помеченный символами атрибутов из $R$ и построенный по следующим правилам.*

1. *Множество изолированных вершин является DA-графом вывода над $F$.*
2. *Пусть DA-граф вывода над $F$ содержит **n** вершин $\nu_i$ с метками $A_i$ и в $F$ существует ФЗ $A_1A_2\ldots A_k \to CZ$ ($k \le n$). Определим новый граф, добавив к исходному DA-графу вывода вершину **u** с меткой $C$ и дуги $(\nu_1, u), (\nu_2, u), \ldots, (\nu_k, u)$. Полученный граф является DA-графом вывода над $F$.*
3. *Никакой другой граф не является DA-графом вывода над $F$.*

Сокращённо *DA*-граф вывода над $F$ называют *DDA*-графом над $F$ (от английского *derivation directed acyclic*). Любой *DDA*-граф получается однократным применением правила (1) и многократным применением правила (2).

Пример 7

Пусть $F$ – множество ФЗ из примера 5: $F = \{AB \to E, AG \to J, BE \to I, E \to G, GI \to H\}$. Представим этапы построения *DDA*-графа над $F$ для $AB \to GH$.




Конец примера

***Определение.*** *Пусть H – DDA-граф, содержащий вершину v, которая не имеет входящих дуг. Тогда v называется начальной вершиной. Начальная вершина добавляется с помощью правила (1).*

***Определение.*** *Пусть H – DDA-граф над F. H называется DDA-графом для X→Y, если X – множество меток начальных вершин и каждый атрибут в Y – метка вершины в H.*

***Определение.*** *Используемым множеством* ***U****(H) в DDA-графе H над F называется множество всех ФЗ в F, использованных при применении правила (2) во время построения графа H.*

Пример 8

В примере 7 *DDA*-граф для ФЗ *AB→AB* содержит две изолированных вершины. Его используемое множество пусто.

Конец примера

***Теорема.*** *Для множества ФЗ F над R и ФЗ X→Y следующие утверждения эквивалентны:*

1. *F |= X→Y;*
2. *Существует последовательность вывода на F для X→Y;*
3. *Существует DDA-граф над F для X→Y.*

***Следствие.*** *DDA-граф H над F для X→Y с* ***U****(H)=G существует тогда и только тогда, когда существует RAP-последовательность вывода на F для X→Y с используемым множеством G.*

При формировании множества функциональных зависимостей возникает вопрос: выводима ли очередная добавляемая функциональная зависимость из уже существующих? Если это так, она лишняя. Другими словами, надо проверить, содержится ли она в замыкании множества ранее существующих зависимостей. В [17] приводится алгоритм, проверяющий принадлежность $X \rightarrow Y$ множеству $F^+$ (выводимость функциональной зависимости) с временной сложностью *O(n)*, где *n* – количество ФЗ в *F*.

### *Определение реляционной базы данных*

Сформулируем определение ключа в терминах ФЗ. Ключ для схемы $R$ – это подмножество $K \subseteq R$, такое, что любое отношение $r(R)$ удовлетворяет ФЗ $K \to R$, но никакое собственное подмножество $K' \subset K$ этим свойством не обладает.

Будем считать, что схема $R$ некоторого отношения состоит из двух частей: $S$ и $\boldsymbol{K}$, где $S$ – множество атрибутов, $\boldsymbol{K}$ – множество выделенных ключей. Выделенным ключом может быть, в частности, суперключ. Введём обозначение $R=(S, \boldsymbol{K})$.

***Определение.*** *Пусть* $\boldsymbol{U}$ *– множество атрибутов. Схемой реляционной БД* $\boldsymbol{R}$ *над* $\boldsymbol{U}$ *называется совокупность схем отношений* $\{R_1, R_2,\ldots, R_n\}$, *где* $R_i=(S_i, \boldsymbol{K_i})$, $i = 1,2,\ldots, n$, $\cup^{n}_{i=1} S_i = \boldsymbol{U}$, *и* $S_i \neq S_j$ *при* $i \neq j$.

