Управление данными
(конспект лекций)

План лекций

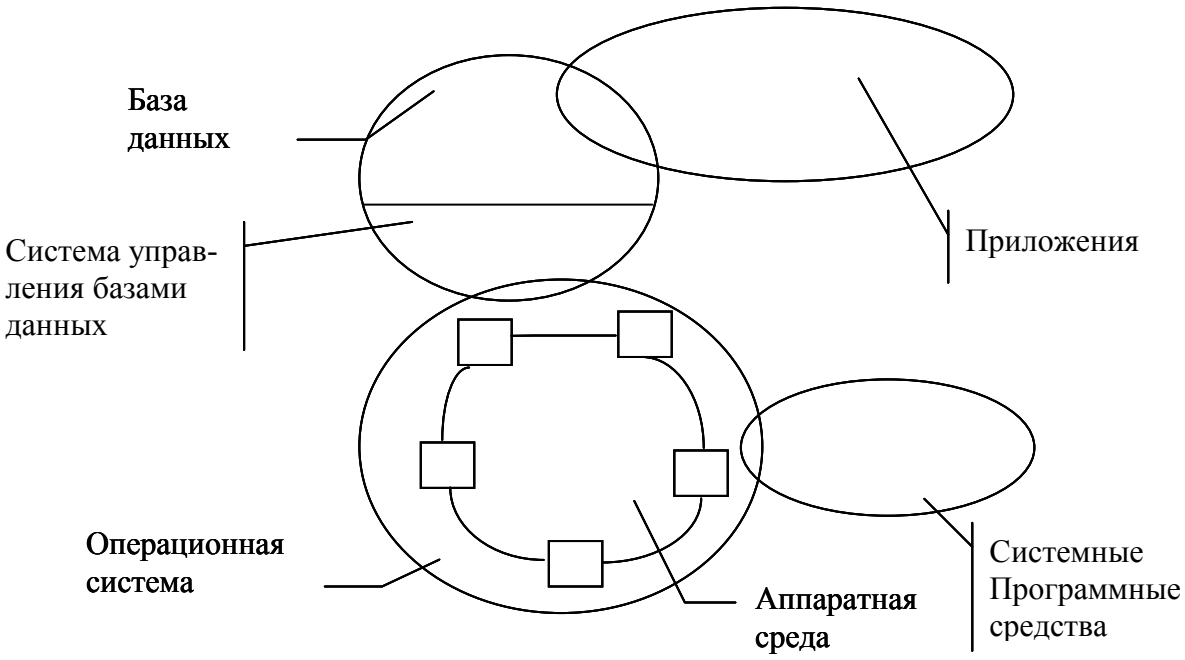
1. Основные понятия управления данными в вычислительных системах.
2. Реляционные системы.
   - 2.1.Понятие и структура реляционных систем.
     - 2.1.1. Реляционные модели данных.
       - 2.1.1.1. Понятие и уровни представления реляционных моделей.
       - 2.1.1.2. Нормализация реляционных моделей.
       - 2.1.1.3. Операции над отношениями.
     - 2.1.2. Проектирование реляционных моделей.
       - 2.1.2.1. Организация проектирования реляционных моделей.
       - 2.1.2.2. Эмпирическая схема проектирования модели данных.
       - 2.1.2.3. Синтез реляционных моделей с использованием множества функциональных зависимостей.
     - 2.1.3. Запросы в реляционных системах.
       - 2.1.3.1. Понятие и структура запроса.
       - 2.1.3.2. Способы формирования запроса.
       - 2.1.3.3. Язык SQL как основная форма описания запросов.
         - 2.1.3.3.1. Основные конструкции SQL
       - 2.1.3.4. Использование SQL для работы БД в файл-серверных и клиент серверных технологиях.
         - 2.1.3.4.1. Понятие файл и клиент серверные технологии.
         - 2.1.3.4.2. Реализация запросов в файл серверных технологиях.
         - 2.1.3.4.3. Реализация запросов в клиент серверных технологиях: представления, хранимые процедуры, функции.
     - 2.1.4. Проектирование приложений к реляционным базам данных.
       - 2.1.4.1. Структура приложений.
       - 2.1.4.2. Объектно - ориентированный подход к разработке приложений.
       - 2.1.4.3. Понятие объекта, метода, свойств.
       - 2.1.4.4. Классы, экземпляры и семейства.
       - 2.1.4.5. Иерархии классов.
     - 2.1.5. Проектирование объектно ориентированных приложений.
   - 2.2.Распределенная обработка данных.
     - 2.2.1. Понятие транзакции.
       - 2.2.1.1. Восстановление системы. Двухфазная фиксация.
       - 2.2.1.2. Параллелизм.
       - 2.2.1.3. Безопасность.

Основные понятия управления данными в вычислительных системах.

Дисциплина - управление данными изучает процессы, связанные с автоматизацией сбора, накоплением, хранением и использованием информации появляющейся в результате деятельности человека.

Информацию можно определить как набор фактов, сведений, воспринимаемых человеком и устраняющими у него ранее существовавшую неопределенность. Синонимом информации является понятие «данные», которые можно определить как набор фактов, сведений, представленных в закодированном виде, которые могут храниться, передаваться и обрабатываться человеком или машиной.

Термин – управление данными охватывает такие области знаний как структуризация и моделирование данных, методы обработки данных, организацию использования данных в различных аппаратных средах. В настоящее время одной из основных форм организации данных в вычислительных средах являются базы данных. В общем случае место БД в вычислительной среде можно отразить следующей схемой:



Аппаратная среда это совокупность вычислительных и сетевых средств организованных в функциональные структуры. Управление работой аппаратных средств осуществляется с помощью специальных программных средств, называемых операционными системами. Кроме того, в целях поддержания нормальной работы системы «аппаратная среда – операционная система», называемой вычислительной сис-

темой используются системные программные средства. Накопление и хранение данных поддерживается системами управления данными, с помощью которых создаются базы данных. Базы данных могут быть использованы различными приложениями для решения задач пользователей. Такая схема использования вычислительной среды для решения информационных задач в настоящее время наиболее распространенной.

Идея создания баз данных базировалась на следующих исторических обстоятельствах. В середине 70-х годов XX века появляются вычислительные средства с достаточно большой памятью и быстродействием для использования при решении экономических задач. Одним из основных свойств экономической информации является ее массовость, т.е. возникновение за короткий период времени больших объемов информации и необходимость ее длительного хранения. В процессе решения экономических задач проявились две крупные проблемы, осознание которых привело к созданию централизованных информационных структур, называемых базами данных.

Первая состояла в том, что вся информация для решения задач хранилась в файлах. При этом занесение данных в файлы, выборка и обработка их производилась с помощью программ, создание которых требовало значительных трудозатрат. Любые дополнения, изменения структуры данных требовали изменения программ. В итоге, стоимостные, временные и трудовые затраты не соответствовали ценности полученной дополнительной информации.

Вторая была связана с тем, что разные подразделения одной организации, используя пересекающуюся информацию, хранили их в различных файлах. В результате непоследовательного обновления одних и тех же данных в разных файлах схожие по смыслу результаты различных подразделений имели разные значения. Это приводило к недоверию использования вычислительной техники при решении экономических задач.

В итоге, обществом была осознана необходимость централизованного управления данными и появилось понятие банка или базы данных (БД). БД можно определить как взаимосвязанную совокупность данных, хранящуюся в электронном виде и предназначенную для коллективного использования. Появление БД привело к возникновению новых следующих понятий:

- Системы управления базами данных (СУБД);
- Администратор данных (АД) и администратор базы данных (АБД);

СУБД представляет собой совокупность программных средств, предназначенных для организации хранения данных в электронном виде и доступа к ним.

Администратор данных – это человек, отвечающий за стратегию и политику принятия решений, связанных с данными объекта управления. Администратор базы данных – это человек или группа людей, обеспечивающих проектирование структуры БД, управление созданием базы и поддержанием ее работоспособности, обучение и консультации пользователей. В основе БД лежит модель данных.

Модели данных

Основными понятиями, использующимися при работе с данными, являются: элемент данных, логическая запись и файл. Элемент данных (ЭД) это наименьшая, имеющая смысл, единица данных. Каждый ЭД имеет имя и набор свойств. К свойствам ЭД относятся тип данного и его размер. Типы данных – числовой, символьный, логический, тип дата и т.д. связаны с представлением данных в памяти компьютера. Размер характеризует место в памяти, занимаемое данным. Элементы данных, связанные между собой по смыслу, могут объединяться в группы, называемые логическими записями. Поименованная совокупность логических записей, размещенная на внешнем запоминающем устройстве, называется файлом.

База данных представляет собой структурированную совокупность элементов данных, объединенных в логические записи, и связей между ними. Связи между ЭД или логическими записями отражают обязательные соответствия между ними. Для отображения состава логических записей базы данных и связей между ними используют схемы различных видов, которые принято называть моделями данных.

Уровни представления данных

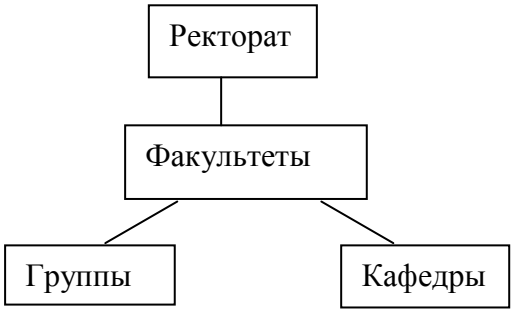
Понятия элемент данных, логическая запись и файл относятся к данным, хранимым в электронном виде. При работе с моделями данных используют другие понятия. В соответствии с ними модель данных состоит из: основных элементарных данных предметной области, называемых объектами или сущностями; элементарных данных, описывающих сущности и называемых атрибутами; ассоциации между экземплярами элементарных данных, называемых связями.

В отношении "объект-атрибут-связь" пользователь описывает интересующие его элементы предметной области с помощью объектов. Затем определяются свойства объектов путем использования атрибутов. Для описания соответствия между объектами используются связи. В качестве объекта может выступать личность, место и т.д. С объектом связаны два понятия: тип и экземпляр объекта. Понятие тип объекта относится к набору однородных предметов или вещей, выступающему как единое целое. Тип объекта - это концепт. Экземпляр объекта относится к конкретной вещи. Например типом объекта может быть СТУДЕНТ,а экземпляром-Петров Н.С.,Харламов А.И. Атрибутом называется поименованная характеристика объекта. Атрибут это элемент данных, с помощью которого определяются свойства объекта. Например, ВОЗРАСТ объекта СТУДЕНТ. В реальном мире все явления и предметы взаимодействуют друг с другом, т.е. одни объекты связаны с другими. Под связями понимаются ассоциации (соответствия) между одинаковыми или различными типами объектов. Для описания объекта атрибуты могут быть объединены в логические записи.

Выделяют три уровня абстракции для определения модели БД: концептуальный (с позиций администратора предприятия), уровень реализации или логический уровень (с позиций прикладного программиста) и физический (с позиций системного программиста или системного аналитика).

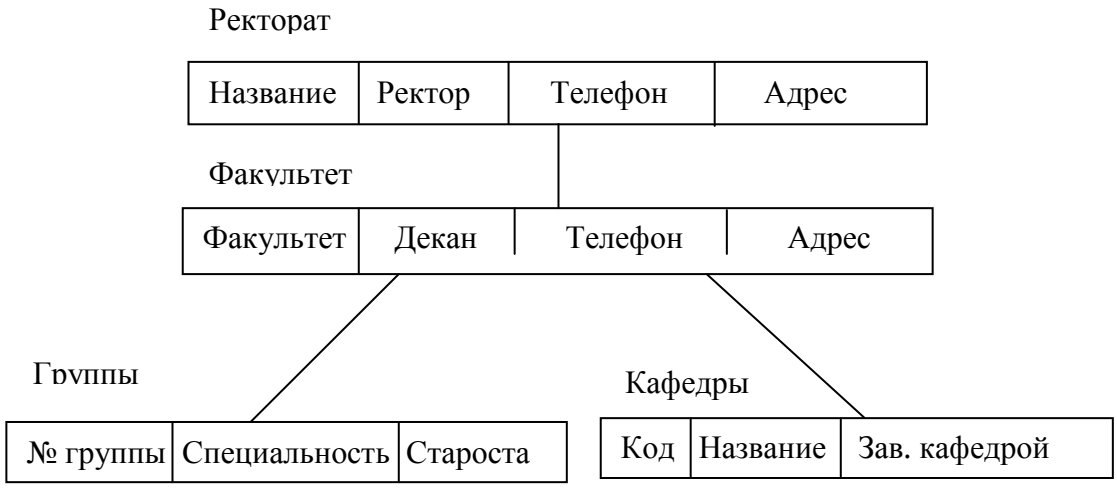
Концептуальный уровень. Концептуальный уровень предполагает изображение модели в виде поименованных объектов и связей между ними.

Пример.



Логический уровень. Логический уровень состоит из логических записей, составляющих их атрибутов и связей между ними. Логическая запись состоит из элементов данных – атрибутов и определяет характеристики объекта.

Пример.



Физический уровень или физическое представление так же характеризуется записями и связями между ними. Однако записи организованы в соответствии с физическими особенностями носителей, на которых они хранятся. Связи между хранимыми записями осуществляются путём их группировки и хранения в одном месте носителя или включением в них дополнительного элемента, называемого указателем. Физический уровень модели определяется и используется СУБД.

Связи в моделях

Говорят, что между объектами или атрибутами существует связь, если между экземплярами различных объектов (атрибутов) можно установить закономерность соответствия. Различают два основных типа связей – один к одному (1:1) и один ко многим (1:M).

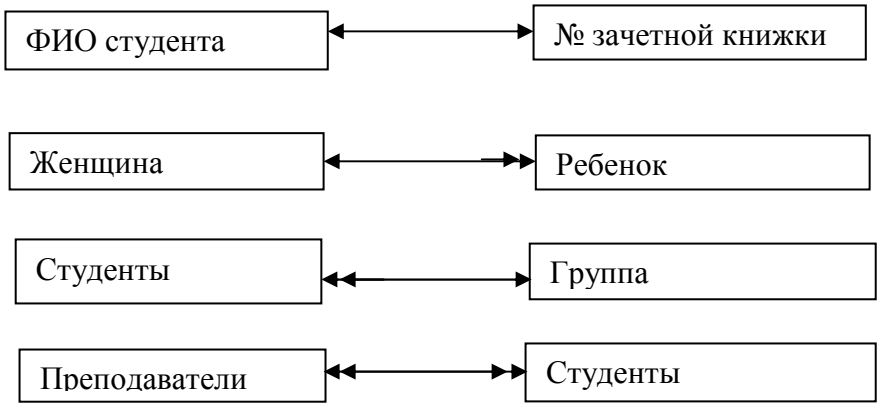
Определения:

Между элементами A и B определена связь один к одному, если в каждый момент времени каждому элементу A соответствует только один ассоциированный с ним элемент B.

Между элементами A и B определена связь один ко многим, если в каждый момент времени каждому элементу A соответствует ноль, один или несколько ассоциированных с ним элементов B.

Связи между объектами (атрибутами) могут существовать в обоих направлениях, т.е. возможны четыре варианта связей: 1:1, 1:M, M:1, M:M.

Примеры.

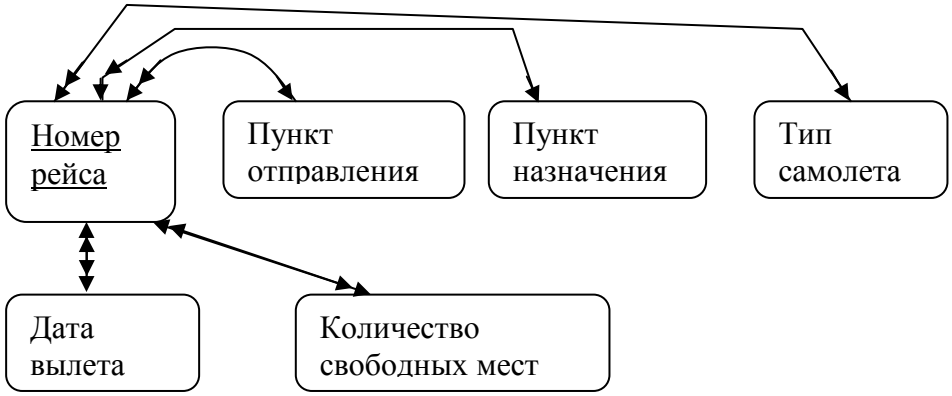


Данная классификация связей исчерпывает возможные варианты и позволяет строить модели со строгим упорядочением отношений между данными.

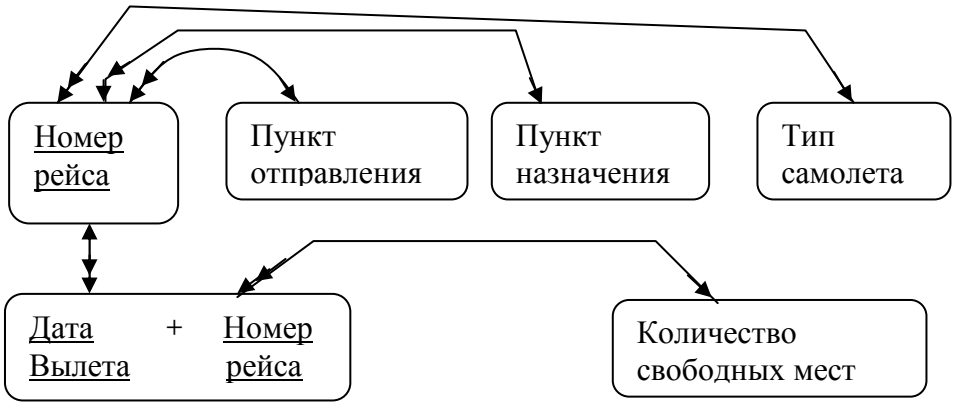
Построение логических записей

Логическая запись описывает объект и его свойства и состоит из совокупности взаимосвязанных атрибутов. Причем один или несколько атрибутов отражают суть объекта, отличающую один экземпляр объекта от другого. Эти атрибуты называются ключом. Значения ключа являются уникальными для каждого типа записей. Все остальные атрибуты логической записи связаны с ключом. Причем допускаются связи 1:1 или M:1 со стороны ключа. Данные принципы создают формальную основу для образования логической записи. Рассмотрим следующий пример.

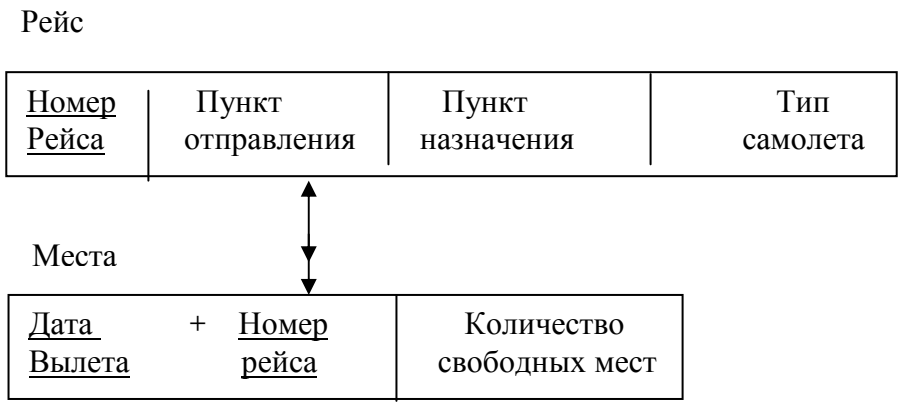
Пусть требуется разработать модель данных системы резервирования авиабилетов. Причем известны следующие элементы данных.