Согласно этому определению, в базе данных не может быть двух отношений с одинаковыми схемами. Однако реально необходимость в этом может быть.

***Определение.*** *Реляционной БД d со схемой БД* $\boldsymbol{R}$ *называется такая совокупность отношений* $d=\{r_1, r_2,\ldots, r_n\}$, $r_i = r_i(S_i)$, $i=1,\ldots,n$, *что для каждой схемы* $R = (S, \boldsymbol{K}) \in \boldsymbol{R}$ *существует отношение* $r(S) \in d$ *и удовлетворяющее каждому ключу из* $\boldsymbol{K}$.

Пример

***рейс***

| *Пилот* | *Рейс* | *Дата* |
|---|---|---|
| Иванов | 83 | 09.01.2000 |
| Петров | 83 | 11.01.2000 |
| Сидоров | 83 | 13.01.2000 |
| Петров | 81 | 08.01.2000 |
| Федоров | 81 | 09.01.2000 |
| Федоров | 81 | 13.01.2000 |
| Федоров | 41 | 15.01.2000 |
| Петров | 31 | 12.01.2000 |
| Иванов | 16 | 10.01.2000 |
| Сидоров | 16 | 12.01.2000 |

***время***

| *Рейс* | *Время* |
|---|---|
| 83 | 10:00 |
| 81 | 05:00 |
| 41 | 13:00 |
| 31 | 18:00 |
| 16 | 13:00 |
| 16 | 13:00 |

БД $d=\{$*рейс*, *время*$\}$ имеет схему

$\boldsymbol{R}$ = {(Пилот Рейс Дата, {Пилот Дата}), (Рейс Время, {Рейс})}.

Конец примера

### *Представление множества функциональных зависимостей*

Ряд следующих простых определений потребуются для дальнейшего изложения.

***Определение.*** *Если K – выделенный ключ из* $\boldsymbol{K}$, *схема* $R = (S, \boldsymbol{K})$ *включает ФЗ* $K \to R$.

***Определение.*** *Схема БД* $\boldsymbol{R} = \{R_1, R_2,\ldots, R_n\}$ *представляет множество ФЗ* $G = \{K \to Y \mid \exists R_i \in \boldsymbol{R}$, *которая включает* $K \to Y\}$. *Говорят, что схема БД* $\boldsymbol{R}$ *полностью характеризует множество ФЗ F, если* $F \equiv G$.

Пример

Схема БД из предыдущего примера представляет множество ФЗ $G = \{$*Пилот Дата* $\to$ *Рейс Дата, Рейс* $\to$ *Рейс Время*$\}$. Она полностью характеризует множество $F = \{$*Пилот Дата* $\to$ *Рейс Время, Рейс* $\to$ *Время*$\}$.

Конец примера

***Определение.*** *ФЗ $X \to Y$ применима к R, если $X \subseteq R$ и $Y \subseteq R$.*

***Определение.*** *Пусть $d=\{r_1, r_2, \ldots, r_n\}$ – база данных со схемой $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ над **U**. Она удовлетворяет F, если каждая $X \to Y$ из $F^+$, применимая к схеме $R_i$ из **R**, выполняется в каком-то отношении $r_i$.*

***Определение.*** *Пусть G – множество всех ФЗ в $F^+$, которые применимы к какой-нибудь схеме $R_i$ в **R**. Любая ФЗ в $G^+$ называется навязанной **R**, а ФЗ из $(F^+ – G^+)$ – ненавязанной **R**. Множество F навязано схеме БД **R**, если каждая ФЗ в $F^+$ навязана **R**.*

***Определение.*** *БД d со схемой **R** подчиняется множеству ФЗ F, если F навязано схеме **R** и d удовлетворяет $F^+$.*

То же самое в виде краткой таблицы (подробности в определении).