Поскольку все элементы данных в примере относятся к рейсам самолетов, то ключом логической записи логично было бы выбрать номер рейса. Действительно этот атрибут обладает основным свойством ключа – каждый рейс имеет свой уникальный номер. Связи между элементами определяются ролью каждого из них по отношению к ключу. Так, из пункта отправления могут отправляться много рейсов, но каждый рейс имеет только один пункт отправления, поэтому связь между ними со стороны ключа будет M:1. Следовательно, эти атрибуты можно объединить в логическую запись. То же касается атрибутов «Пункт назначения», «Тип самолета». Между ключом и атрибутом «Дата вылета» существует связь M:M, так как один и тот рейс может вылетать в разные даты, а в одну и ту же дату могут вылетать разные рейсы, то есть эти атрибуты нельзя объединить в логическую запись. То же касается и атрибута «Количество свободных мест». Однако эти два атрибута несут существенную информационную нагрузку и должны быть включены в модель. Чтобы решить эту проблему, можно преобразовать схему так.



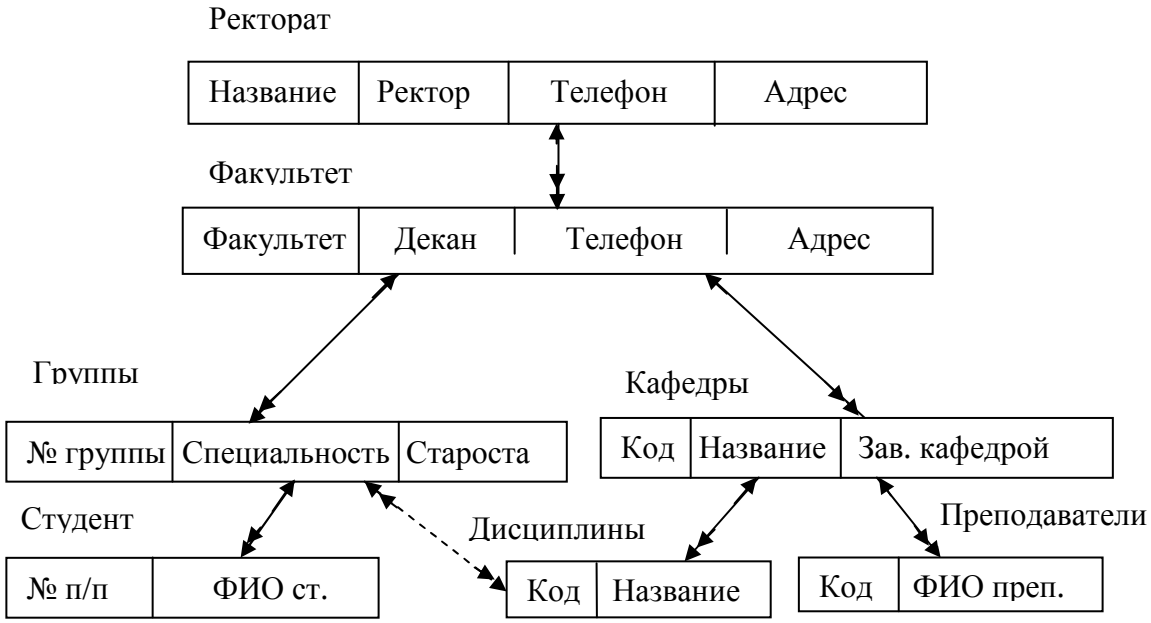
В итоге образуются две логические записи. Причем первая соотносится со второй как 1:M. Чтобы не указывать связи между элементами данных, проще изобразить полученную модель следующим образом.



Модели данных различаются по характеру связей между логическими записями. В зависимости от этого параметра различают иерархические, сетевые и реляционные модели данных.

Иерархические модели данных

В иерархической модели все логические записи распределены по уровням. Причем каждая логическая запись связана с 0, 1 или несколькими записями нижнего уровня и одной записью верхнего, если он есть. Причем используются связи 1:M сверху вниз. В качестве примера приведем фрагмент модели ВУЗа. Связи, относящиеся к данной модели, показаны сплошными стрелками.



Данный тип моделей отличает простота построения. С помощью таких моделей хорошо описываются иерархически структурированные системы. В то же время необходимо помнить, что основное назначение модели данных, описание структуры базы данных. При этом БД используется для получения ответов на запросы. И критерием качества модели может служить ее возможность получать выборки данных

на различные запросы. Под запросом понимается описание требований к выбираемым из базы данным. Рассмотрим реализацию трех запросов с использованием модели данных приведенного примера.

Найти группу, где учится студент А.

Найти факультет, где учится студент А.

Найти дисциплины, которые изучает студент А.

Для получения ответа на первый запрос необходимо найти запись о студенте среди записей «Студент» и далее найти связанную с ним запись из записей «Группа».

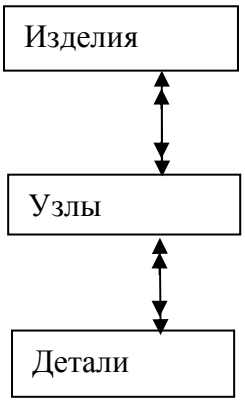
Для получения ответа на второй запрос необходимо найти запись о студенте среди записей «Студент» , найти связанную с ним запись из записей «Группа» и далее связанную с последней запись из группы записей «Факультет».

При ответе на третий запрос, можно найти группу, где учится студент. Искать факультет нет смысла, так как между записями о факультете и записями о дисциплинах невозможно установить однозначной связи. Ответ на запрос может быть получен, если ввести дополнительную связь между записями «Группа» и «Дисциплины». Причем это должна быть связь М:М, так как в каждой группе одновременно читаются несколько дисциплин, а каждая дисциплина может читаться в нескольких группах. Эта связь указана в модели пунктирной линией. Полученную в результате появления этой связи модель, нельзя назвать иерархической и оно переходит в класс сетевых.

Сетевые модели данных.

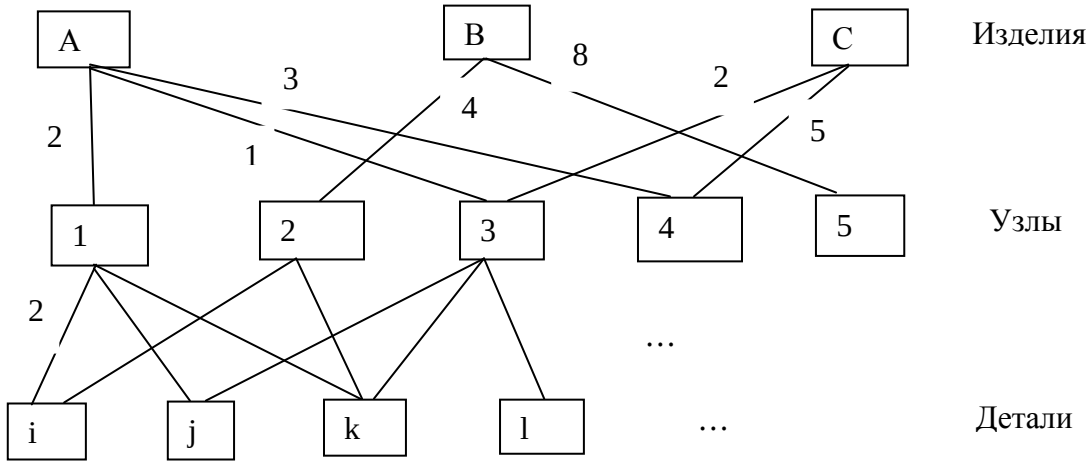
В сетевых моделях связи могут устанавливаться произвольным образом. Кроме того в них допускаются связи М:М. Однако этот тип связей несет неопределенность соответствия записей. Этот тип связей показывает только характер соответствия записей, но не может быть использован для получения ответов на запросы. Проблемы, связанные с применением этого типа связей рассмотрим на следующем примере.

На машиностроительных, мебельных производствах часто решается задача разузлования, концептуальная модель БД которой имеет следующий вид.



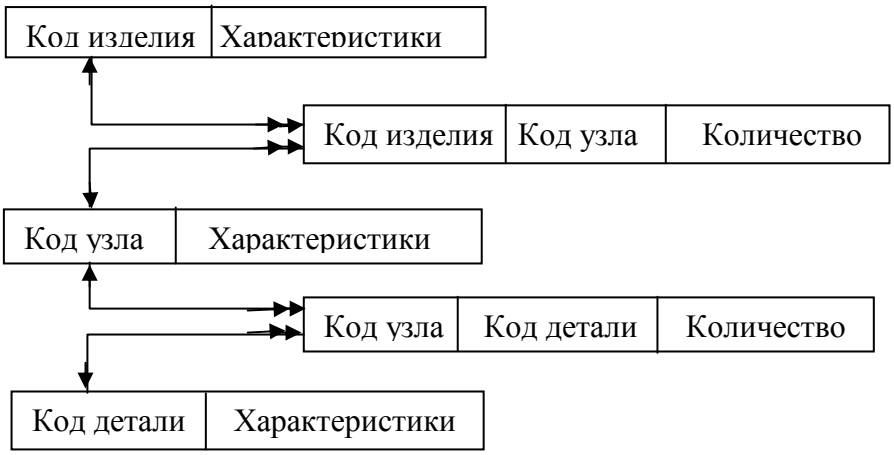
Смысл модели состоит в следующем. Предприятие выпускает некоторые изделия, которые состоят из узлов. В свою очередь узлы состоят из деталей. При этом в каждое изделие входят несколько узлов, в то же время каждый узел может входить в несколько изделий. Такая же взаимосвязь между узлами и деталями.

Для того, чтобы представить проблемы этой модели, рассмотрим поэкземплярную схему базы данных.



Связи между экземплярами показывают: какие узлы входят в изделия и из каких деталей они состоят. Цифры рядом со связями показывают количество вхождений одних элементов в другие. Они называются данными пересечения. В рассматриваемой модели размещение данных пересечения является проблемой. Действительно, если поместить данные пересечения в запись об изделии, то для каждого экземпляра записи нужно будет иметь количество элементов (атрибутов) соответствующее числу узлов, входящих в изделие. Поскольку в каждое изделие входит разное количество узлов, то атрибутов в них должно быть переменное количество. Но структура записи не может быть переменной. Если оставить размещение данных пересечения в записях об изделии, то количество атрибутов под них надо брать по максимуму, а это ведет к большому перерасходу памяти. Такая же ситуация возникает, если попытаться разместить данные пересечения в записях об узлах.

Для решения этой проблемы необходимо преобразовать исходную модель к следующему виду.



В ней введены дополнительные записи, связанные с существовавшими связями М:1. В соответствии с этой моделью каждой записи об изделиях соответствует столько записей с данными пересечения «изделие-узел», сколько узлов в нем содержится. Каждой записи об узлах соответствует столько записей с данными пересечения «изделие-узел», во сколько изделий входит данный узел. Та же логика и в связях между узлами и деталями.

Таким образом из исходной модели удалены связи М:М и найдена возможность размещения данных пересечения с минимальной избыточностью хранения данных.

Реляционные модели данных

Основные понятия

Сетевые модели удобны на начальном этапе разработки модели для базы данных, так как содержат минимум ограничений и удобны для проектировщика. Однако они мало формализованы и для них трудно создать СУБД , позволяющую эффективно обслуживать их. Поэтому в настоящее время наибольшее распространение получили реляционные модели данных. Основу реляционной модели составляют таблицы или *отношения*. Отношение является полным аналогом логической записи. Под отношением понимают совокупность логически связанных между собой данных структурированных по строкам и столбцам. Строки отношения принято называть *кортежами*, а столбцы *доменами*. В дальнейшем эти понятия будут рассмотрены более точно. Под связью между таблицами (отношениями) понимается соответствие между значениями одинаковых доменов отношений. В реляционных моделях допускаются связи 1:М, М:1 и 1:1. Принцип установления связи между двумя отношениями можно проследить на следующем примере.

Пусть задано отношение, содержащее данные о поставке продукции некоторыми поставщиками.

№ заказа	Поставщик	Дата	Товар	Характеристики	Цена
121	А…	17.04.03	П…	О……	256.00
121	А…	17.04.03	Р…	Л…	4598.00
121	А…	17.04.03	Е…	Н…..	785.00
122	В…	18.04.03	П…	О……	256.00

В этой таблице для каждого товара приходится повторять сведения о заказе и поставщике, что приводит к излишнему дублированию данных. Разобьем эту таблицу на две таблицы следующего вида:

№ заказа	Поставщик	Дата
121	А…	17.04.03
122	В…	18.04.03

№ заказа	Товар	Характеристики	Цена
121	П…	О……	256.00
121	Р…	Л…	4598.00
121	Е…	Н…..	785.00
122	П…	О……	256.00

В первой из них приводятся неповторяющиеся сведения о заказах. При этом данные в домене «№ заказа» по смыслу не повторяются, т.е. этот домен обладает основным признаком ключа. Во второй таблице введен дополнительный столбец «№ заказа». Он необходим для установления связи со строками первой таблицы. Таким образом, для установления связи между двумя отношениями необходимо наличие в каждом из них одинаковых доменов, т.е. доменов имеющих одинаковый тип данных и данные соответствующие друг другу. Совпадение названий доменов необязательно. В данном примере в первом отношении «№ заказа» является ключом, и следовательно его значения не повторяются. Во втором отношении ключом является совокупность атрибутов «№ заказа», «Товар», поэтому допускается повторение номера заказа. Следовательно, между первым и вторым отношениями существует связь 1:M по домену «№ заказа».

Реляционные модели принято записывать в строчном виде. Для каждого отношения сначала указывается его имя, затем в скобках перечисляются атрибуты, ключи подчеркиваются. Для приведенного примера модель будет выглядеть так.

ЗАКАЗ(№ заказа, Поставщик, Дата)

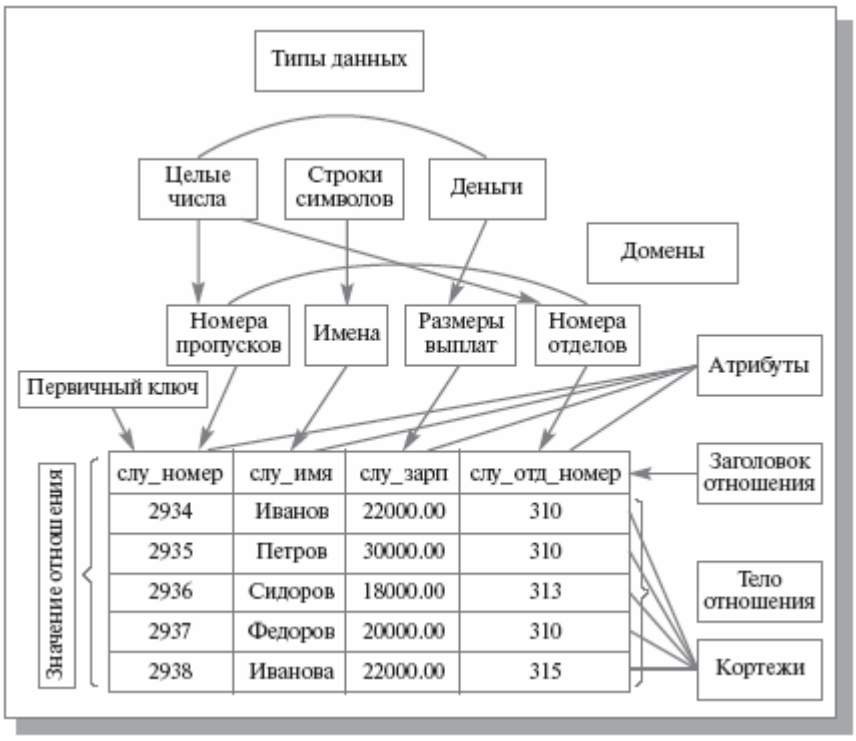
ТОВАРЫ(№ заказа, Товар, Характеристики, Цена)

Расширение основных понятий реляционных моделей

Рассмотренные понятия реляционных моделей данных носят обобщённый характер. Для дальнейшего изучения потребуется более подробное описание основных понятий. Материалы этого параграфа были заимствованы из лекций С.Д. Кузнецова [4].

Реляционный подход к организации баз данных был заложен в конце 1960-х гг. Эдгаром Коддом. В последние десятилетия этот подход является наиболее распространенным. Достоинствами реляционного подхода принято считать следующие свойства: реляционный подход основывается на небольшом числе интуитивно понятных абстракций, на основе которых возможно простое моделирование наиболее распространенных предметных областей; эти абстракции могут быть точно и формально определены; теоретическим базисом реляционного подхода к организации баз данных служит простой и мощный математический аппарат теории множеств и математической логики; реляционный подход обеспечивает возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти. Компьютерный мир далеко не сразу признал реляционные системы. В 70-е года прошлого века, когда уже были получены почти все основные теоретические результаты и даже существовали первые прототипы реляционных СУБД, многие авторитетные специалисты отрицали возможность добиться эффективной реализации таких систем. Однако преимущества реляционного подхода и развитие методов и алгоритмов организации и управления реляционными базами данных привели к тому, что к концу 80-х годов реляционные системы заняли на мировом рынке СУБД доминирующее положение.

Пояснением к дальнейшему описанию понятий является следующий пример.



Тип данных

Значения данных, хранимые в реляционной базе данных, являются типизированными, т. е. известен тип каждого хранимого значения. Понятие типа данных в реляционной модели данных полностью соответствует понятию типа данных в языках программирования. Напомним, что традиционное (нестрогое) определение ***типа данных*** состоит из трех основных компонентов: определение множества значений данного типа; определение набора операций, применимых к значениям типа; определение способа внешнего представления значений типа (литералов).

Обычно в современных реляционных базах данных допускается хранение символьных, числовых данных (точных и приблизительных), специализированных числовых данных (таких, как «деньги»), а также специальных «темпоральных» данных (дата, время, временной интервал). Кроме того, в реляционных системах поддерживается возможность определения пользователями собственных типов данных.

В примере мы имеем дело с данными трех типов: строки символов, целые числа и «деньги».

Домен

Понятие домена более специфично для баз данных, хотя и имеются аналогии с подтипами в некоторых языках программирования (более того, в своем«Третьем манифесте» Кристофер Дейт и Хью Дарвен вообще ликвидируют различие между доменом и типом данных). В общем виде домен определяется путем задания некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу этого типа данных (ограничения домена). Элемент данных является элементом домена в том и только в том случае, если вычисление этого логического выражения дает результат истина. С каждым доменом связывается имя, уникальное среди имен всех доменов соответствующей базы данных.