| | |
|---|---|
| *ФЗ применима к R* | *если $X \subseteq R$ и $Y \subseteq R$* |
| *БД удовлетворяет F* | *если $\forall X \to Y \in F^+$, применимая к $R_i$, выполняется в $r_i$* |
| *ФЗ $\in G^+$ навязана **R*** | *если $G \in F^+$, G – все применимые к $R_i$ ФЗ* |
| *ФЗ не навязана **R*** | *если ФЗ $\in (F^+ - G^+)$* |
| *F навязано **R*** | *если $\forall$ ФЗ $\in F^+$ навязана* |
| *d(**R**) подчиняется F* | *если (F навязано **R**) & (d удовлетворяет $F^+$)* |

Пример

Рассмотрим схему БД $\mathbf{R} = \{R_1, R_2, R_3\}$, где $R_1 = ABC$, $R_2 = BCD$, $R_3 = DE$, и множество ФЗ $F = \{A \to BC, C \to A, A \to D, D \to E, A \to E\}$.

Функциональные зависимости $A \to D$ и $A \to E$ неприменимы ни к одной схеме из **R**.

Однако множество F навязано схеме **R**, так как существует $G = \{A \to BC, C \to A, C \to D, D \to E\}$, эквивалентное F, каждая функциональная зависимость которого применима к некоторой схеме из **R**. А множество $\{A \to D\}$, как легко видеть, не навязано **R**.

Конец примера

# Лекция 15. Покрытия функциональных зависимостей

При работе с базами данных необходимо поддерживать её корректность, то есть следить, чтобы выполнялись все функциональные зависимости, которым она удовлетворяет. Поэтому эффективность работы с базой данных зависит от представления функциональных зависимостей. Следовательно, нужно найти такое множество ФЗ, которое, будучи эквивалентным заданному, обладает лучшими в каком-то смысле свойствами. Рассмотрим методы представления ФЗ, которые позволят упростить эту задачу.

Пример

Пусть $F = \{A\to B, B\to C, A\to C, AB\to C, A\to BC\}$, $G = \{A\to B, B\to C\}$. Здесь все ФЗ из $F$ выводятся из $G$, то есть $F \equiv G$. Однако представление $G$ предпочтительнее: временная сложность алгоритма оценки множества ФЗ зависит от его объема.
Конец примера

***Определение.*** *Множество ФЗ $F$ называется покрытием $G$, если $F \equiv G$.*

Очевидно, что определение симметрично относительно множеств ФЗ, каждое из них будет покрытием другого, но обычно подразумевают, что объем покрытия меньше. Так как $F^+ = G^+$, для $\forall\ X\to Y \in G$ выполняется $F \models X\to Y$.

### Лемма об эквивалентности функциональных зависимостей

Обобщим понятие выводимости. Будем считать, что $F \models G$, если верно, что $F \models X\to Y$ для $\forall\ X\to Y \in G$.

***Лемма.*** *Для заданных множеств ФЗ F и G над схемой R тождество $F \equiv G$ имеет место тогда и только тогда, когда $F \models G$ и $G \models F$.*

***Доказательство***

Пусть $F \equiv G$, тогда для каждой ФЗ $X\to Y$ из $F$ имеет место $G \models X\to Y$, то есть $G \models F$, аналогично $F \models G$.

Пусть теперь $F \models G$, тогда $G \subseteq F^+$, применяя операцию замыкания к обеим частям, получим $G^+ \subseteq (F^+)^+ = F^+$, аналогично $(G \models F) \Rightarrow (F^+ \subseteq G^+)$, таким образом, $F^+ = G^+$.

Из этой леммы следует простой способ проверки эквивалентности множеств функциональных зависимостей, который использует алгоритм определения принадлежности функциональной зависимости замыканию множества ФЗ.

### Неизбыточные покрытия

***Определение.*** *Множество ФЗ F неизбыточно, если у него нет собственного подмножества $F' \subset F$, такого, что $F' \equiv F$. Если F – покрытие G, то F – неизбыточное покрытие G.*

Пример

Пусть $G = \{AB\to C, A\to B, B\to C, A\to C\}$, $F = \{AB\to C, A\to B, B\to C\}$. Здесь $F \equiv G$, но $F$ – избыточное покрытие, так как существует $F' = \{A\to B, B\to C\}$, $F' \equiv F$.
Конец примера

Функциональная зависимость $X\to Y \in F$ избыточна в $F$, если $\{F - \{X\to Y\}\} \models X\to Y$. Множество $F$ неизбыточно, если в нём нет избыточных функциональных зависимостей.

Приведём алгоритм построения неизбыточного покрытия. На вход алгоритма поступает некоторое множество ФЗ $G$, на выходе формируется его неизбыточное покрытие $F$. Заметим, что исходное множество может иметь более одного неизбыточного покрытия.

***Алгоритм.*** *Сначала положим $F = G$, затем для каждой ФЗ $X \to Y$ из $G$ проверяем её принадлежность $\{F - \{X \to Y\}\}^+$, и если $\{F - \{X \to Y\}\} \models X \to Y$, то $F := F - \{X \to Y\}$.*

Пример

Пусть $F = \{A \to B, B \to A, B \to C, A \to C\}$. Результатом применения алгоритма при выборе функциональных зависимостей в заданном порядке будет множество $\{A \to B, B \to A, A \to C\}$. Если их выбирать в порядке $A \to B, A \to C, B \to A, B \to C$, то в результате применения алгоритма получим $\{A \to B, B \to A, B \to C\}$.
Конец примера

### *Посторонние атрибуты*

Пусть $F$ – неизбыточное множество ФЗ. В этом случае удаление любой ФЗ приводит к множеству, не эквивалентному данному. Но ситуацию иногда можно улучшить за счет уменьшения количества атрибутов в некоторых ФЗ.

***Определение.*** *Атрибут $A \in R$ называется посторонним в $X \to Y \in F$, если выполнено хотя бы одно из следующих условий:*

1. $X = AZ$, $X \neq Z$, $\{F - \{X \to Y\}\} \cup \{Z \to Y\} \equiv F$;
2. $Y = AW$, $Y \neq W$, $\{F - \{X \to Y\}\} \cup \{X \to W\} \equiv F$.

Иначе говоря, атрибут посторонний в ФЗ относительно некоторого множества ФЗ, если его можно удалить из правой или левой частей этой ФЗ без изменения замыкания получившегося множества ФЗ.

Пример

Пусть $F = \{A \to BC, B \to C, AB \to D\}$.
Атрибут $C$ посторонний в правой части $A \to BC$.
Атрибут $B$ посторонний в левой части $AB \to D$.
Конец примера

***Определение.*** *ФЗ $X \to Y \in F$ называется полной или редуцированной слева, если $X$ не содержит атрибута $A$, постороннего в $X \to Y$.*

***Определение.*** *ФЗ $X \to Y \in F$ редуцированной справа, если $Y$ не содержит атрибута $A$, постороннего в $X \to Y$.*

***Определение.*** *ФЗ называется редуцированной, если она полная, редуцированная справа и $Y \neq \varnothing$. Множество $F$ называется редуцированным (слева, например), если редуцирована (слева) каждая его ФЗ.*

### *Канонические покрытия*

***Определение.*** *Неизбыточное множество ФЗ, редуцированное слева, называется каноническим, если каждая его ФЗ имеет вид $X \to A$.*

Заметим, что каждая ФЗ канонического множества редуцирована и справа тоже, поскольку имеет один атрибут справа и $F$ неизбыточно. Если правый атрибут удалить,

то возникает ФЗ вида $X \to \varnothing$, который можно удалить из $F$, что противоречит неизбыточности. Таким образом, всякое каноническое множество редуцировано.