Наиболее правильной интуитивной трактовкой понятия домена является его восприятие как допустимого потенциального, ограниченного подмножества значений данного типа. Например, домен ИМЕНА в нашем примере определен на базовом типе символьных строк, но в число его значений могут входить только те строки, которые могут представлять имена (в частности, для возможности представления русских имен такие строки не могут начинаться с мягкого или твердого знака и не могут быть длиннее, например, 20 символов). Если некоторый атрибут отношения определяется на некотором домене (в примере атрибут СЛУ_ИМЯ определяется на домене ИМЕНА), то в дальнейшем ограничение домена играет роль ограничения целостности, накладываемого на значения этого атрибута.

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. В нашем примере значения доменов НОМЕРА ПРОПУСКОВ и НОМЕРА ОТДЕЛОВ относятся к типу целых чисел, но не являются сравнимыми (допускать их сравнение было бы бессмысленно).

Заголовок отношения, кортеж, тело отношения, значение отношения, переменная отношения

Понятие отношения является наиболее фундаментальным в реляционном подходе к организации баз данных, поскольку n-арное отношение является единственной родовой структурой данных, хранящихся в реляционной базе данных. Это отражено и в общем названии подхода – термин реляционный (relational) происходит от relation (отношение). Однако сам термин отношение является исключительно неточным, поскольку, говоря про любые сохраняемые данные, мы должны иметь в виду тип этих данных, значения этого типа и переменные, в которых сохраняются значения. Соответственно, для уточнения термина отношение выделяются понятия заголовка отношения, значения отношения и переменной отношения. Кроме того, нам потребуется вспомогательное понятие кортежа.

Итак, заголовком (или схемой) отношения **r (Hr)** называется конечное множество упорядоченных пар вида **<A, T>**, где **A** называется именем атрибута, а **T** обозначает имя некоторого базового типа или ранее определенного домена. По определению требуется, чтобы все имена атрибутов в заголовке отношения были различны. В примере заголовком отношения СЛУЖАЩИЕ является множество пар ***{<слу_номер, номера_пропусков>, <слу_имя, имена>, <слу_зарп, размеры_выплат>, <слу_отд_номер, номера_отделов>}***.

Если все атрибуты заголовка отношения определены на разных доменах, то, чтобы не плодить лишних имен, разумно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это всего лишь удобный способ именования, который не устраняет различия между понятиями домена и атрибута).

Кортежем **tr**, соответствующим заголовку **Hr**, называется множество упорядоченных триплетов вида **<A, T, v>**, по одному такому триплету для каждого атрибута в **Hr**. Третий элемент – **v** – триплета **<A, T, v>** должен являться допустимым значением **типа данных** или домена **T**. Заголовку отношения СЛУЖАЩИЕ соответствуют, например, следующие кортежи: ***{<слу_номер, номера_пропусков, 2934>, <слу_имя, имена, Иванов>, <слу_зарп, размеры_выплат, 22.000>, <слу_отд_номер, номера_отделов, 310>}, {<слу_номер, номера_пропусков, 2940>, <слу_имя, имена, Кузнецов>, <слу_зарп, размеры_выплат, 35.000>, <слу_отд_номер, номера_отделов, 320>}***.

Телом **Br** отношения **r** называется произвольное множество кортежей **tr**. Одно из возможных тел отношения СЛУЖАЩИЕ показано на примере. Заметим, что в общем случае, могут существовать такие кортежи tr, которые соответствуют **Hr**, но не входят в **Br**.

Значением **Vr** отношения **r** называется пара множеств **Hr** и **Br**. Одно из допустимых значений отношения СЛУЖАЩИЕ показано на примере.

В изменчивой реляционной базе данных хранятся отношения, значения которых изменяются во времени. *Переменной* **VARr** называется именованный контейнер, который может содержать любое допустимое значение **Vr**. Естественно, что

при определении любой **VARr** требуется указывать соответствующий заголовок отношения **Hr**.

Здесь стоит подчеркнуть, что любая принятая на практике операция обновления базы данных – **INSERT** (вставка кортежа в переменную отношения),**DELETE** (удаление кортежа из значения-отношения переменной отношения) и **UPDATE** (модификация кортежа значения-отношения переменной отношения) – с модельной точки зрения является операцией присваивания переменной отношения некоторого нового значения-отношения. Это совсем не означает, что перечисленные операции должны выполняться именно таким образом в СУБД: главное, чтобы результат операций соответствовал этой модельной семантике.

Заметим, что в дальнейшем в тех случаях, когда точный смысл термина понятен из контекста, мы будем использовать термин отношение как в смысле значение отношения, так и в смысле переменная отношения.

По определению, степенью, или «арностью», заголовка отношения, кортежа, соответствующего этому заголовку, тела отношения, значения отношения и переменной отношения является мощность заголовка отношения. Например, степень отношения СЛУЖАЩИЕ равна четырем, т. е. оно является 4-арным (кватернарным).

При приведенных определениях разумно считать схемой реляционной базы данных набор пар **<имя_VARr, Hr>**, включающий имена и заголовки всех переменных отношения, которые определены в базе данных. Реляционная база данных – это набор пар **<VARr, Hr>** (конечно, каждая переменная отношения в любой момент времени содержит некоторое значение-отношение, в частности, пустое).

Заметим, что в классических реляционных базах данных после определения схемы базы данных могли изменяться только значения переменных отношений. Однако теперь в большинстве реализаций допускается и изменение схемы базы данных: определение новых и изменение заголовков существующих переменных отношений. Это принято называть *эволюцией* схемы базы данных.

Преобразование сетевых моделей в реляционные

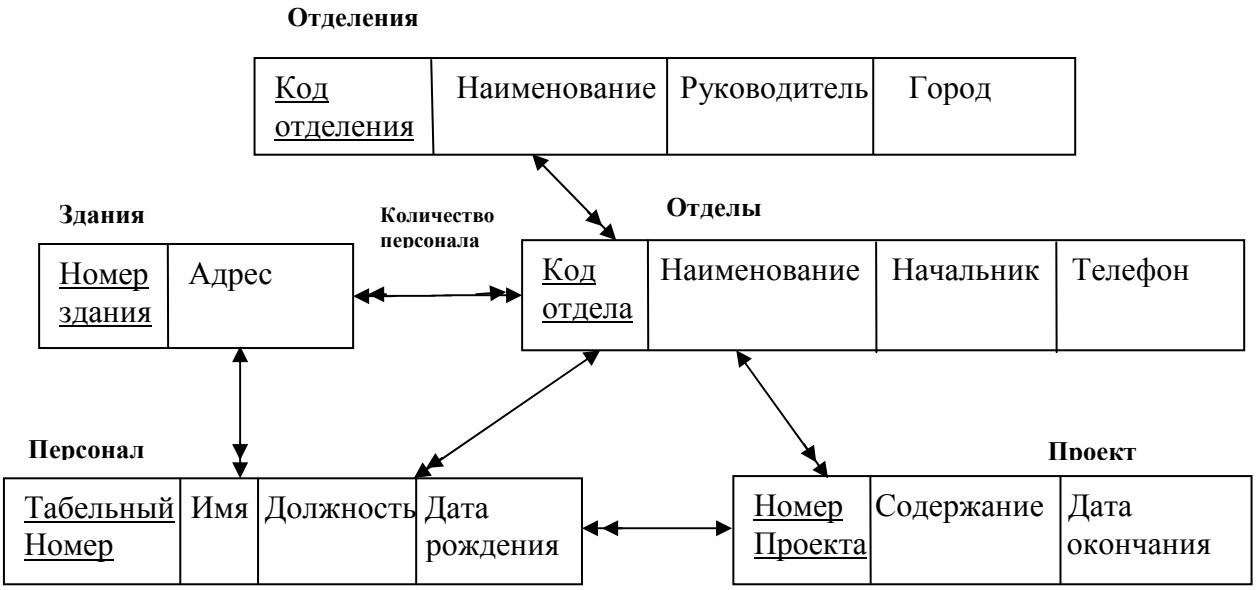
Сетевые модели имеют минимум ограничений и позволяют максимально полно отразить связи в информационной среде проектируемого объекта. Однако для этих моделей почти не существует СУБД из-за плохой формализации моделей и сложности построения программного обеспечения. Поэтому часто используют следующий подход к проектированию моделей баз данных.:

Разрабатывают сетевую модель базы данных объекта, с указанием всех возможных связей между структурами данных (логическими записями).

Преобразуют сетевую модель к реляционной.

Процесс преобразования моделей удобно рассмотреть на следующем примере.

Пусть дана сетевая модель некой крупной фирмы занимающейся разработкой проектов и состоящей из отделений, отделов, расположенных в различных зданиях города.



Стрелки показывают следующие взаимосвязи объектов. В отделение входит несколько отделов. Отдел может располагаться в нескольких зданиях. В то же время, в здании может располагаться несколько отделов. В здании работает много служащих. В отделе работает много служащих и каждый служащий работает только в одном отделе. Отдел ведет несколько проектов и каждый проект ведется только одним отделом. Каждый служащий работает только над одним проектом и над проектом работает много служащих. Данная модель является сетевой и содержит связи M:M. Для того, чтобы перейти от этой модели к реляционной, заменим стрелки обозначающие связи на связи в смысле реляционных моделей, т.е. будем рассматривать логические записи как отношения, а для связи отношений введем в отношения ассоциированные атрибуты. В результате модель примет следующий вид.

Отделения

Код отделения	Наименование	Руководитель	Город

Здания

Номер здания	Адрес

Отделы

Код отдела	Код отделения	Наименование	Начальник	Телефон

Размещение

Номер Здания	Код отдела	Количество персонала

Проект

Номер проекта	Код отдела	Содержание	Дата окончания

Персонал

Табельный Номер	Код отдела	Номер проекта	Имя	Должность	Дата рождения

Как видно из полученной модели связующие атрибуты добавлялись в отношения стоящие в исходной модели по стрелке со стороны М. Для устранения связи М:М и размещения данных пересечения (количество персонала отделов, работающих в зданиях) создано отношение **Размещение** с составным ключом. Записанная по правилам записи реляционных моделей полученная схема примет следующий вид.

ОТДЕЛЕНИЯ(Код отделения, Наименование, Руководитель, Город)

ЗДАНИЯ(Номер здания, Адрес)

ОТДЕЛЫ(Код отдела, Код отделения, Наименование, Начальник, Телефон)

РАЗМЕЩЕНИЕ(Номер здания, Код отдела, Количество персонала)

ПРОЕКТ(Номер проекта, Код отдела, Содержание, Дата окончания)

ПЕРСОНАЛ(Табельный номер, Код отдела, Номер проекта, Имя, Должность, Дата рождения)

Говорят, что отношения реляционной модели представлены в *первой нормальной форме*, если каждое из них включает набор атомарных (неделимых) атрибутов с выделенным ключом.

Функциональная зависимость атрибутов

Ранее были рассмотрены принципы объединения элементов данных в логические записи и показано, что все элементы данных в логической записи связаны меж-

ду собой. В реляционных моделях существует аналогичное понятие функциональной зависимости атрибутов. Говорят, что атрибут B *функционально зависит* от атрибута A, если в каждый момент времени каждому значению атрибута A соответствует только одно, связанное с ним значение атрибута B и обозначают A → B. Это же выражение можно прочитать как A *функционально определяет* B.

В следующем примере отношения

ПЕРСОНАЛ(Табельный номер, ФИО, Должность, Номер проекта, Дата окончания)

можно определить следующие функциональные зависимости:

Табельный номер →ФИО

Табельный номер →Должность

Табельный номер →Номер проекта

Номер проекта → Дата окончания

Действительно, каждому табельному номеру соответствует только одна фамилия, имя, отчество. Каждому проекту соответствует только одна дата окончания.

В общем случае определить функциональную зависимость атрибутов на множестве атрибутов и их значений можно следующим образом. Объединить атрибуты и их значения в одно отношение, отсортировать кортежи отношения по значению определяющего атрибута (атрибутов) и выяснить для каждого атрибута, зависимость которого необходимо найти, наличие повторяющихся значений. Если таковые есть, то атрибут не зависит от определяющего. В противном случае зависимость существует.

Например, пусть имеется множество атрибутов A, B, C и их значения, собранные в отношении R.

A	B	C
M	**2**	**4**
N	5	8
M	2	6
L	6	9
N	5	8
L	6	7

Попытаемся установить следующие зависимости A → B и A → C . Для этого отсортируем отношение по домену A:

A	B	C
M	2	4
M	2	6
N	5	8
N	5	8
L	6	9
L	6	7

Из таблицы видно, что каждому значению атрибута A соответствует только одно значение атрибута B и разные значения атрибута C. Следовательно A функционально определяет B, а C не зависит от A.

В отношении могут существовать функциональные зависимости между любыми атрибутами. Но при построении реляционных моделей представляют интерес только зависимости между ключом и остальными атрибутами отношения и транзитивные зависимости. Под транзитивной зависимостью понимают зависимость одного атрибута от другого через третий атрибут. Так, в рассмотренном выше примере имеет место зависимость:

Табельный номер → Номер проекта → Дата окончания

Здесь **Дата окончания** зависит от атрибута **Табельный номер** через **Номер проекта,** т.е. имеет место транзитивная зависимость.

Характер функциональных зависимостей в отношении влияет на оптимальность реляционной модели с точки зрения качества базы данных, построенной на этой модели.

Вторая нормальная форма

Наличие произвольных функциональных зависимостей в отношении приводит к излишнему дублированию при хранении данных, неудобствам их обработки и т.д. Поэтому существует ряд формальных требований к характеру функциональных зависимостей в отношении, которые позволяют избежать указанных недостатков.

Первое требование формулируется так: все атрибуты отношения не являющиеся первичными ключами, должны зависеть от единственного ключа. Рассмотрим следующий пример.

ЗАКАЗ(Код поставщика, Код товара, Наименование поставщика, Адрес, Наименование товара, Характеристики товара, Цена)

В этом отношении ключ состоит из пары атрибутов Код поставщика, Код товара . При этом **Наименование поставщика, Адрес** функционально зависят от атрибута **Код поставщика** , **Наименование товара, Характеристики товара** зависят от **Код товара** , а **Цена** от ключа отношения. Такое разнообразие функциональных зависимостей приводит к следующим проблемам.

При появлении нового поставщика необходимо добавлять строчку в отношение. Если он еще не начал поставлять товар, то все остальные атрибуты остаются не заполненными.

Если из отношения удаляются сведения о поставщике, то удалятся и сведения о товарах.

Для изменения адреса поставщика, наименование товара нужно проделывать это в нескольких строках отношения.

Для того, чтобы устранить эти проблемы, необходимо разбить это отношение на три следующим образом.

ПОСТАВЩИКИ(Код поставщика, Наименование поставщика, Адрес)

ТОВАРЫ(Код товара, Наименование товара, Характеристики товара)

ЗАКАЗ(Код поставщика, Код товара, Цена)

В этих отношениях каждый атрибут не являющийся ключом зависит только от ключа, дублирование данных минимизировано, и товары и поставщиков можно добавлять и удалять независимо. Говорят, что отношения , в которых каждый атрибут не являющийся ключом функционально зависит только от одного возможного ключа представлены во *второй нормальной форме*.

Третья нормальная форма

Проблемы подобного рода имеют место и для баз данных, в моделях которых встречаются транзитивные зависимости, то есть в отношениях существуют атрибуты, которые зависят от ключа через третьи атрибуты. Например в отношении:
ПЕРСОНАЛ(Табельный номер, ФИО, Должность, Номер проекта, Дата окончания)
Дата окончания зависит от ключа через **Номер проекта** . Наличие транзитивной зависимости приводит к следующим проблемам:

1. при известных номере проекта и дате окончания их негде разместить пока не появятся сведения хотя бы об одном исполнителе;
2. Если изменилась дата окончания проекта, ее надо менять в стольких кортежах, сколько людей работает над данным проектом.

Устранение этих проблем можно сделать, преобразовав исходное отношение к третьей нормальной форме:
ПЕРСОНАЛ(Табельный номер, ФИО, Должность, Номер проекта)

ПРОЕКТЫ(Номер проекта, Дата окончания)
Такое преобразование решает все отмеченные проблемы. Действительно, в отношение ПРОЕКТЫ заносятся сведения о существующих проектах независимо от того, работает над ними кто либо. При изменении даты окончания корректировка вносится только в один кортеж отношения ПРОЕКТЫ.
Говорят, что отношение задано в *третьей нормальной форме*, если оно представлено во второй нормальной форме и каждый атрибут не являющийся ключом не транзитивно зависит от ключа.
В реляционных моделях возможны коллизии, устранение которых требует преобразование отношений к четвертой и пятой нормальным формам. Однако, практически считается, что реляционная модель является удовлетворительной для построения базы данных, если все ее отношения представлены в третьей нормальной форме.

Схема проектирования реляционной модели данных (эмпирический подход)

Для создания реляционной модели данных при разработке базы данных для заданного объекта – предприятия необходимо выполнить следующие действия:

- Обследование информационной деятельности предприятия;
- Анализ информационных потоков и интеграция требований;
- Проектирование сетевой модели, отражающей структуру и информационные связи предприятия;
- Преобразование сетевой модели к реляционной;
- Нормализация отношений реляционной модели

Созданная в результате выполнения этих этапов модель может быть использована для создания базы данных в среде выбранной реляционной СУБД. Считается, что спроектированная таким образом база данных может быть использована для разработки приложений. Под приложением понимается созданный с помощью средств СУБД (в том числе и программных) интерфейс, позволяющий заполнять и эксплуатировать базу данных.

Обследование предприятия выполняется группой разработчиков с целью собрать информацию о структуре предприятия и информационных потоках. Под информационными потоками подразумевают информацию фиксируемую в виде документов и ее движение. Под движением информации понимается возникновение новых документов исходя из информации существующих и информации возникающей в процессе деятельности предприятия.

Анализ информационных потоков позволяет объединить представления пользователей всех подразделений предприятия, устранить дублирование документов. Результатом этого этапа является схема взаимодействия нормативной, справочной и текущей информации.

Полученная схема, вначале существующая как описание информационных структур и их взаимосвязей, преобразуется к более компактному виду – сетевой модели. Последующие этапы не требуют пояснений, так как были рассмотрены выше.

Основы реляционной алгебры

Реляционная модель, полученная в результате описанного технологического процесса, используется в СУБД для создания физической модели, которая и является структурой, где хранятся данные. Работа с базой данных состоит из трех направлений: ввод новых данных, изменение существующих и выборка данных для обработки. В общем случае все эти действия записываются в виде требований, называемых запросами. Наиболее удобно рассматривать запросы применительно к выборке данных. Поэтому все запросы в дальнейшем будем рассматривать применительно к этому виду действий. Со структурной точки зрения запрос состоит из атрибутов одного или нескольких отношений и условий, накладываемых на выборку данных. В процессе выборки данных происходит выделение отношений, относящихся к запросу, и их преобразование к одному отношению, из которого необходимо выбрать данные в соответствии с условиями поиска. Для того, чтобы это было возможным, была разработана формализованная система операций над отношениями, которая легла в основу реляционной алгебры.

Реляционной алгеброй или алгеброй отношений называют систему операций манипулирования отношениями, каждый оператор которой в качестве операнда (операндов) имеет одно или несколько отношений, образуя новое отношение по заранее обусловленному правилу. Основными операциями реляционной алгебры являются:

Операция проекции;

Операция объединения;

Операция разности;

Операция декартова произведения;

Операция селекции.

Кроме того, часто используются дополнительные операции, которые математически могут быть выражены через основные операции. Наиболее распространенными из них являются операция пересечения и операция соединения.

Операция проекции

Обозначение $\pi_{R(A)}$.

Представляет собой выборку кортежей отношения с неповторяющимися значениями домена А. Значения остальных доменов не играет роли.

Пример.

Сессия

Студент	Предмет	Семестр	Оценка
А..	Математика	1	4
А…	Информатика	1	3
Б..	Математика	1	5
Б…	Информатика	1	4
Б…	История	1	3
В...	Математика	1	4
В...	Информатика	1	3
В...	История	1	3

Проекция отношения $\pi_{\text{Сессия(Студент)}}$ будет выглядеть так:

Студент	Предмет	Семестр	Оценка
А..	Математика	1	4
Б..	Математика	1	5
В...	Математика	1	4

т.е. практически это будет список студентов. Значения всех остальных атрибутов берутся из первых встретившихся кортежей и не играют роли.

Операция объединения

Обозначение операции R **U** S . Объединение отношений R и S представляет собой множество кортежей, которые принадлежат отношениям либо R, либо S , либо им обоим. Для того, чтобы объединение было возможным, отношения операнды (R и S) должны быть совместимы для объединения – количество и типы объединяемых доменов должны быть одинаковы.

Пример. Пусть даны два отношения результатов сессии за 1 и 2 семестр.

Сессия1

Студент	Предмет	Семестр	Оценка
А..	Математика	1	4
Б..	Математика	1	5
В...	Математика	1	4

Сессия2

Студент	Предмет	Семестр	Оценка
А..	Математика	2	5
Б..	Математика	2	4
В...	Информатика	2	3
В...	История	2	4

Выполнение операции Сессия1 **U** Сессия2

Студент	Предмет	Семестр	Оценка
А..	Математика	1	4
Б..	Математика	1	5
В...	Математика	1	4
А..	Математика	2	5
Б..	Математика	2	4
В...	Информатика	2	3
В...	История	2	4

Операция разности

Математическое обозначение R – S .

Разностью отношений называется множество кортежей входящих в R , но не входящих в S. Замечание по совместимости отношений справедливо и для разности.

Пример. Пусть даны списки студентов получивших зачет и экзамен. Требуется выявить студентов, сдавших только зачет.

Зачет Экзамен Зачет – Экзамен

ФИО
Аверянов
Баранов
Вольский
Грачев
Григорьев
Дмитриев

ФИО
Баранов
Вольский
Григорьев
Дмитриев
Петров
Семенов

ФИО
Аверьянов
Грачев

Операция декартово произведение

Математическое обозначение R x S.

Декартово произведение на двух отношениях определяет новое отношение, у которого число столбцов равно сумме числа столбцов исходных отношений, а число

кортежей равно произведению числа кортежей операндов. При этом каждому кортежу первого отношения ставятся в соответствие все кортежи второго. Данная операция редко используется самостоятельно, поэтому приведем абстрактный пример

R S R x S

A	B
x	G
y	H
z	L

C	D
1	5
2	6

A	B	C	D
x	G	1	5
x	G	2	6
y	H	1	5
y	H	2	6
z	L	1	5
z	L	2	6

Операция селекции

Математическое обозначение $\sigma_{(A\ \theta\ B)}$ или $\sigma_{(A\ \theta\ V)}$.

Здесь A и B обозначения доменов, V – числовая или символьная константа, θ – знак логической операции (<,>,<>,<=,>=).

Операция селекции, это выборка кортежей со значениями доменов, удовлетворяющих заданному условию. Например, операция селекции $\sigma_{(\text{Оценка} > 3)}$ на приведенном ниже отношении

Студент	Предмет	Семестр	Оценка
А..	Математика	2	5
Б..	Математика	2	4
В...	Информатика	2	3
В...	История	2	4

даст отношение

Студент	Предмет	Семестр	Оценка
А..	Математика	2	5
Б..	Математика	2	4
В...	История	2	4

На базе основных операций реляционной алгебры основаны операции пересечения и соединения.

Операция пересечения

Операция обозначается R ∩ S и может быть выражена через операцию вычитания следующим образом: R – (R – S). По смыслу операция образует из двух отношений новое, которое включает совпадающие кортежи исходных отношений. Для примера рассмотрим исходные отношения операции вычитания. Если необходимо выяснить какие студенты сдали и зачет и экзамен, то результат будет получен при выполнении операции

Зачет –(Зачет –Экзамен)

Операция соединения

Математическое обозначение R $[\sigma_{(A\ \theta\ B)}]$S

Операция соединения представляет собой селекцию из декартова произведения. Разделяют θ – соединение и естественное соединения. В θ соединении из декартова произведения исходных отношений производится селекция по произвольному условию выборки. Например даны два отношения: «Наряд» и «Нормы»

Наряд

ФИО	Код	Объем
А…	01	10
Б…	02	15
В…	01	12
Г…	03	14

Нормы

Код	Наименование	Норма
01	Сварка	11
02	Расточка	14
03	Резка	15
04	Укладка	20

Требуется получить отношение, в котором отражены рабочие, не выполнившие норму. Для этого необходимо выполнить операцию

Наряд [Код = Код And Объем < Норма] Нормы

Выполнение операции включает два этапа декартово произведение и выборка в соответствии с условием.

	ФИО	Код	Объем	Код	Наименование	Норма
√	А…	01	10	01	Сварка	11
	А…	01	10	02	Расточка	14
	А…	01	10	03	Резка	15
	А…	01	10	04	Укладка	20
	Б…	02	15	01	Сварка	11
	Б…	02	15	02	Расточка	14
	Б…	02	15	03	Резка	15
	Б…	02	15	04	Укладка	20
	В…	01	12	01	Сварка	11
	В…	01	12	02	Расточка	14
	В…	01	12	03	Резка	15

	В…	01	12	04	Укладка	20
	Г…	03	14	01	Сварка	11
	Г…	03	14	02	Расточка	14
√	Г…	03	14	03	Резка	15
	Г…	03	14	04	Укладка	20

В результирующее отношение попадут помеченные галочкой два кортежа.

Естественное соединение предполагает в качестве условия отбора выражения, основанные на знаке =. Если для указанного примера необходимо получить отношение, в котором определены объемы и нормы по каждому работнику, то выражение операции и результат должны выглядеть так.

Наряд [Код = Код] Нормы

	ФИО	Код	Объем	Наименование	Норма
√	А…	01	10	Сварка	11
	А…	01	10	Расточка	14
	А…	01	10	Резка	15
	А…	01	10	Укладка	20
	Б…	02	15	Сварка	11
√	Б…	02	15	Расточка	14
	Б…	02	15	Резка	15
	Б…	02	15	Укладка	20
√	В…	01	12	Сварка	11
	В…	01	12	Расточка	14
	В…	01	12	Резка	15
	В…	01	12	Укладка	20
	Г…	03	14	Сварка	11
	Г…	03	14	Расточка	14
√	Г…	03	14	Резка	15
	Г…	03	14	Укладка	20

Можно заметить, что данном декартовом произведении отсутствует повторение домена «Код», т.к. при операции селекции по знаку равно они всегда будут совпадать.

Реляционное исчисление

Реляционное исчисление — прикладная ветвь формального механизма исчисления предикатов первого порядка. В основе исчисления лежит понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы. Наряду с реляционной алгеброй является способом получения результирующе-

го отношения в реляционной модели данных. В зависимости от того, что является областью определения переменной, различают:

- Исчисление кортежей
- Исчисление доменов

В общем случае все результаты полученные с помощью операторов реляционного исчисления могут быть получены с помощью реляционной алгебры. Преимуществом реляционного исчисления перед реляционной алгеброй можно считать то, что пользователю не требуется самому строить алгоритм выполнения запроса, Программа СУБД (при достаточной ее интеллектуальности) сама строит эффективный алгоритм.

Исчисление кортежей - направление реляционного исчисления, где областями определения переменных (операндов) являются отношения базы данных, то есть допустимым значением каждой переменной является кортеж некоторого отношения. В исчислении кортежей, как и в процедурных языках программирования, сначала нужно описать используемые переменные, а затем записать выражения запроса к данным.Описательную часть исчисления можно представить в виде: RANGE OF <переменная> IS <список>.Конструкция RANGE указывает идентификатор переменной кортежа <переменная> и область ее допустимых значений - <список> - последовательность одного или более элементов: x1, ..., xn, каждый из которых является либо отношением, либо выражением над отношением (порядок записи выражений описывается далее). При этом в любой момент <переменная> принимает в качестве значения только один из кортежей <списка> отношений.

Схемы отношений списка должны быть эквивалентными. Область допустимых значений <переменной> образуется путем объединения значений всех элементов списка.

Пример: RANGE OF Студент IS Очный_студент, Заочный_студент

Область определения переменной Студент включает в себя все значения из отношения, которое является объединением отношений Очный_студент и Заочный_студент.

Выражением реляционного исчисления кортежей называется конструкция вида <целевой_список > WHERE <WFF>

Значением выражения является отношение, тело (множество кортежей) которого должно удовлетворять WFF (well formulated formula — правильно построенная формула), а схема (набор атрибутов и их имена) определяется целевым списком. Целевой список по существу определяет операцию проекции, а формула WFF - селекцию кортежей.

В паре <переменная>.<атрибут> первая составляющая служит для указания переменной кортежа (определенной конструкцией RANGE), а вторая — для определения атрибута отношения, на котором изменяется переменная кортежа. Необязательная часть «AS <атрибут>» используется для переименования атрибута целевого отношения. Если она отсутствует, то имя атрибута целевого отношения наследуется от соответствующего имени атрибута исходного отношения.

Употребление в качестве элемента целевого отношения имени переменной равносильно перечислению в списке всех атрибутов соответствующего отношения.

WFF служат для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые сравнения, представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или констант). Например, конструкция "СТУДЕНТ.НОМЕР_ЗАЧЕТНОЙ_КНИЖКИ = 625432" является простым сравнением. По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, является простым сравнением.

Более сложные варианты WFF строятся с помощью логических связок NOT, AND, OR и IF ... THEN. Так, если <формула> - WFF, а <сравнение> - простое сравнение, то

NOT <формула>
<сравнение> AND <формула>
<сравнение> OR <формула>
IF <сравнение> THEN <формула>
являются WFF.

Допускается построение WFF с помощью кванторов. Если <формула> - это WFF, в которой участвует <переменная>, то конструкции

EXISTS <переменная> (<формула>)
FORALL <переменная> (<формула>)
являются WFF.

В первом случае WFF означает: "Существует по крайней мере одно такое значение <переменной>, что вычисление <формулы> дает значение ИСТИНА".

Во втором случае WFF означает: "Для всех значений переменной <переменной> вычисление <формулы> дает значение ИСТИНА".

Переменные, входящие в WFF, могут быть свободными или связанными. Все переменные, входящие в WFF, при построении которой не использовались кванторы, являются свободными. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение ИСТИНА, то эти значения кортежных переменных могут входить в результирующее отношение.

Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS <переменная> (<формула>) или FORALL <переменная> (<формула>), то в этой WFF и во всех WFF, построенных с ее участием, <переменная> - это связанная переменная. Это означает, что такая переменная не видна за пределами минимальной WFF, связавшей эту переменную. При вычислении значения такой WFF используется не одно значение связанной переменной, а вся ее область определения.

Пример. Пусть СОТР1 и СОТР2 - две кортежные переменные, определенные на отношении СОТРУДНИКИ.

СОТРУДНИКИ

ФИО	ЗАРПЛАТА
А…	14000
П…	9000
В…	6500
Л…	12000
Д…	8300

COTP2

(FOROLL)COTP1 true
(EXISTS)COTP1 true

Тогда, WFF EXISTS COTP2 (COTP1. ЗАРПЛАТА > COTP2. ЗАРПЛАТА) для текущего кортежа переменной COTP1 принимает значение true в том и только в том случае, если во всем отношении СОТРУДНИКИ найдется кортеж (связанный с переменной COTP2) такой, что значение его атрибута ЗАРПЛАТА удовлетворяет внутреннему условию сравнения.

WFF FORALL COTP2 (COTP1. ЗАРПЛАТА > COTP2. ЗАРПЛАТА) для текущего кортежа переменной COTP1 принимает значение true в том и только в том случае, если для всех кортежей отношения СОТРУДНИКИ (связанных с переменной COTP2) значения атрибута COTP_ЗАРП удовлетворяют условию сравнения.

Описанное исчисление не обладает вычислительной полнотой, так как не позволяет выполнять вычисления. Добавление вычислительных функций в исчисление можно реализовать путем расширения определения операндов сравнения и элементов целевого списка таким образом, чтобы они допускали использование скалярных выражений с литералами, ссылками на атрибуты и итоговыми функциями.

качестве итоговых могут выступать следующие функции: COUNT (количество), SUMM (сумма), AVG (среднее), MAX (максимальное), MIN (минимальное).

Для данных элементов целесообразно использовать спецификацию вида "AS <имя атрибута>, где можно явно задать имя результирующему атрибуту.

Пример.
Определить студента с максимальным рейтингом
Студент.ФИО, MAX(Рейтинг) AS Максимальный_Рейтинг
WHERE Студент.Номер_зачетной_книжки=Рейтинг.Номер_ зачетной_книжки _

В исчислении доменов областью определения переменных являются не отношения, а домены. Применительно к базе данных Рейтинг студентов можно говорить, например, о доменных переменных ИМЯ (значения - допустимые имена) или Номер_зачетной_книжки (значения - допустимые номера зачетных книжек студентов).

Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов, позволяющих выражать так называемые условия членства. Если R - это n-арное отношение с атрибутами a1, a2, ..., an, то условие членства имеет вид
R(a1i:v1i...aim:vim)(m<=n)

где vij - это либо литерально задаваемая константа, либо имя кортежной переменной. Условие членства принимает значение true в том и только в том случае, если в отношении R существует кортеж, содержащий указанные значения указанных атрибутов.

Если vij - константа, то на атрибут aij задается жесткое условие, не зависящее от текущих значений доменных переменных; если же - имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Во всем остальном формулы и выражения исчисления доменов выглядят похожими на формулы и выражения исчисления кортежей. В частности, конечно, различаются свободные и связанные вхождения доменных переменных.

Для примера сформулируем с использованием исчисления доменов запрос "Выдать номера и имена студентов сотрудников, не получающих минимальную заработную плату" (будем считать для простоты, что мы определили доменные переменные, имена которых совпадают с именами атрибутов отношения СОТРУДНИКИ, а в случае, когда требуется несколько доменных переменных, определенных на одном домене, мы будем добавлять в конце имени цифры):

```
СОТР_НОМ, СОТР_ИМЯ
WHERE EXISTS СОТР_ЗАРП1
(СОТРУДНИКИ (СОТР_ЗАРП1) AND
СОТРУДНИКИ (СОТР_НОМ, СОТР_ИМЯ, СОТР_ЗАРП) AND
СОТР_ЗАРП > СОТР_ЗАРП1)
```

Реляционное исчисление доменов является основой большинства языков запросов, основанных на использовании форм. В частности, на этом исчислении базировался известный язык Query-by-Example, который был первым (и наиболее интересным) языком в семействе языков, основанных на табличных формах.

Теоретические основы проектирования БД.

Основные понятия.

Дальнейшее изложение материала базируется на основных понятиях теории множеств, поэтому приведу основные определения этой теории. Наиболее простая структура данных, используемая в математике, имеет место в случае, когда между отдельными изолированными данными отсутствуют какие-либо взаимосвязи. Совокупность таких данных представляет собой множество. Понятие множества является неопределяемым понятием. Множество не обладает внутренней структурой. Множество можно представить себе как совокупность элементов, обладающих некоторым общим свойством. Для того чтобы некоторую совокупность элементов можно было назвать множеством, необходимо, чтобы выполнялись следующие условия:

Должно существовать правило, позволяющее определить, принадлежит ли указанный элемент данной совокупности.

Должно существовать правило, позволяющее отличать элементы друг от друга. (Это, в частности, означает, что множество не может содержать двух одинаковых элементов).

Множества обычно обозначаются заглавными латинскими буквами. Если элемент x принадлежит множеству A, то это обозначается:

$$x \in A$$

Если каждый элемент множества B является также и элементом множества A, то говорят, что множество B является *подмножеством* множества A:

$$B \subset A$$

Подмножество B множества A называется *собственным подмножеством*, если

$$B \neq A$$

Если допускается равенство подмножества и множества, то это обозначается:

$$B \subseteq A$$

Поскольку рассматриваемый подход к разработке реляционной модели базируется на формальной логике, то в его основе должны лежать некоторые фундаментальные формализации. В теории реляционных баз данных к ним относятся понятия атрибута, отношения, ключа и функциональной зависимости.

Атрибутом будем называть поименованное свойство объекта и обозначать A_i , где $1 \le i \le n$. Домен атрибута A_i обозначим dom(A_i). Тогда отношением R называется конечное множество атрибутов $\{A_1, A_2, ..., A_n\}$. Ключ отношения R является подмножеством $K = \{B_1, B_2, ..., B_m\} \subseteq R$ со следующим свойством. Для любых двух различных кортежей t_1 и t_2 в R существует такое $B \in R$, что $t_1(B) \neq t_2(B)$. Другими словами , не существует двух кортежей, имеющих одно и то же значение на всех атрибутах из К . Таким образом, достаточно знать К - значение кортежа, чтобы идентифицировать кортеж однозначно.

Пример.

СТУДЕНТ[<u>НОМЕР_ЗАЧЕТКИ</u>,ИМЯ,КУРС,ГРУППА]

Ключи, явно указанные в модели называются выделенными. Могут быть ключи отличные от выделенных и называемые неявными ключами. Например ИМЯ в предыдущем прмере.

Под функциональной зависимостью атрибутов или F-зависимостью понимают такую связь между атрибутами, когда значения кортежа на одном множестве атрибутов единственным образом определяют эти значения на другом множестве атрибутов. Так в отношении:

ГРАФИК[ПИЛОТ,РЕЙС,ДАТА,ВРЕМЯ]

ПИЛОТ функционально зависит от {РЕЙС,ДАТА}

F-зависимости принято обозначать {РЕЙС,ДАТА}-> ПИЛОТ и говорят, что РЕЙС и ДАТА функционально определяют ПИЛОТ.