Пример

Пусть $F = \{A \to B,\ A \to C,\ A \to D,\ A \to E,\ BI \to J\}$ – каноническое покрытие для $G = \{A \to BCE,\ AB \to DE,\ BI \to J\}$.
Конец примера

***Определение.*** *Если F – покрытие множества G и F – каноническое множество, то F называется каноническим покрытием множества G.*

***Лемма.*** *Пусть F – редуцированное покрытие. Образуем G расщеплением каждой функциональной зависимости $X \to A_1 \ldots A_m \in F$ на $X \to A_1, \ldots, X \to A_m$. Тогда G – каноническое покрытие A. Обратно, если G – каноническое покрытие, а F образуется из G объединением функциональных зависимостей с совпадающими левыми частями, то F – редуцированное покрытие.*

### *Оптимальные покрытия*

***Определение.*** *Множество ФЗ F минимально, если оно содержит не больше ФЗ, чем любое эквивалентное множество ФЗ.*

Минимальное множество $F$ не может содержать избыточных ФЗ, то есть оно не избыточно.

Пример

Множество $G = \{A \to BC,\ B \to A,\ AD \to E,\ BD \to J\}$ не избыточно, но и не минимально, так как существует $F = \{A \to BC,\ B \to A,\ AD \to EJ\}$, $F \equiv G$, которое содержит меньше элементов. В данном случае $F$ минимально.
Конец примера

Определение минимального покрытия не конструктивно: не позволяет его непосредственно вычислить. Для вычисления минимального покрытия существует специальный подход, который в данном курсе лекций мы рассматривать не будем. Детально познакомиться с ним можно в [17].

***Определение.*** *Множество F оптимально, если не существует эквивалентного множества с меньшим числом атрибутных символов.*

Пример

Множество $F = \{EC \to D,\ AB \to E,\ E \to AB\}$—оптимальное покрытие для $G = \{ABC \to D,\ AB \to E,\ AB \to E,\ E \to AB\}$, которое редуцировано и минимально, но не оптимально.
Конец примера

***Лемма.*** *Если F – оптимальное множество ФЗ, то оно редуцировано и минимально.*

## *Уточнение нормальных форм*

### 2 нормальная форма

***Определение.*** *Множество Y для ФЗ $X \to Y$ из $F^+$ называется полностью зависимым от X, если $X \to Y$ полная ФЗ, то есть не $\exists X' \subseteq X \therefore (X' \to Y) \subseteq F^+$. В противном случае множество Y называется частично зависимым от X.*

***Определение.*** *Атрибут, не содержащийся ни в каком ключе схемы R, называется непервичным в R.*

***Определение.*** *Схема отношения R находится во второй НФ относительно множества ФЗ F, если для каждого атрибута A значения adom(A) атомарны и каждый атрибут, не содержащийся ни в каком ключе схемы R (он называется непервичным в R), полностью зависит от каждого ключа для R.*

Схема БД ***R*** имеет вторую НФ относительно *F*, если каждая схема отношения из ***R*** находится во второй НФ относительно *F*.

### 3 нормальная форма

***Определение.*** *Схема отношения R находится в третьей НФ относительно множества ФЗ F, если для каждого атрибута A значения adom(A) атомарны и ни каждый непервичный атрибутов не зависит транзитивно от ключа для R.*

***Утверждение.*** *Любая схема отношения, находящаяся в третьей НФ относительно некоторого множества ФЗ, находится во второй НФ относительно того же множества.*

Схема БД ***R*** находится в третьей НФ относительно *F*, если каждая схема отношения из ***R*** находится в третьей НФ относительно *F*.

## *Нормализация через декомпозицию*

Некоторую схему отношения *R*, не находящуюся в третьей НФ относительно множества ФЗ *F*, всегда можно разложить в схему БД, находящуюся в третьей НФ относительно *F*. Рассмотрим алгоритм, позволяющий выполнить это преобразование.

Предположим, что в $R = (S, \boldsymbol{K})$ существует транзитивная зависимость от ключа в *R*. Это означает, что есть ключ $K \in \boldsymbol{K}$, множество $Y \subseteq R$ и непервичный элемент $A \in R$, $A \notin KY$, такие, что $\{K \to Y, Y \to A\} \subseteq F$ и $Y \to K \notin F$. Определим новые отношения со схемами $R_1$ и $R_2$. Пусть $R_1 = R - A$, $R_2 = YA$. Пусть $\boldsymbol{K}$ – множество выделенных ключей для $R_1$, а $\{Y\}$ – соответственно, для $R_2$. Если $\exists K \in \boldsymbol{K}$ такой, что $A \in K$, определим для $R_1$ новое множество выделенных ключей $\boldsymbol{K}' = \boldsymbol{K} - \{K\} \cup \{K'\}$, где $K' = K - A$.