В терминах теории множеств и реляционной алгебры F-зависимость определяется так. Пусть R отношение и X, Y подмножества атрибутов в R. Отношение R удовлетворяет функциональной зависимости X -> Y, если $\pi_Y(\sigma_{X\text{-}x}®)$ имеет не более чем один кортеж для каждого X - значения x. В F-зависимости X->Y подмножество X называется левой частью, а Y - правой частью.

Такая интерпретация функциональной зависимости является основой алгоритма SATISFIES, приводимого ниже.

SATISFIES

Вход: Отношение R и F-зависимость X->Y.

Выход: истина, если R удовлетворяет X->Y, ложь - в противном случае.

SATISFIES(R,X->Y)

1. Отсортировать отношение R по X-столбцам так, чтобы собрать кортежи с равными X-значениями вмести.

2. Если каждая совокупность кортежей с равными X-значениями имеет также равные Y-значения, то на выходе получаем истину, а в противном случае - ложь.

Этот алгоритм проверяет, удовлетворяет ли отношение R F-зависимости X -> Y.

Пример.

В результате выполнения алгоритма SATISFIES выясним удовлетворяет ли F-зависимость РЕЙС -> ВРЕМЯ_ВЫЛЕТА следующему отношению

ГРАФИК

ПИЛОТ	РЕЙС	ДАТА	ВРЕМЯ_ВЫЛЕТА
А...	83	9 авг	10:15
П...	83	11 авг	10:15
А...	116	10 авг	13:25
Р...	116	12 авг	13:25
П...	281	8 авг	5:50
С...	281	9 авг	5:50
П...	301	12 авг	18:35
С...	412	15 авг	13:25

Однако F-зависимость ВРЕМЯ_ВЫЛЕТА -> РЕЙС согласно этому алгоритму не выполняется для этого отношения

ГРАФИК

ПИЛОТ	РЕЙС	ДАТА	ВРЕ-МЯ_ВЫЛЕТА
П...	281	8 авг	5:50
С...	281	9 авг	5:50
А...	83	9 авг	10:15
П...	83	11 авг	10:15
А...	116	10 авг	13:25
Р...	116	12 авг	13:25
С...	412	15 авг	13:25
П...	301	12 авг	18:35

Для разработки модели базы данных необходимо знать полное множество F-зависимостей. Чтобы найти их, необходимы семантические знания об исходном отношении R. Поэтому можно считать семейство F-завсимостей заданным. Обозначим его L. Однако при таком подходе нельзя быть уверенным, что найдены все F-зависимости отношения R. Для того, чтобы найти все F-зависимости, если известны некоторые из них, можно воспользоваться аксиомами вывода. Возможность получения новых F-зависимостей с помощью аксиом вывода базируется на следующем правиле. Мнжество F-зависимостей L влечет за собой F-зависимость X -> Y (обозначение: L = X -> Y), если каждое отношение удовлетворяющее всем зависимостям в L, удовлетворяет также зависимости X -> Y. Аксиома вывода - это правило, устанавливающее, что если отношение удовлетворяет определенным F-зависимостям, то оно должно удовлетворять и некоторым другим F-зависимостям. Существует шесть аксиом вывода:

Рефлексивность: X -> X.
Пополнение: X -> Y влечет за собой XZ -> y.
Аддитивность: X -> Y и X -> Z влечет за собой X -> YZ.
Проективность: X -> YZ влечет за собой X -> Z.
Транзитивность: X -> Y и Y -> Z влечет за собой X -> Z.
Псевдотранзитивность: X -> Y и YZ -> W влечет за собой XZ -> W.

Пример.

Пусть дано отношение R , а X , Y и Z подмножества R . Предположим, что отношению удовлетворяет XY -> Z и X -> Y . Согласно аксиоме псевдотранзитивности получим XX -> Z или X -> Z.

Если даны аксиомы рефлексивности, пополнения и псевдотранзитивности, то из них можно вывести все остальные. Иногда их называют аксиомами Армстронга.

Пусть L-множество F-зависимостей для отношения R . *Замыкание* L , обозначаемое L^+ , - это наименьшее содержащее L множество, такое что при применении к нему аксиом Армстронга нельзя получить ни одной F - зависимости, не принадлежащей L.

Пример.

Пусть L = {AB -> C, C -> B } - множество F-зависимостей на R(ABC). L^+ = {A -> A, AB -> A, AC -> A, ABC -> A, B -> B, AB -> B, BC -> B, ABC -> B, C -> C, AC -> C, BC -> C, ABC -> C, AB -> AB, ABC -> AB, AC -> AC, ABC -> AC, BC -> BC, ABC -> BC, ABC -> ABC, AB -> C, AB -> AC, AB -> BC, AB -> ABC, C -> B, C -> BC, AC -> B, AC -> AB}

Таким образом, если известно множество F-зависимостей удовлетворяющих отношению R, можно найти все F- зависимости, удовлетворяющие этому отношению. Говорят, что L |= X -> Y ,если X -> Y $\subseteq L^+$.

Получение замыкания L^+ не обязательно для установления L = X -> Y.

Для этого достаточно воспользоваться алгоритмом MEMBER .

Алгоритм MEMBER.

Вход: Множество F-зависимостей L и F-зависимость X -> Y.

Выход: истина, если L = X -> Y, ложь в противном случае.

MEMBER(L, X -> Y)

if Y $\subseteq$ **CLOSURE**(X,L) then return (истина) else return(ложь)

Здесь CLOSURE алгоритм, позволяющий выявить список атрибутов входящих в множество L, который имеет вид.

Алгоритм CLOSURE.

Вход: Множество атрибутов X и множество F-зависимостей L.

Выход: Замыкание X над L.

CLOSURE(X,L)

OLDDEP = 0; NEWDEP = X
while NEWDEP $\neq$ OLDDEP

OLDDEP = NEWDEP
for каждая F- зависимость W -> Z в L

if W $\subseteq$ NEWDEP then NEWDEP = NEWDEP $\cup$ Z

next

end while

return(NEWDEP)

Пример работы алгоритма MEMBER

Пусть L = {НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА -> КОЛИЧЕСТВО_МЕСТ,

НОМЕР_РЕЙСА -> ПУНКТ_ОТПРАВЛЕНИЯ, НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА -> ПИЛОТ} и необходимо установить L |= НОМЕР_РЕЙСА -> ПИЛОТ

Используем для этого алгоритм MEMBER

Обращение к нему будет выглядеть так: MEMBER (L,НОМЕР_РЕЙСА -> ПИЛОТ). Результатом работы алгоритма будет истина, если будет истинным выражение $Y \subseteq$ CLOSURE(X,L), или с учетом данных примера ПИЛОТ $\subseteq$ CLOSURE(НОМЕР_РЕЙСА,L). Выполним поэтапно работу алгоритма CLOSURE как показано в таблице. В результате будет возвращено значение НОМЕР_РЕЙСА. ПУНКТ_ОТПРАВЛЕНИЯ. Атрибут ПИЛОТ не является подмножеством результата работы CLOSURE, поэтому L не влечет за собой НОМЕР_РЕЙСА -> ПИЛОТ, т.е. эту функциональную зависимость нельзя вывести из L.

Шаг	**NEWDEP**	**OLDDEP**	**Зависимость из L**	**Пояснения**
1	НОМЕР_РЕЙСА	0		Начальные значения переменных, если они не равны, то OLDDEP= NEWDEP
2	НОМЕР_РЕЙСА	НОМЕР_РЕЙСА	НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА -> КОЛИЧЕСТВО_МЕСТ	НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА не является подмножеством содержимого NEWDEP поэтому следующая зависимость из L
3	НОМЕР_РЕЙСА	НОМЕР_РЕЙСА	НОМЕР_РЕЙСА -> ПУНКТ_ОТПРАВЛЕНИЯ	НОМЕР_РЕЙСА подмножество NEWDEP поэтому добавим к NEWDEP ПУНКТ_ОТПРАВЛЕНИЯ
4	НОМЕР_РЕЙСА. ПУНКТ_ОТПРАВЛЕНИЯ	НОМЕР_РЕЙСА	НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА -> ПИЛОТ	НОМЕР_РЕЙСА, ДАТА_ВЫЛЕТА не является подмножеством NEWDEP и это последняя зависимость из L. NEWDEP не равно OLDDEP поэтому присваиваем OLDDEP значение NEWDEP и повторяем цикл
5	НОМЕР_РЕЙСА. ПУНКТ_ОТПРАВЛЕНИЯ	НОМЕР_РЕЙСА. ПУНКТ_ОТПРАВЛЕНИЯ		Повторяя перебор зависимостей из L убедимся, что NEWDEP не изменится, поэтому после окончания цикла окажется, что NEWDEP= OLDDEP и алгоритм CLOSURE возвращает значение НОМЕР_РЕЙСА. ПУНКТ_ОТПРАВЛЕНИЯ

Покрытия функциональных зависимостей

Для формирования БД, как системы взаимосвязанных отношений на основании известных (из семантических соображений) F-зависимостей необходимо иметь способ минимизации первоначального множества F-зависимостей. Это необходимо для минимизации дублирования данных, обеспечения их согласованности и целостности. Теоретической основой для построения такого способа является теория покрытий функциональных зависимостей.

Определение.

Два множества F-зависимостей L и G над отношением R эквивалентны, $L \equiv G$, если $L^+ = G^+$. Если $L \equiv G$, то L есть *покрытие* для G. Предполагается, что имеет смысл рассматривать в качестве покрытий такие множества L, которые не превосходят множество G по числу F-зависимостей.

Из этого определения следует, что для установления факта, что множество функциональных зависимостей L является покрытием G , необходимо определить эквивалентность L и G. Практически это достигается путем использования следующего алгоритма:

Алгоритм EQUIV
Вход: два множества F- зависимостей L и G.
Выход: истина, если $L \equiv G$; ложь в противном случае.

EQUIV(L,G)

```
v=DERIVES(L,G) and DERIVES(G,L);
return(v)
```

Здесь DERIVES алгоритм проверяет условие L |= G и имеет вид:

Алгоритм DERIVES
Вход: два множества F- зависимостей L и G.
Выход: истина, если L|= G; ложь в противном случае.

DERIVES(L,G)

```
v= истина
for каждая F-зависимость X -> Y из G
    v = v and MEMBER(L, X -> Y)
next
return(v)
```

Множество F-зависимостей L не избыточно, если у него нет такого собственного подмножества L’ , что L’≡ L . Если такое множество L’ существует, то L избыточно. L является не избыточным (минимальным) покрытием G, если L есть покрытие G и L не избыточно.

Пример. Пусть G = { AB -> C, A -> B, B -> C, A -> C}. Множество L= {AB -> C, A -> B, B -> C} эквивалентно G, но избыточно, потому что L’ = {A -> B, B ->

C} также является покрытием G. Множество L’ представляет собой не избыточное покрытие G.

Действительно, согласно алгоритму EQUIV $L \equiv G$, так как DERIVES(L,G) дает истину и DERIVES(G,L) так же истина. То же самое можно сказать относительно L’ и G.

Множество L не избыточно, если в нем не существует F-зависимости X -> Y, такой, что L - { X -> Y} |= X -> Y . Назовем F-зависимость из L избыточной в L , если L - { X -> Y} |= X -> Y.

Для любого множества F-зависимостей G существует некоторое подмножество L, такое, что L является не избыточным покрытием G. Если G не избыточно, то L=G. Для определения не избыточного покрытия множества F- зависимостей G существует алгоритм NONREDUN, который имеет вид:

Вход: множество F-зависимостей G.
Выход: не избыточное покрытие G.

NONREDUN(G)

```
L=G
for каждая зависимость X -> Y из G
   if MEMBER(L-(X->Y),X->Y) then L=L-(X->Y)
next
return(F)
```

Пример: Пусть G= {A -> B, B -> A, B -> C, A -> C}.

Результатом работы алгоритма является множество:

{A -> B, B -> A, A -> C}.

Если бы G было представлено в порядке {A -> B, A -> C, B -> A , B -> C} , то результатом работы алгоритма было бы

{A -> B, B -> A, B -> C}.

Из примера видно, что множество F-зависимостей G может иметь более одного не избыточного покрытия. Могут так же существовать не избыточные покрытия G, не содержащиеся в G. В приведенном примере таким не избыточным покрытием будет множество

L = {A -> B, B -> A, AB -> C}.

Если L-не избыточное множество F-зависимостей, то в нем нет “лишних” зависимостей в том смысле, что нельзя уменьшить L , удалив некоторые из них. Удаление любой F-зависимости из L приведет к множеству, не эквивалентному L. Однако размер можно уменьшить, удалив некоторые атрибуты F-зависимостей L.

Определение. Пусть L-множество F-зависимостей над R и X -> Y есть F-зависимость из L. Атрибут A из R называется посторонним в X -> Y относительно L, если

$X = AZ, X \neq Z$ и (L - {X -> Y}) $\cup$ {Z -> Y} $\equiv$ L или
Y = AW, Y $\neq$ W и (L - {X -> Y}) $\cup$ {X -> W} $\equiv$ L.

Иными словами, A - посторонний атрибут, если он может быть удален из правой или левой части X -> Y без изменения замыкания L.

Пример. Пусть L = {A -> BC,B -> C,AB -> D}. Атрибут C является посторонним в правой части A -> BC, а атрибут B - в левой части AB -> D.

Определение. Пусть L - множество F-зависимостей над R и X -> Y принадлежит L. F-зависимость X -> Y называется редуцированной слева, если X не содержит постороннего атрибута A и редуцированной справа, если Y не содержит атрибута A , постороннего для X -> Y. Зависимость X -> Y называется редуцированной, если она редуцирована слева и справа и Y $\neq \varnothing$. Редуцированная слева F-зависимость называется также полной F-зависимостью.

Определение. Множество F-зависимостей L называется редуцированным (слева, справа), если каждая F-зависимость из L редуцирована (соответственно слева, справа).

Пример. Множество L = {A -> BC, B -> C, AB -> D} не является редуцированным ни справа, ни слева. Множество L_1 = {A -> BC, B -> C, A -> D} редуцировано слева, но не справа, а L_2 = {A -> B, B -> C, AB -> D} редуцировано справа, но не слева. Множество L_3 = {A -> B, B -> C, A -> D} редуцировано слева и справа, следовательно, поскольку правые части не пусты, редуцировано.

Для нахождения редуцированных покрытий используется алгоритм:

REDUCE

Вход: множество F-зависимостей G.
Выход: редуцированное покрытие G.

REDUCE(G)

L= RIGHTRED(LEFTRED(G))
удалить из L все F-зависимости вида X -> $\varnothing$
return(L)

- здесь RIGHTRED и LEFTRED алгоритмы редуцирования соответственно справа и слева, которые имеют вид:

RIGHTRED

Вход: множество F-зависимостей G.
Выход: редуцированное справа покрытие G.

RIGHTRED(G)

```
 L = G
for каждая зависимость X -> Y из G
    for каждый атрибут A из Y
      if MEMBER(L ,(X ->(Y – A)) then  удалить A из Y в X -> Y из L
    next
next
return(L)
```

Алгоритм LEFTRED

```
Вход: множество F-зависимостей G.
Выход: редуцированое  слева покрытие G.
L = G
 for каждая зависимость X -> Y из G
   for каждый атрибут A из X
     if MEMBER(L,((X - A) -> Y)) then удалить A из X в X -> Y из L
   next
next
return(L)
```

Пример. Пусть G'= {A -> C, AB -> DE, AB ->CDI, AC -> J}. Из LEFTRED(G') получаем G" = {A -> C, AB -> DE, AB -> CDI, A -> J} и из RIGHTRED(G") получаем F= {A -> C, AB -> E, AB -> DI, A -> J}, уже редуцированное.

Далее потребуется находить разбиение множества F- зависимостей, заданных на отношении R на подмножества, которые представляют собой классы F-зависимостей с эквивалентной левой частью.

Определение: два множества атрибутов X и Y называются эквивалентными на множестве F- зависимостей L, если L |= X->Y и L |= Y ->X.

Например пусть дано L={A -> BC, B -> A, AD -> E}, найдем все F- зависимости эквивалентные левой части первой, т.е. A. Левая часть второй F- зависимости (B) эквивалентна A ? Проверим F |= A -> B и F |= B -> A . Это действительно так. Следовательно A эквивалентно B и первые две F- зависимости можно объединить в один класс эквивалентности, который обозначается в общем случае $E_A(X)$. Множество классов эквивалентности для заданного множества F- зависимостей обозначается $\overline{E}_L$. Сокращенным способом записи F- зависимостей с эквивалентными левыми частями является составная функциональная зависимость (CF-зависимость), которая имеет вид:

$$(X_1, X_2, ..., X_k) \rightarrow Y$$

где X_1, X_2, ..., X_k , множество эквивалентных левых частей F- зависимостей, а Y объединение правых частей F- зависимостей.

Синтез реляционных баз данных

База данных состоит из множества атрибутов и ключей. С точки зрения теоретико-множественного описания реляционной базой данных называется такая совокупность отношений $\{R_1, R_2, ...,R_p\}$, в которой каждое отношение имеет вид $R_i= (S_i,K_i)$, где S_i- множество атрибутов, а K_i - множество атрибутов образующих ключ.

Предположим на входе задано множество F- зависимостей L над R. С их помощью требуется создать базу данных $R=(R_1, R_2, ...,R_p)$. Эта БД должна удовлетворять следующим требованиям:

1. множество L полностью характеризуется с помощью R , т.е.

$$L \equiv \{Ki \rightarrow Ri \mid Ri \in R\}$$

где K_i – выделенный ключ R_i}

2. Каждое отношение Ri находится в третьей нормальной форме.
3. Не существует базы данных с меньшим числом отношений, удовлетворяющим пунктам 1 и 2.
4. Соединение всех полученных отношений Ri дает исходное отношение R.

Алгоритм порождающий базу данных из заданных F-зависимостей называется алгоритмом синтеза.

Определение. Если R – база данных и на ней задано множество F-зависимостей G, то в ней существует по крайней мере $\overline{E}$ G отношений. Это означает, что в R столько же отношений, сколько и классов эквивалентности. Из этого следует следующее.

Пусть L - множество F – зависимостей. Любая база данных должна иметь $\overline{E}$ L’ отношений, где L’ неизбыточное покрытие для L.

Исходя из этого строится способ построения структуры базы данных.

Сначала находится неизбыточное покрытие L’ для L и в $\overline{E}$ L’ вычисляем классы эквивалентности. Для каждого $E_{L'(X)}$ строим отношение, состоящее из всех атрибутов, появляющихся в $E_{L'(X)}$. При этом атрибуты левой части каждого класса эквивалентности образуют выделенный ключ.

Реализация этого способа позволяет получить алгоритм SYNTHESIZE

Вход: множество F – зависимостей L над R.

Выход: полная схема баз данных для L.

1. Найти для L редуцированное минимальное покрытие G.

2. Для каждой CF – зависимости $(X_1,X_2,\ldots,X_k) \rightarrow Y$ из G построить отношение $R_j= X_1X_2\ldots X_kY$ с выделенными ключами $K=\{X_1,X_2,\ldots X_k)$.
3. Вернуться к п. 2.

Пример.