Можно доказать [17], что для любого $r(R)$, удовлетворяющего *F*, верно равенство $r = \pi_{R1}(r) \parallel \pi_{R2}(r)$. Таким образом, путём *декомпозиции* *R* на $R_1$ и $R_2$ транзитивная зависимость удалена, хотя этот результат и получен ценой увеличения количества отношений.

Если в $R_1$ или в $R_2$ остались транзитивные зависимости, можно выполнить декомпозицию ещё раз. Процесс конечен, так как на каждом шаге получаются всё более короткие схемы, а в схеме с двумя атрибутами транзитивной зависимости быть не может. Процесс декомпозиции можно ускорить, проверяя в $R - KY$ наличие других непервичных атрибутов, зависящих от *Y*. Такие атрибуты тоже зависят от *K*, и их можно уда-

лить одновременно с $A$. Пусть это атрибуты $A_1, A_2, \ldots, A_m$. Тогда $R_1 = R - A_1A_2\ldots A_m$, $R_2 = YA_1A_2\ldots A_m$, и отношение $R$ по-прежнему разлагается без потерь на $R_1$ и $R_2$.

Заметим, что получающаяся в результате декомпозиции схема базы даны неоднозначна, она зависит от порядка выбора атрибутов для декомпозиции, если количество транзитивных зависимостей больше одной.

К недостаткам алгоритма нормализации через декомпозицию можно отнести следующие:

1. Большая временная сложность, которая может не ограничиваться полиномиальной.
2. Число порождённых процессом схем отношений может быть больше, чем необходимо.
3. При декомпозиции могут возникать частичные зависимости от ключа, которые, в свою очередь, могут порождать больше схем, чем необходимо.
4. Для построенной схемы БД множество ФЗ $F$ может оказаться ненавязанным.
5. С помощью этого алгоритма можно получать схемы со «скрытыми» транзитивными зависимостями.

Пример, иллюстрирующий недостатки 2-5.

2. Задана схема $R=ABCDE$ и $F = \{ AB\to CDE, AC\to BDE, B\to C, C\to B, C\to D, B\to E \}$, ключи $\boldsymbol{K} = \{ AB, AC \}$. Здесь $D$ зависит транзитивно от $AB$ через $C$. Тогда

$$R_1 = ABCDE,\ R_2 = CD,\ \boldsymbol{K_1} = \{ AB, AC \},\ \boldsymbol{K_2} = \{ C \}.$$

Далее, в $R_1$ существует транзитивная зависимость $E$ от $AB$ через $B$. Преобразуем:

$$R_{11} = ABC,\ R_{12} = BE,\ \boldsymbol{K_{11}} = \{ AB, AC \},\ \boldsymbol{K_{12}} = \{ B \}.$$

Окончательно $\boldsymbol{R} = \{ R_{11}, R_{12}, R_2 \}$. Но можно обойтись следующей декомпозицией $R$:

$$R_1 = ABC,\ R_2 = BDE,\ \boldsymbol{K_1} = \{ AB, AC \},\ \boldsymbol{K_2} = \{ B \}.$$

3. Задана схема $R=ABCD$ и $F = \{ A\to BCD, C\to D \}$, ключи $\boldsymbol{K} = \{ A \}$. Атрибут $D$ зависит транзитивно от $A$ через $BC$. Тогда

$$R_1 = ABC,\ R_2 = BCD,\ \boldsymbol{K_1} = \{ A \},\ \boldsymbol{K_2} = \{ BC \}.$$

В $R_2$ $D$ частично зависит от ключа, следовательно, существует транзитивная зависимость $D$ от $BC$. Преобразуя, получаем три схемы $R_1, R_{21}, R_{22}$, хотя реально достаточно $R_1, R_{21}, R_{22}$.