$A \longrightarrow B_1B_2C_1C_2DEI_1I_2I_3J$

$B_1B_2C_1 \rightarrow AC_2DEI_1I_2I_3J$

$B_1B_2C_2 \longrightarrow AC_1DEI_1I_2I_3J$

$E \quad I_1I_2I_3$

$C_1D \longrightarrow J \qquad C_2D \longrightarrow J$

$I_1I_2 \longrightarrow I_3 \qquad I_2I_2 \longrightarrow I_1 \qquad I_1I_3 \longrightarrow I_2$

И пусть $R= AB_1B_2C_1C_2DEI_1I_2I_3J$

Множество минимально, но не редуцировано. Редуцируя F , получим

$F'= \{A \longrightarrow B_1B_2C_1C_2DE \qquad E \longrightarrow I_1I_2$

$B_1B_2C_1 \longrightarrow A \qquad B_1B_2C_2 \longrightarrow A$

$C_1D \longrightarrow J \qquad C_2D \longrightarrow J$

$I_1I_2 \longrightarrow I_3 \qquad I_2I_2 \longrightarrow I_1 \qquad I_1I_3 \longrightarrow I_2\}$

Образуя классы эквивалентности имеем

$G=\{ (AB_1B_2C_1 ,B_1B_2C_2) \longrightarrow DE$

$(E) \longrightarrow I_1I_2$

$(C_1D) \longrightarrow J \qquad (C_2D) \longrightarrow J$

$(I_1I_2, I_2I_2, I_1I_3)\}$

Преобразуя каждую CF – в отношения с выделенными ключами, получим

$R_1=AB_1B_2C_1C_2DE \quad K_1= \{AB_1B_2C_1 ,B_1B_2C_2\}$

$R_2= EI_1I_2 \qquad K_2=\{E\}$

$R_3= C_1DJ \qquad K_3=\{C_1D\}$

$R_4= C_2DJ \qquad K_4=\{C_2D\}$

$R_5= I_1I_2I_3 \quad K_5=\{ I_1I_2, I_2I_2, I_1I_3\}$

Окончательная схема БД будет $R=(R_1, R_2, R_3, R_4 ,R_5)$

Работа с реляционными базами данных

Основы реляционной модели данных были впервые изложены в статье Е.Кодда в 1970 г. Эта работа послужила стимулом для большого количества ста-

тей и книг, в которых реляционная модель получила дальнейшее развитие. Наиболее распространенная трактовка реляционной модели данных принадлежит К.Дейту. Согласно Дейту, реляционная модель состоит из трех частей:

- Структурной части.
- Целостной части.
- Манипуляционной части.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения.

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными - реляционную алгебру и реляционное исчисление. Эти механизмы манипулирования данными различаются уровнем процедурности:

- запрос, представленный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операций с учетом их старшинства и возможных скобок
- формула реляционного исчисления только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются более непроцедурными или декларативными.

Любые данные, используемые в программировании, имеют свои типы данных.Реляционная модель требует, чтобы типы используемых данных были простыми. Как правило, типы данных делятся на три группы:

- Простые типы данных.
- Структурированные типы данных.
- Ссылочные типы данных.

Простые, или атомарные, типы данных не обладают внутренней структурой. Данные такого типа называют скалярами. К простым типам данных относятся следующие типы:

- Логический.
- Строковый.
- Численный.

Различные языки программирования могут расширять и уточнять этот список, добавляя такие типы как:

- Целый.
- Вещественный.
- Дата.
- Время.
- Денежный.
- Перечислимый.
- Интервальный.
- И т.д.…

Конечно, понятие атомарности довольно относительно. Так, строковый тип данных можно рассматривать как одномерный массив символов, а целый тип данных - как набор битов. Важно лишь то, что при переходе на такой низкий уровень теряется семантика (смысл) данных. Если строку, выражающую, например, фамилию сотрудника, разложить в массив символов, то при этом теряется смысл такой строки как единого целого.

Структурированные типы данных предназначены для задания сложных структур данных. Структурированные типы данных конструируются из составляющих элементов, называемых компонентами, которые, в свою очередь, могут обладать структурой. В качестве структурированных типов данных можно привести следующие типы данных:

Массивы

Записи (Структуры)

С математической точки зрения массив представляет собой функцию с конечной областью определения. Например, рассмотрим конечное множество натуральных чисел

$$A = \{1,2,\ldots,n\}$$

называемое множеством индексов. Отображение

$$F : A \to \mathrm{R}$$

из множества A во множество вещественных чисел R задает одномерный вещественный массив. Значение этой функции для некоторого значения индекса i называется элементом массива, соответствующим i. Аналогично можно задавать многомерные массивы.

Запись (или структура) представляет собой кортеж из некоторого декартового произведения множеств. Действительно, запись представляет собой именованный упорядоченный набор элементов r_i, каждый из которых принадлежит типу T_i. Таким образом, запись $r = (r_1, r_2, \ldots, r_n)$ есть элемент множества $T = T_1 \times T_2 \times \ldots \times T_n$.

Объявляя новые типы записей на основе уже имеющихся типов, пользователь может конструировать сколь угодно сложные типы данных.

Общим для структурированных типов данных является то, что они имеют внутреннюю структуру, используемую на том же уровне абстракции, что и сами типы данных.

Поясним это следующим образом. При работе с массивами или записями можно манипулировать массивом или записью и как с единым целым (создавать, удалять, копировать целые массивы или записи), так и поэлементно. Для структурированных типов данных есть специальные функции - конструкторы типов, позволяющие создавать массивы или записи из элементов более простых типов.

Работая же с простыми типами данных, например с числовыми, мы манипулируем ими как неделимыми целыми объектами. Чтобы "увидеть", что числовой тип данных на самом деле сложен (является набором битов), нужно перейти на более низкий уровень абстракции. На уровне программного кода это будет выглядеть как ассемблерные вставки в код на языке высокого уровня или использование специальных побитных операций.

Ссылочный тип данных (указатели) предназначен для обеспечения возможности указания на другие данные. Указатели характерны для языков процедурного типа, в которых есть понятие области памяти для хранения данных. Ссылочный тип данных предназначен для обработки сложных изменяющихся структур, например деревьев, графов, рекурсивных структур.

Технологии работы с реляционными базами данных

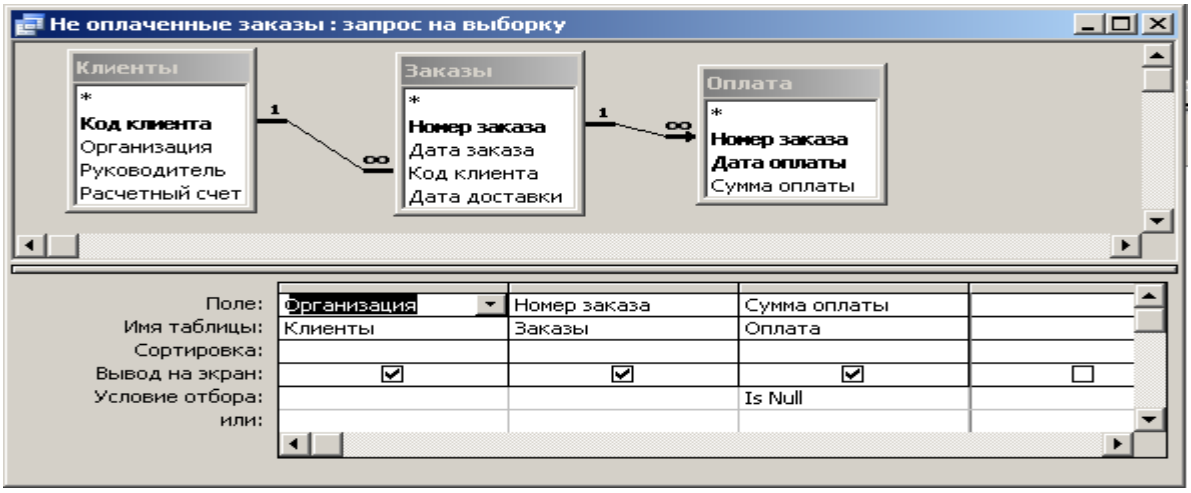
Реляционные базы данных представляют собой совокупность взаимосвязанных отношений (таблиц), управление которыми осуществляется с помощью систем управления базами данных (СУБД). В сложных вычислительных системах, включающих множество компьютеров, соединённых с помощью сетевого оборудования размещение баз данных в зависимости от их назначения может выполняться различными СУБД и разными способами. Для баз данных обслуживающих решение задач в небольшой локальной сети (до 20 компьютеров) достаточно настольной СУБД типа Access, Paradox или FoxPro. Создавать базы данных решающих единый комплекс задач на каждом из компьютеров такой сети сложно, так как придется постоянно обмениваться данными между ними, что трудоемко с программной точки зрения и создает излишнюю нагрузку на сеть. Поэтому базу данных устанавливают на один из компьютеров в случае одноранговой сети или на выделенный сервер, обеспечивая остальным компьютерам сети доступ к ней. Как правило, функции управления данными названных систем достаточно ограничены и сводятся к предоставлению данных (отношений), необходимых для реализации запроса, созданию и восстановлению копий баз данных. В этих системах реализуется *файл-серверная* технология, в соответствии с которой СУБД пересылает по сети компьютеру пославшему запрос набор отношений (таблиц), необходимых для его выполнения. Этот компьютер реализует запрос и при необходимости возвращает изменённые таблицы серверу. Недостаток этой технологии очевиден: по сети передаются достаточно большие

объёмы данных, которые перегружают трафик сети. Поэтому в больших сетях используют *клиент-серверные* технологии. Для их реализации используют более мощные СУБД, такие как MSSQL SERVER, MYSQL SERVER, ORACLE и другие, функциональность которых включает обеспечение хранения не только данных, но и особых форм запросов подразделяющихся на три группы: представления, хранимые процедуры и функции. Представления это обычные запросы, но хранящиеся в базе на сервере. Хранимые процедуры это фрагменты программ написанных на расширенном SQL, которые могут принимать параметры управляющие ходом вычислений в процедуре и возвращать пользователю требуемые результаты. Функции это те же хранимые процедуры, но возвращают результат только одного типа, определённого в функции. Особенностью работы представлений, хранимых процедур и функций в клиент-серверной технологии является то, что пользовать, посылая запрос серверу, инициирует выполнение представления, хранимой процедуры или функции на сервере. Пользователю возвращаются только результаты. Таким образом, резко снижается трафик сети. СУБД поддерживающие клиент-серверные технологии включают множество дополнительных функций, такие, как управление доступом к базам данных, управление резервным копированием и восстановлением баз данных и так далее. Как правило, клиент-серверные технологии используются в сетях корпоративного и выше уровня. Подробно работа с этими элементами будет рассмотрена в курсе «База данных».

Основной формой управления данными в базах являются запросы. Под запросом понимается формализованное требование на манипулирование данными базы. Запрос начинается со словесного формулирования требования на манипулирование данными базы. Под манипулированием данными понимается выборка (SELECT), добавление (INSERT), обновление (UPDATE) и удаление (DELETE) данных реляционной базы. Для того, чтобы формализовать словесное требование, необходимо преобразовать его таким образом, чтобы оно выражалось через атрибуты отношений базы. А затем выразить либо на языке SQL, либо в форме QBE, которая в конечном итоге преобразует запрос в SQL выражение.

Операции реляционной алгебры используются в процессе выполнения запросов над базой данных. Наиболее распространенными формами описания запросов являются запись запроса на языке SQL(Structured Query Language) и в форме QBE(Query By Example). Кратко рассмотрим конструкции обоих форм представления. Реализацию запросов в обоих формах рассмотрим на примере базы данных построенной на практических занятиях. Пусть нужно реализовать запрос: Какие организации не оплатили заказы?

Из текста запроса ясно, что для ответа на него нужно выбрать данные атрибутов: наименование организации, номер заказа и сумма оплаты. Получить эти данные можно из отношений «Клиенты», «Заказы» и «Оплата». В соответствии с моделью данных базы между этими отношениями установлены следующие связи.



В форме QBE запрос примет вид показанный на рисунке. В соответствии с требованием запроса из базы нужно выбрать значения указанных атрибутов, для которых вообще не введена оплата, т.е. значение соответствующего атрибута «Сумма оплаты» пусто (Null). В организации связей этого запроса есть одна особенность. Связь 1:М между отношениями «Заказы» и «Оплата» подразумевает выборку только существующих кортежей. Однако, если для какого либо заказа оплата не введена, то не будет кортежа в отношении «Оплата», соответствующего кортежу отношения «Заказы», т.е. условие «Сумма оплаты» = Null не будет работать. Для того, чтобы данное условие заработало нужно модифицировать связь, наложив на нее требование показывать все кортежи отношения «Заказы», а если им не соответствует не одного кортежа в отношении «Оплата», то ставить вместо него кортеж со значениями атрибутов Null. Таким образом условие запроса приобретает смысл.

В форме SQL данный запрос приобретает следующий вид.

SELECT Клиенты.Организация, Заказы.[Номер заказа], Оплата.[Сумма оплаты]
FROM Клиенты INNER JOIN (Заказы LEFT JOIN Оплата ON Заказы.[Номер заказа] = Оплата.[Номер заказа]) ON Клиенты.[Код клиента] = Заказы.[Код клиента]
WHERE (((Оплата.[Сумма оплаты]) Is Null));

В этом запросе использованы следующие операторы языка SQL:

SELECT – выбрать

FROM - из

INNER JOIN –соединить один ко многим

LEFT JOIN – соединить слева

WHERE – где

ON - по

В соответствии со смыслом операторов данный запрос на языке SQL можно прочитать как:

ВЫБРАТЬ Клиенты.Организация, Заказы.[Номер заказа], Оплата.[Сумма оплаты]
ИЗ Клиенты **СОЕДИНЕННЫЙ 1:М** (Заказы **СОЕДИНЕННЫЙ СЛЕВА** Оплата

ПО Заказы.[Номер заказа] = Оплата.[Номер заказа]) **ПО** Клиенты.[Код клиента] = Заказы.[Код клиента] **ГДЕ** (((Оплата.[Сумма оплаты]) Is Null));

Данный пример показывает использование операции соединения. Далее операторы SQL будут рассмотрены более полно и подробно.

Распределенная обработка данных

Под распределенной обработкой данных понимается такой способ хранения и обработки данных, когда отдельное приложение может обрабатывать данные распределенные на множестве различных баз данных, управление которыми осуществляют различными СУБД, работающие на различных машинах с различными операционными системами, соединенных коммуникационными системами. Распределенная база данных (РБД) является виртуальным объектом, части которого расположены на удаленных базах данных, связанных каналами связи.

Физически РБД состоит из набора **узлов**, связанных коммуникационной сетью, в которой:

- Каждый узел обладает своими собственными системами баз данных;
- Узлы работают согласованно, поэтому пользователь может получить доступ к данным на любом узле сети, как будто все данные находятся на собственном узле.

Каждый узел обладает своими собственными базами данных, собственными локальными пользователями, собственной СУБД и программным обеспечением для управления транзакциями, а так же собственным диспетчером передачи данных. Распределенная СУБД может рассматриваться как некий способ совместной работы отдельных локальных СУБД, расположенных на разных локальных узлах. Причем новый компонент программного обеспечения на каждом узле поддерживает все необходимые функции совместной работы. Комбинация этого компонента и существующей СУБД называется Распределенной Системой Управления Базами Данных (РСУБД).

В основе распределённых баз данных лежат следующие требования:

1. Локальная автономия;
2. Независимость от центрального узла;
3. Непрерывное функционирование;
4. Независимость от расположения;
5. Независимость от фрагментации;
6. Независимость от репликации;
7. Обработка распределённых запросов;
8. Управление распределёнными транзакциями;

9. Независимость от аппаратного обеспечения;
10. Независимость от операционной системы;
11. Независимость от сети;
12. Независимость от СУБД.

Локальная автономия

В распределенной системе узлы следует делать автономными. Локальная автономия означает, что функционирование любого узла X не зависит от успешного выполнения операций на некотором узле У . В противном случае выход из строя узла У может привести к невозможности выполнения операций на узле X . Из принципа локальной автономии следует, что владение и управление данными осуществляется локально вместе с локальным ведением учета. В действительности цель локальной автономии достигается не полностью, поскольку часто узел X должен представлять некоторую часть управления узлу У, поэтому говорят не о полной, а о максимально возможной автономии.

Независимость от центрального узла.

Под локальной автономией понимается, что все узлы должны рассматриваться как равные. Следовательно, не должно существовать никакой зависимости и от центрального «основного» узла с некоторым централизованным обслуживанием, например централизованной обработкой запросов, централизованным управлением транзакциями или централизованным присвоением имен. Зависимость от центрального узла нежелательна по двум причинам. Во-первых, центральный узел может быть «узким» местом всей системы, а во-вторых, более важно то, что система в целом становится уязвимой, т.е. при повреждении центрального узла может выйти из строя вся система.

Непрерывное функционирование

Одним из преимуществ распределенных систем является то, что они обеспечивают более высокую надежность и доступность.

- **Надежность** (вероятность того, что система выполняет свойственные ей функции в заданный момент времени) повышается благодаря работе распределенных систем не по принципу «все или ничего», а в постоянном режиме; т.е. работа системы продолжается , хотя и на более низком уровне, даже в случае неисправности некоторого отдельного компонента, например узла.
- **Доступность** (вероятность того, что система исправна и работает в течение некоторого промежутка времени) повышается частично по

той же причине, а частично благодаря возможности репликации данных.

Независимость от расположения

Эта цель предполагает обеспечение такого режима работы с данными, при котором пользователю не нужно знать, на каком узле находятся требуемые данные. При этом значительно упрощаются пользовательские программы, а также не требуется их изменения при перемещении данных в системе.

Независимость от фрагментации

В системе поддерживается фрагментация данных, если некоторое отношение из соображений физического хранения необходимо разделить на части или фрагменты. Фрагментация желательна для повышения производительности системы, поскольку данные лучше хранить в том месте, где они наиболее часто используются. При такой организации многие операции становятся локальными, а объем передаваемых в сети данных снизится.

Существует два типа фрагментации – горизонтальная и вертикальная, которые связаны с операциями селекции и проекции соответственно, т.е. горизонтальный фрагмент может быть получен с помощью операции селекции, а вертикальный – проекцией. Реконструкцию исходного отношения на основе его фрагментов можно осуществить с помощью операций соединения (для вертикальных фрагментов) и объединения (для горизонтальных фрагментов).

В фрагментированной системе необходимо обеспечить поддержку независимости от фрагментации, т.е. пользователь не должен «ощущать» фрагментацию данных.

Независимость от репликации

В системе поддерживается независимость от репликации, если заданное отношение или фрагмент могут быть представлены различными копиями (репликами) хранимыми на разных узлах. Репликация полезна по двум причинам. Во-первых, благодаря ей достигается большая производительность, т.к. приложения могут работать с локальными копиями , не обмениваясь данными с удаленными узлами. Во-вторых, репликация позволяет обеспечить большую доступность, т.к. реплицированный объект остается доступным для обработки до тех пор, пока остается хотя бы одна его реплика. Главный недостаток репликации заключается в том, что при обновлении реплицируемого объекта, все его копии должны синхронизироваться.

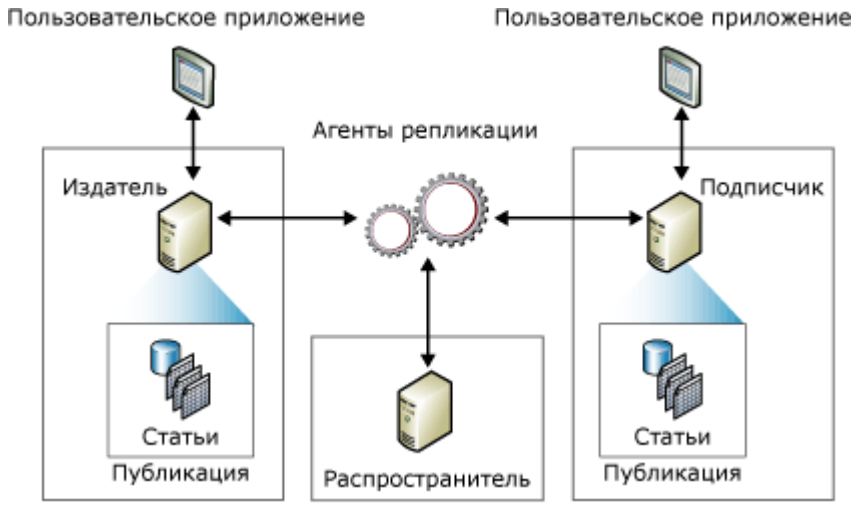
Репликация использует терминологию издательской отрасли для представления компонентов в топологии репликации, которые включают издатель, распространитель, подписчики, публикации, статьи и подписки. Удобно представить репликацию Microsoft SQL Server с помощью терминологии журнала:

- Издатель журнала производит одну или несколько публикаций

- Публикация содержит статьи
- Издатель или распространяет журнал напрямую, или использует распространитель
- Подписчики получают публикации, на которые они подписались

Хотя сравнение с журналом полезно для понимания репликации, следует отметить, что репликация SQL Server включает возможности, которые не представлены в данной метафоре, в частности возможность подписчика выполнять обновления и возможность издателя посылать дополнительные изменения в статьи публикации.

Топология репликации определяет отношения между серверами и копиями данных, и проясняет логику, определяющую порядок обмена данными между серверами. Существует несколько процессов репликации (называемых агентами), которые отвечают за копирование и перемещение данных между издателем и подписчиками. Следующая иллюстрация представляет собой обзор компонентов и процессов, входящих в репликацию.



Издатель

Издатель — это экземпляр базы данных, который делает данные доступными в других местах посредством репликации. Издатель может иметь одну или более публикаций, каждая из которых определяет логически связный набор объектов и данных для репликации.

Распространитель

Распространитель — это экземпляр базы данных, который действует как хранилище специальных данных репликации, связанных с одним или несколькими издателями. Каждый издатель связан с одной базой данных (называемой базой данных распространителя), располагающейся на распространителе. База данных распространителя хранит информацию о состоянии репликации, метаданные публикации, и иногда действует как очередь для перемещения данных с издателя на подписчики. Нередко единичный экземпляр сервера базы данных действует одновременно как издатель и как распространитель. Такой сервер базы известен как локальный распространитель. Когда издатель и распространитель настроены на раз-

ных экземплярах серверов баз данных, распространитель называют удаленным распространителем.

Подписчики

Подписчик — это экземпляр базы данных, который получает реплицированные данные. Подписчик может получать данные от нескольких издателей и публикаций. В зависимости от выбранного типа репликации подписчик может также передавать изменения данных издателю или переиздавать эти данные на другие подписчики.

Статья

Статья определяет объект базы данных, включенный в публикацию. Публикация может содержать разные типы статей, включая таблицы, представления, хранимые процедуры и другие объекты. Если таблицы публикуются в виде статей, можно использовать фильтры для ограничения столбцов и строк данных, посылаемых подписчикам.

Публикация

Публикация — это коллекция из одной или нескольких статей, принадлежащих одной базе данных. Группировка нескольких статей в публикацию упрощает указание логически связанного набора объектов и данных базы данных, реплицируемых в виде единого блока.

Подписка

Подписка — это запрос на доставку копии публикации подписчику. Подписка определяет, какая публикация будет получена, где и когда. Существует два типа подписок: принудительная и по запросу.

В системе, которая поддерживает репликацию данных, должна также поддерживаться независимость от репликации, т.е. пользователь не должен касаться проблем связанных с созданием и синхронизацией копий.

Обработка распределенных запросов

При обработке в распределенной системе запроса необходимо выработать эффективную стратегию его реализации. Например, запрос на объединение отношений Rx , расположенного на узле X , и отношения Ry , хранимого на узле Y , может быть выполнен с помощью перемещения отношения Rx на узел Y , перемещения отношения Ry на узел X или перемещения этих двух отношений на третий узел Z и т.д. Это означает, что при выполнении запроса на распределенной БД необходим его предварительный анализ с последующим выбором оптимальной стратегии его реализации.

Управление распределенными транзакциями

В распределенной системе выполнение транзакции связано с исполнением программных кодов на нескольких узлах. Транзакция это логическая единица работы, которая включает всю совокупность действий, необходимых для реализации за-

проса. Транзакция считается неделимым процессом, т.е. если какое либо из составляющих действий окажется не выполненным, то вся транзакция считается не выполненной. Каждый программный код, исполняемый на каком либо узле при выполнении транзакции, называется агентом. Таким образом, транзакция состоит из нескольких агентов, т.е. процессов реализующих транзакцию.

В процессе управления транзакцией выделяют управление восстановлением и управление параллельной обработкой. Первое из них базируется на протоколе двухфазной фиксации. В грубом приближении в соответствии с этим протоколом в начале транзакции устанавливается точка фиксации данных, т.е. как бы создается копия данных, которые предполагается изменить в результате транзакции. Если транзакция завершена нормально, то точка фиксации сохраняется до выполнения следующей транзакции. Если же произошел сбой, то система возвращает состояние данных в точку фиксации, позволяя не допустить необратимого неправильного изменения БД. Управление параллельной обработкой предполагает установку блокировок на отношения, группы записей с целью не допустить изменение данных другим пользователем во время выполнения транзакции. Более подробно транзакции рассмотрены ниже.

Независимость от аппаратного обеспечения

Используемые в настоящее время компьютеры характеризуются большим разнообразием. В связи с этим существует необходимость интеграции данных на всех системах и создания для пользователя представления единой системы. Должна иметься возможность запуска одной и той же СУБД на разном аппаратном обеспечении.

Независимость от операционной системы

Эта цель является следствием предыдущей. Необходимо, чтобы одна и та же СУБД могла работать под управлением разных ОС.

Независимость от сети

Если система в состоянии поддерживать несколько узлов с разным аппаратным обеспечением и разными операционными системами, то желательно, чтобы в ней поддерживались разные типы сетей.

Независимость от СУБД

Эта цель означает, что желательно, чтобы распределенная БД допускала использование различных СУБД разными пользователями. Это возможно только если эти СУБД поддерживают некоторый общий стандарт представления данных, например, официальный стандарт языка SQL.

Реализация запросов с использованием языка SQL

Текущая версия стандарта языка SQL принята в 1992 г. (Официальное название стандарта - Международный стандарт языка баз данных SQL (1992)

(International Standart Database Language SQL), неофициальное название - SQL/92, или SQL-92, или SQL2). Документ, описывающий стандарт, содержит более 600 страниц.

Язык SQL стал фактически стандартным языком доступа к базам данных. Все СУБД, претендующие на название "реляционные", реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют в настоящее время средства доступа к реляционным данным. Целью стандартизации является переносимость приложений между различными СУБД.

Нужно заметить, что в настоящее время, ни одна система не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект - это надмножество некоторого подмножества стандарта SQL. Это затрудняет переносимость приложений, разработанных для одних СУБД в другие СУБД.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее. Например, отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия.

Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

Операторы SQL

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям.

Можно выделить следующие группы операторов (перечислены не все операторы SQL):

Операторы DDL (Data Definition Language) - операторы определения объектов базы данных

- CREATE SCHEMA - создать схему базы данных
- DROP SHEMA - удалить схему базы данных
- CREATE TABLE - создать таблицу
- ALTER TABLE - изменить таблицу
- DROP TABLE - удалить таблицу
- CREATE DOMAIN - создать домен
- ALTER DOMAIN - изменить домен
- DROP DOMAIN - удалить домен

- CREATE COLLATION - создать последовательность
- DROP COLLATION - удалить последовательность
- CREATE VIEW - создать представление
- DROP VIEW - удалить представление

Операторы DML (Data Manipulation Language) - операторы манипулирования данными

- SELECT - отобрать строки из таблиц
- INSERT - добавить строки в таблицу
- UPDATE - изменить строки в таблице
- DELETE - удалить строки в таблице
- COMMIT - зафиксировать внесенные изменения
- ROLLBACK - откатить внесенные изменения

Операторы защиты и управления данными

- CREATE ASSERTION - создать ограничение
- DROP ASSERTION - удалить ограничение
- GRANT - предоставить привилегии пользователю или приложению на манипулирование объектами
- REVOKE - отменить привилегии пользователя или приложения

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Наиболее важными для пользователя являются операторы манипулирования данными (DML).

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

Для того, чтобы показать, что язык SQL является реляционно полным, нужно показать, что любой реляционный оператор может быть выражен средствами SQL. На самом деле достаточно показать, что средствами SQL можно выразить любой из *примитивных* реляционных операторов.

Оператор декартового произведения

Реляционная алгебра: $A \text{ TIMES } B$

Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A, B;
```

или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A CROSS JOIN B;
```

Оператор проекции

Реляционная алгебра: $A[X,Y,\ldots,Z]$

Оператор SQL:

```
SELECT DISTINCT X, Y, …, Z
    FROM A;
```

Оператор выборки

Реляционная алгебра: $A\ \mathrm{WHERE}\ c$,

Оператор SQL:

```
SELECT *
    FROM A
    WHERE c;
```

Оператор объединения

Реляционная алгебра: $A\ \mathrm{UNION}\ B$

Оператор SQL:

```
SELECT *
    FROM A
    UNION
    SELECT *
        FROM B;
```

Оператор вычитания

Реляционная алгебра: $A\ \mathrm{MINUS}\ B$

```
Оператор SQL:
SELECT *
    FROM A
    EXCEPT
    SELECT *
        FROM B
```

Реляционный оператор переименования RENAME выражается при помощи ключевого слова AS в списке отбираемых полей оператора SELECT. Таким образом, язык SQL является реляционно полным.

Остальные операторы реляционной алгебры (соединение, пересечение, деление) выражаются через примитивные, следовательно, могут быть выражены операторами SQL. Тем не менее, для практических целей приведем их.

Оператор соединения

Реляционная алгебра: $(A \text{ TIMES } B) \text{ WHERE } c$

Оператор SQL:

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A, B
    WHERE c;
```

или

```
SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …
    FROM A CROSS JOIN B
    WHERE c;
```

Приложение

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

Пример 1. Вставка одной строки в таблицу:

```
INSERT INTO
  P (PNUM, PNAME)
  VALUES (4, "Иванов");
```

Пример 2. Вставка в таблицу нескольких строк, выбранных из другой таблицы (в таблицу TMP_TABLE вставляются данные о поставщиках из таблицы P, имеющие номера, большие 2):

```
INSERT INTO
  TMP_TABLE (PNUM, PNAME)
  SELECT PNUM, PNAME
    FROM P
    WHERE P.PNUM>2;
```

UPDATE - обновление строк в таблице

Пример 3. Обновление нескольких строк в таблице:

```
UPDATE P
```

```
  SET PNAME = "Пушников"
  WHERE P.PNUM = 1;
```

DELETE - удаление строк в таблице

Пример 4. Удаление нескольких строк в таблице:

```
DELETE FROM P
  WHERE P.PNUM = 1;
```

Пример 5. Удаление всех строк в таблице:

```
DELETE FROM P;
```

Примеры использования оператора SELECT

Оператор SELECT является фактически самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных - предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных.

Замечание. На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления. Представления - это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления - это таблица, которая не хранится постоянно в базе данных, а "возникает" в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между хранимыми таблицами и представлениями, но эти различия *скрыты* от пользователя.

Результатом выполнения оператора SELECT всегда является таблица. Таким образом, по результатам действий оператор SELECT похож на операторы реляционной алгебры. Любой оператор реляционной алгебры может быть выражен подходящим образом сформулированным оператором SELECT. Сложность оператора SELECT определяется тем, что он содержит в себе все возможности реляционной алгебры, а также дополнительные возможности, которых в реляционной алгебре нет.

Отбор данных из одной таблицы

Пример 6. Выбрать все данные из таблицы поставщиков (ключевые слова ***SELECT***… ***FROM***…):

```
SELECT *
  FROM P;
```

Замечание. В результате получим новую таблицу, содержащую полную копию данных из исходной таблицы P.

Пример 7. Выбрать все строки из таблицы поставщиков, удовлетворяющих некоторому условию (ключевое слово ***WHERE***…):

```
SELECT *
  FROM P
  WHERE P.PNUM > 2;
```

Замечание. В качестве условия в разделе WHERE можно использовать сложные логические выражения, использующие поля таблиц, константы, сравнения (>, <, = и т.д.), скобки, союзы AND и OR, отрицание NOT.

Пример 8. Выбрать некоторые колонки из исходной таблицы (указание списка отбираемых колонок):

```
SELECT P.NAME
  FROM P;
```

Замечание. В результате получим таблицу с одной колонкой, содержащую все наименования поставщиков.

Замечание. Если в исходной таблице присутствовало несколько поставщиков с разными номерами, но одинаковыми наименованиями, то в результатирующей таблице *будут строки с повторениями* - дубликаты строк автоматически не отбрасываются.

Пример 9. Выбрать некоторые колонки из исходной таблицы, удалив из результата повторяющиеся строки (ключевое слово ***DISTINCT***):

```
SELECT DISTINCT P.NAME
  FROM P;
```

Замечание. Использование ключевого слова DISTINCT приводит к тому, что в результатирующей таблице будут удалены все повторяющиеся строки.

Пример 10. Использование скалярных выражений и переименований колонок в запросах (ключевое слово ***AS***…):

```
SELECT
    TOVAR.TNAME,
    TOVAR.KOL,
    TOVAR.PRICE,
    "=" AS EQU,
    TOVAR.KOL*TOVAR.PRICE AS SUMMA
  FROM TOVAR;
```

В результате получим таблицу с колонками, которых не было в исходной таблице TOVAR:

TNAME	KOL	PRICE	EQU	SUMMA
Болт	10	100	=	1000
Гайка	20	200	=	4000

Винт	30	300	=	9000

Пример 11.Упорядочение результатов запроса (ключевое слово ***ORDER BY***…):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY DNUM;
```

В результате получим следующую таблицу, упорядоченную по полю DNUM:

PNUM	DNUM	VOLUME
1	1	100
2	1	150
3	1	1000
1	2	200
2	2	250
1	3	300

Пример 12. Упорядочение результатов запроса по нескольким полям с возрастанием или убыванием (ключевые слова ***ASC***, ***DESC***):

```
SELECT
    PD.PNUM,
    PD.DNUM,
    PD.VOLUME
  FROM PD
  ORDER BY
    DNUM ASC,
    VOLUME DESC;
```

В результате получим таблицу, в которой строки идут в порядке возрастания значения поля DNUM, а строки, с одинаковым значением DNUM идут в порядке убывания значения поля VOLUME:

PNUM	DNUM	VOLUME
3	1	1000
2	1	150
1	1	100
2	2	250
1	2	200
1	3	300

Замечание. Если явно не указаны ключевые слова ASC или DESC, то по умолчанию принимается упорядочение по возрастанию (ASC).

Отбор данных из нескольких таблиц

Пример 13. Естественное соединение таблиц (способ 1 - явное указание условий соединения):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P, PD
  WHERE P.PNUM = PD.PNUM;
```

В результате получим новую таблицу, в которой строки с данными о поставщиках соединены со строками с данными о поставках деталей:

PNUM	PNAME	DNUM	VOLUME
1	Иванов	1	100
1	Иванов	2	200
1	Иванов	3	300
2	Петров	1	150
2	Петров	2	250
3	Сидоров	1	1000

Замечание. Соединяемые таблицы перечислены в разделе FROM оператора, условие соединения приведено в разделе WHERE. Раздел WHERE, помимо условия соединения таблиц, может также содержать и условия отбора строк.

Пример 14. Естественное соединение таблиц (способ 2 - ключевые слова ***JOIN… USING…***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P JOIN PD USING PNUM;
```

Замечание. Ключевое слово USING позволяет *явно указать*, по каким из *общих* колонок таблиц будет производиться соединение.

Пример 15. Естественное соединение таблиц (способ 3 - ключевое слово ***NATURAL JOIN***):

```
SELECT
    P.PNUM,
    P.PNAME,
    PD.DNUM,
    PD.VOLUME
  FROM P NATURAL JOIN PD;
```

Замечание. В разделе FROM не указано, по каким полям производится соединение. NATURAL JOIN автоматически соединяет *по всем одинаковым полям* в таблицах.

Пример 16. Естественное соединение трех таблиц:

```
SELECT
    P.PNAME,
    D.DNAME,
    PD.VOLUME
  FROM
    P NATURAL JOIN PD NATURAL JOIN D;
```

В результате получим следующую таблицу:

PNAME	DNAME	VOLUME
Иванов	Болт	100
Иванов	Гайка	200
Иванов	Винт	300
Петров	Болт	150
Петров	Гайка	250
Сидоров	Болт	1000

Пример 17. Прямое произведение таблиц:

```
SELECT
    P.PNUM,
    P.PNAME,
    D.DNUM,
    D.DNAME
  FROM P, D;
```

В результате получим следующую таблицу:

PNUM	PNAME	DNUM	DNAME
1	Иванов	1	Болт
1	Иванов	2	Гайка
1	Иванов	3	Винт
2	Петров	1	Болт
2	Петров	2	Гайка
2	Петров	3	Винт
3	Сидоров	1	Болт
3	Сидоров	2	Гайка
3	Сидоров	3	Винт

Замечание. Т.к. не указано условие соединения таблиц, то *каждая* строка первой таблицы соединится с *каждой* строкой второй таблицы.