4. Задана схема $R=ABCDE$ и $F = \{ A\to BCDE, CD\to E, EC\to B \}$, ключи $\boldsymbol{K} = \{ A \}$. Здесь $E$ зависит транзитивно от $A$ через $CD$. Тогда

$$R_1 = ABCD,\ R_2 = CDE,\ \boldsymbol{K_1} = \{ A \},\ \boldsymbol{K_2} = \{ CD \}.$$

Множество F ненавязано схеме $\boldsymbol{R} = \{ R_1, R_2 \}$ из-за невыводимости зависимости $EC\to B$ из зависимостей, приложимых к каждой из схем в отдельности.

5. Задана схема $R=ABCD$ и $F = \{ A\to B, B\to C \}$, ключи $\boldsymbol{K} = \{ AD \}$. Атрибут $B$ частично зависит от $AD$. Тогда

$$R_1 = ACD,\ R_2 = AB,\ \boldsymbol{K_1} = \{ AD \},\ \boldsymbol{K_2} = \{ A \}.$$

И $R_1$, и $R_2$ находятся в 3НФ, но существует «скрытая» транзитивная зависимость $C$ от $AD$ через $B$.

Конец примера

Существуют другие методы получения третьей нормальной формы, не порождающие проблем декомпозиции, в частности, алгоритм синтеза [17], но их рассмотрение выходит за пределы нашего курса.

## Литература


1. Астахова И.Ф., Толстобров А.П. SQL в примерах и задачах. Учебное пособие, 2002.
2. Атре Ш. Структурный подход к организации баз данных: Пер. с англ. – М.: Финансы и статистика, 1983. –317 с.
3. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем. – М.: Финансы и статистика, 2002.
4. Грабер М. Введение в SQL. – М.: «ЛОРИ», 1996. – 375 с.
5. Дейт К. Введение в системы баз данных, 8 изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2005.
6. Дейт К. Руководство по реляционной СУБД DB2: Пер. с англ. – М.: Финансы и статистика, 1988. –320 с.
7. Диго С.М. Базы данных: проектирование и использование: Учебник. М.: Финансы и статистика, 2005.
8. Жоголев Е.А. Введению в технологию программирования. Конспект лекций. – М.: «ДИАЛОГ-МГУ», 1994. – 112 с.
9. Каратыгин С.А., Тихонов А.Ф., Тихонова Л.Н. Visual FoxPro 7. – М.: Бином-Пресс, 2003.
10. Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002.
11. Кодд Э.Ф. Реляционная база данных: практическая основа эффективности. // Лекции лауреатов премии Тьюринга за первые двадцать лет 1966-1985. // Пер. с англ. – М.: Мир, 1993, с. 451-474.
12. Конноли Т. и др. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – М.: Addison-Wesley, 2001.
13. Крёнке Д. Теория и практика построения баз данных, 9 изд. – М.: Питер, 2005.
14. Лукин В.Н.,.Марасанов А.М, Ротанина М.В., Чернышов Л.Н / Под ред. Марасанова А.М.. Использование СУБД в прикладных программных системах. – М.: МАИ, 1996. – 56 с.
15. Маклаков С.В.. Erwin и BPwin. Case-средства разработки информационных систем. М., Диалог-МИФИ, 2-е изд., 2001.
16. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник. – М.: Финансы и статистика, 2004.
17. Мейер Д. Теория реляционных баз данных: Пер. с англ.: – М.: Мир 1987. – 608 с.
18. Озкарахан Э. Машины баз данных и управление базами данных: Пер с англ. – М.: Мир, 1989.
19. Тиори Т., Фрай Дж. Проектирование структур баз данных (в 2 томах): Пер. с англ. – М.: Мир, 1985.
20. Чен П. Модель «Сущность-Связь» – шаг к единому представлению данных. // СУБД, 1995, №3.