Пример 18. Соединение таблиц по произвольному условию. Рассмотрим таблицы поставщиков и деталей, которыми присвоен некоторый статую (см. пример 8 из предыдущей главы):

PNUM	PNAME	PSTATUS
1	Иванов	4
2	Петров	1
3	Сидоров	2

Таблица 1 Отношение P (Поставщики)

DNUM	DNAME	DSTATUS
1	Болт	3
2	Гайка	2
3	Винт	1

Таблица 2 Отношение D (Детали)

Ответ на вопрос "какие поставщики имеют право поставлять какие детали?" дает следующий запрос:

```
SELECT
    P.PNUM,
    P.PNAME,
    P.PSTATUS,
    D.DNUM,
    D.DNAME,
    D.DSTATUS
  FROM P, D
  WHERE P.PSTATUS >= D.DSTATUS;
```

В результате получим следующую таблицу:

PNUM	PNAME	PSTATUS	DNUM	DNAME	DSTATUS
1	Иванов	4	1	Болт	3
1	Иванов	4	2	Гайка	2
1	Иванов	4	3	Винт	1
2	Петров	1	3	Винт	1
3	Сидоров	2	2	Гайка	2
3	Сидоров	2	3	Винт	1

Использование имен корреляции (алиасов, псевдонимов)

Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются ***имена корреляции*** (***алиасы***, ***псевдонимы***), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени таблицы. Имена корреляции должны использоваться в качестве префикса перед именем столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результатирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

Пример 19. Отобрать все пары поставщиков таким образом, чтобы первый поставщик в паре имел статус, больший статуса второго поставщика:

```
SELECT
    P1.PNAME AS PNAME1,
    P1.PSTATUS AS PSTATUS1,
    P2.PNAME AS PNAME2,
    P2.PSTATUS AS PSTATUS2
  FROM
    P P1, P P2
  WHERE P1.PSTATUS1 > P2.PSTATUS2;
```

В результате получим следующую таблицу:

PNAME1	PSTATUS1	PNAME2	PSTATUS2
Иванов	4	Петров	1
Иванов	4	Сидоров	2
Сидоров	2	Петров	1

Пример 20. Рассмотрим ситуацию, когда некоторые поставщики (назовем их контрагенты) могут выступать как в качестве поставщиков деталей, так и в качестве получателей. Таблицы, хранящие данные могут иметь следующий вид:

Номер контрагента NUM	Наименование контрагента NAME
1	Иванов
2	Петров
3	Сидоров

Таблица 3 Отношение CONTRAGENTS

Номер детали DNUM	Наименование детали DNAME
1	Болт
2	Гайка
3	Винт

Таблица 4 Отношение DETAILS (Детали)

Номер поставщика PNUM	Номер получателя CNUM	Номер детали DNUM	Поставляемое количество VOLUME
1	2	1	100
1	3	2	200
1	3	3	300
2	3	1	150
2	3	2	250
3	1	1	1000

Таблица 5 Отношение CD (Поставки)

В таблице CD (поставки) поля PNUM и CNUM являются внешними ключами, ссылающимися на потенциальный ключ NUM в таблице CONTRAGENTS.

Ответ на вопрос "кто кому что в каком количестве поставляет" дается следующим запросом:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS,
    CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM AND
    D.DNUM = CD.DNUM;
```

В результате получим следующую таблицу:

Наименование поставщика PNAME	Наименование получателя CNAME	Наименование детали DNAME	Поставляемое количество VOLUME
Иванов	Петров	Болт	100
Иванов	Сидоров	Гайка	200
Иванов	Сидоров	Винт	300
Петров	Сидоров	Болт	150
Петров	Сидоров	Гайка	250
Сидоров	Иванов	Болт	1000

Замечание. Этот же запрос может быть выражен очень большим количеством способов, например, так:

```
SELECT
    P.NAME AS PNAME,
    C.NAME AS CNAME,
    DETAILS.DNAME,
    CD.VOLUME
  FROM
    CONTRAGENTS P,
    CONTRAGENTS C,
    DETAILS NATURAL JOIN CD
  WHERE
    P.NUM = CD.PNUM AND
    C.NUM = CD.CNUM;
```

Использование агрегатных функций в запросах

Пример 21. Получить общее количество поставщиков (ключевое слово ***COUNT***):

```
SELECT COUNT(*) AS N
  FROM P;
```

В результате получим таблицу с одним столбцом и одной строкой, содержащей количество строк из таблицы P:

N
3

Пример 22. Получить общее, максимальное, минимальное и среднее количества поставляемых деталей (ключевые слова ***SUM***, ***MAX***, ***MIN***, ***AVG***):

```
SELECT
    SUM(PD.VOLUME) AS SM,
    MAX(PD.VOLUME) AS MX,
    MIN(PD.VOLUME) AS MN,
    AVG(PD.VOLUME) AS AV
  FROM PD;
```

В результате получим следующую таблицу с одной строкой:

SM	MX	MN	AV
2000	1000	100	333.33333333

Использование агрегатных функций с группировками

Пример 23. Для каждой детали получить суммарное поставляемое количество (ключевое слово ***GROUP BY***…):

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Этот запрос будет выполняться следующим образом. Сначала строки исходной таблицы будут сгруппированы так, чтобы в каждую группу попали строки с одинаковыми значениями DNUM. Потом внутри каждой группы будет просуммировано поле VOLUME. От каждой группы в результатирующую таблицу будет включена одна строка:

DNUM	SM
1	1250
2	450
3	300

Замечание. В списке отбираемых полей оператора SELECT, содержащего раздел GROUP BY можно включать *только* агрегатные функции и поля, *которые входят в условие группировки*. Следующий запрос выдаст синтаксическую ошибку:

```
SELECT
    PD.PNUM,
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM;
```

Причина ошибки в том, что в список отбираемых полей включено поле PNUM, которое *не входит* в раздел GROUP BY. И действительно, в каждую полученную группу строк может входить несколько строк с *различными* значениями поля PNUM. Из каждой группы строк будет сформировано по одной итоговой строке. При этом нет однозначного ответа на вопрос, какое значение выбрать для поля PNUM в итоговой строке.

Замечание. Некоторые диалекты SQL не считают это за ошибку. Запрос будет выполнен, но предсказать, какие значения будут внесены в поле PNUM в результатирующей таблице, невозможно.

Пример 24. Получить номера деталей, суммарное поставляемое количество которых превосходит 400 (ключевое слово ***HAVING***…):

Замечание. Условие, что суммарное поставляемое количество должно быть больше 400 не может быть сформулировано в разделе WHERE, т.к. в этом разделе нельзя использовать агрегатные функции. Условия, использующие агрегатные функции должны быть размещены в специальном разделе HAVING:

```
SELECT
    PD.DNUM,
    SUM(PD.VOLUME) AS SM
  GROUP BY PD.DNUM
  HAVING SUM(PD.VOLUME) > 400;
```

В результате получим следующую таблицу:

DNUM	SM
1	1250
2	450

Замечание. В одном запросе могут встретиться как условия отбора строк в разделе WHERE, так и условия отбора групп в разделе HAVING. Условия отбора групп нельзя перенести из раздела HAVING в раздел WHERE. Аналогично и условия отбора строк нельзя перенести из раздела WHERE в раздел HAVING, за исключением условий, включающих поля из списка группировки GROUP BY.

Использование подзапросов

Очень удобным средством, позволяющим формулировать запросы более понятным образом, является возможность использования подзапросов, вложенных в основной запрос.

Пример 25. Получить список поставщиков, статус которых меньше максимального статуса в таблице поставщиков (сравнение с подзапросом):

```
SELECT *
  FROM P
  WHERE P.STATYS <
    (SELECT MAX(P.STATUS)
        FROM P);
```

Замечание. Т.к. поле P.STATUS сравнивается с результатом подзапроса, то подзапрос должен быть сформулирован так, чтобы возвращать таблицу, состоящую *ровно из одной строки и одной колонки*.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить максимальное значение статуса.
2. Просканировать таблицу поставщиков P, каждый раз сравнивая значение статуса поставщика с результатом подзапроса, и отобрать только те строки, в которых статус меньше максимального.

Пример 26. Использование предиката ***IN***. Получить список поставщиков, поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE P.PNUM IN
    (SELECT DISTINCT PD.PNUM
        FROM PD
        WHERE PD.DNUM = 2);
```

Замечание. В данном случае вложенный подзапрос может возвращать таблицу, содержащую несколько строк.

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Выполнить *один раз* вложенный подзапрос и получить список номеров поставщиков, поставляющих деталь номер 2.
2. Просканировать таблицу поставщиков P, каждый раз проверяя, содержится ли номер поставщика в результате подзапроса.

Пример 27. Использование предиката ***EXIST***. Получить список поставщиков, поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE EXIST
    (SELECT *
        FROM PD
        WHERE
            PD.PNUM = P.PNUM AND
            PD.DNUM = 2);
```

Замечание. Результат выполнения запроса будет эквивалентен результату следующей последовательности действий:

1. Просканировать таблицу поставщиков P, *каждый раз выполняя подзапрос* с новым значением номера поставщика, взятым из таблицы P.
2. В результат запроса включить только те строки из таблицы поставщиков, для которых вложенный подзапрос вернул непустое множество строк.

Замечание. В отличие от двух предыдущих примеров, вложенный подзапрос содержит параметр (внешнюю ссылку), передаваемый из основного запроса - номер поставщика P.PNUM. Такие подзапросы называются ***коррелируемыми*** (***correlated***). Внешняя ссылка может принимать различные значения для каждой строки-кандидата, оцениваемого с помощью подзапроса, поэтому подзапрос должен выполняться заново для каждой строки, отбираемой в основном запросе. Такие подзапросы характерны для предиката EXIST, но могут быть использованы и в других подзапросах.

Замечание. Может показаться, что запросы, содержащие коррелируемые подзапросы будут выполняться медленнее, чем запросы с некоррелируемыми подзапросами. На самом деле это не так, т.к. то, как пользователь, сформулировал запрос, *не определяет*, как этот запрос будет выполняться. Язык SQL является непроцедурным, а декларативным. Это значит, что пользователь, формулирующий запрос, просто описывает, *каким должен быть результат запроса*, а как этот результат будет получен - за это отвечает сама СУБД.

Пример 28. Использование предиката ***NOT EXIST***. Получить список поставщиков, не поставляющих деталь номер 2:

```
SELECT *
  FROM P
  WHERE NOT EXIST
    (SELECT *
        FROM PD
        WHERE
            PD.PNUM = P.PNUM AND
            PD.DNUM = 2);
```

Замечание. Также как и в предыдущем примере, здесь используется коррелируемый подзапрос. Отличие в том, что в основном запросе будут отобраны те строки из таблицы поставщиков, для которых вложенный подзапрос не выдаст ни одной строки.

Пример 29. Получить имена поставщиков, поставляющих все детали:

```
SELECT DISTINCT PNAME
```

```
  FROM P
  WHERE NOT EXIST
    (SELECT *
        FROM D
        WHERE NOT EXIST
          (SELECT *
              FROM PD
              WHERE
                  PD.DNUM = D.DNUM AND
                  PD.PNUM = P.PNUM));
```

Замечание. Данный запрос содержит два вложенных подзапроса и реализует реляционную операцию *деления отношений*.

Самый внутренний подзапрос параметризован двумя параметрами (D.DNUM, P.PNUM) и имеет следующий смысл: отобрать все строки, содержащие данные о поставках поставщика с номером PNUM детали с номером DNUM. Отрицание NOT EXIST говорит о том, что данный поставщик не поставляет данную деталь. Внешний к нему подзапрос, сам являющийся вложенным и параметризованным параметром P.PNUM, имеет смысл: отобрать список деталей, которые не поставляются поставщиком PNUM. Отрицание NOT EXIST говорит о том, что для поставщика с номером PNUM не должно быть деталей, которые не поставлялись бы этим поставщиком. Это в точности означает, что во внешнем запросе отбираются только поставщики, поставляющие все детали.

Использование объединения, пересечения и разности

Пример 30. Получить имена поставщиков, имеющих статус, больший 3 или поставляющих хотя бы одну деталь номер 2 (объединение двух подзапросов - ключевое слово ***UNION***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  UNION
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
        PD.DNUM = 2;
```

Замечание. Результатирующие таблицы объединяемых запросов должны быть совместимы, т.е. иметь одинаковое количество столбцов и одинаковые типы столбцов в порядке их перечисления. *Не требуется*, чтобы объединяемые таблицы имели бы одинаковые имена колонок. Это отличает операцию объединения запросов в SQL от операции объединения в реляционной алгебре. Наименования колонок в результатирующем запросе будут автоматически взяты из результата первого запроса в объединении.

Пример 31. Получить имена поставщиков, имеющих статус, больший 3 и одновременно поставляющих хотя бы одну деталь номер 2 (пересечение двух подзапросов - ключевое слово ***INTERSECT***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
  INTERSECT
```

```
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
         PD.DNUM = 2;
```

Пример 32. Получить имена поставщиков, имеющих статус, больший 3, за исключением тех, кто поставляет хотя бы одну деталь номер 2 (разность двух подзапросов - ключевое слово ***EXCEPT***):

```
SELECT P.PNAME
  FROM P
  WHERE P.STATUS > 3
EXCEPT
  SELECT P.PNAME
    FROM P, PD
    WHERE P.PNUM = PD.PNUM AND
         PD.DNUM = 2;
```

Порядок выполнения оператора SELECT

Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

Стадия 1. Выполнение одиночного оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:

Шаг 1 (FROM). Вычисляется прямое декартовое произведение всех таблиц, указанных в обязательном разделе FROM. В результате шага 1 получаем таблицу A.

Шаг 2 (WHERE). Если в операторе SELECT присутствует раздел WHERE, то сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат. Если раздел WHERE опущен, то сразу переходим к шагу 3. Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой. В результате шага 2 получаем таблицу B.

Шаг 3 (GROUP BY). Если в операторе SELECT присутствует раздел GROUP BY, то строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки, приведенным в разделе GROUP BY. Если раздел GROUP BY опущен, то сразу переходим к шагу 4. В результате шага 3 получаем таблицу C.

Шаг 4 (HAVING). Если в операторе SELECT присутствует раздел HAVING, то группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются. Если раздел HAVING опущен, то сразу переходим к шагу 5. В результате шага 4 получаем таблицу D.

Шаг 5 (SELECT). Каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом. Вычисляются все скалярные выражения, указанные в разделе SELECT. По пра-

вилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы. Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT. Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что имеется всего одна группа. Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что имеется столько групп, сколько строк отобрано к данному моменту. В результате шага 5 получаем таблицу E, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.

Стадия 3. Упорядочение результата

Если в операторе SELECT присутствует раздел ORDER BY, то строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения, приведенном в разделе ORDER BY.

Как на самом деле выполняется оператор SELECT

Если внимательно рассмотреть приведенный выше концептуальный алгоритм вычисления результата оператора SELECT, то сразу понятно, что выполнять его непосредственно в таком виде чрезвычайно накладно. Даже на самом первом шаге, когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.

На самом деле в РСУБД имеется ***оптимизатор***, функцией которого является нахождение такого *оптимального алгоритма* выполнения запроса, который гарантирует получение правильного результата.

Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов:

Шаг 1 (Синтаксический анализ). Поступивший запрос подвергается синтаксическому анализу. На этом шаге определяется, правильно ли вообще (с точки зрения синтаксиса SQL) сформулирован запрос. В ходе синтаксического анализа вырабатывается некоторое внутренне представление запроса, используемое на последующих шагах.

Шаг 2 (Преобразование в каноническую форму). Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании к канонической форме используются как синтаксические, так и семантические преобразования. Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить новое внутренне представление запроса, синтаксически *эквивалентное* исходному, но стандартное в некотором смысле. Семантические преобразования используют дополнительные знания, которыми владеет система, например, ограничения целостности. В результате семантических преобразований получается запрос, синтаксически *не эквивалентный* исходному, но дающий *тот же самый результат*.

Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана). На этом шаге оптимизатор генерирует множество возможных планов выполнения запроса. Каждый план строится как комбинация низкоуровневых процедур доступа к данным из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, статистических данных о распределении значений в таблицах, и т.п. Стоимость плана это, как правило, сумма стоимостей выполнения отдельных низкоуровневых процедур, которые используются для его выполнения. В стоимость выполнения отдельной процедуры могут входить оценки количества обращений к дискам, степень загруженности процессора и другие параметры.

Шаг 4. (Выполнение плана запроса). На этом шаге план, выбранный на предыдущем шаге, передается на реальное выполнение.

Во многом качество конкретной СУБД определяется качеством ее оптимизатора. Хороший оптимизатор может повысить скорость выполнения запроса на несколько порядков. Качество оптимизатора определяется тем, какие методы преобразований он может использовать, какой статистической и иной информацией о таблицах он располагает, какие методы для оценки стоимости выполнения плана он знает.

Оператор деления

Реляционная алгебра: $A(X,Y) \text{ DEVIDBY } B(Y)$

Оператор SQL:

```
SELECT DISTINCT A.X
  FROM A
  WHERE NOT EXIST
    (SELECT *
        FROM B
        WHERE NOT EXIST
          (SELECT *
              FROM A A1
              WHERE
                  A1.X = A.X AND
                  A1.Y = B.Y));
```

Замечание. Оператор SQL, реализующий деление отношений трудно запомнить, поэтому дадим пример эквивалентного преобразования выражений, представляющих суть запроса.

Пусть отношение A содержит данные о поставках деталей, отношение B содержит список всех деталей, которые могут поставляться. Атрибут X является номером поставщика, атрибут Y является номером детали.

Разделить отношение A на отношение B означает в данном примере "отобрать номера поставщиков, которые поставляют *все* детали".

Преобразуем текст выражения:

"Отобрать номера поставщиков, которые поставляют *все* детали" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* непоставляемых деталей в таблице B" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* тех номеров деталей из таблицы B, которые *не поставляются* этим поставщиком" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых *не существует* тех номеров деталей из таблицы B, для которых *не существует* записей о поставках в таблице A для этого поставщика и этой детали".

Последнее выражение дословно переводится на язык SQL. При переводе выражения на язык SQL нужно учесть, что во внутреннем подзапросе таблица A должна быть переименована, для того чтобы отличать ее от экземпляра этой же таблицы, используемой во внешнем запросе.

Транзакции и целостность баз данных

Это понятие не входит в реляционную модель данных, т.к. транзакции рассматриваются не только в реляционных СУБД, но и в СУБД других типов, а также и в других типах информационных систем.

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "*все или ничего*", т.е. либо транзакция выполняется целиком и переводит базу данных из одного *целостного состояния* в другое *целостное состояние*, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции). С этой точки зрения, транзакции важны как в многопользовательских, так и в однопользовательских системах. В однопользовательских системах транзакции - это логические единицы работы, после выполнения которых база данных остается *в целостном состоянии*. Транзакции также являются *единицами восстановления* данных после сбоев - восстанавливаясь, система ликвидирует следы транзакций, не успевших успешно завершиться в результате программного или аппаратного сбоя. Эти два свойства транзакций определяют атомарность (неделимость) транзакции. В многопользовательских системах, кроме того, транзакции служат для обеспечения *изолированной* работы отдельных пользователей - пользователям, одновременно работающим с одной базой данных, кажется, что они работают как бы в однопользовательской системе и не мешают друг другу.

Пример нарушения целостности базы

Для иллюстрации возможного нарушения целостности базы данных рассмотрим следующий пример:

Пример 1. Пусть имеется система, в которой хранятся данные о подразделениях и работающих в них сотрудниках. Список подразделений хранится в таблице DEPART(Dep_Id, Dep_Name, Dept_Kol), где Dept_Id - идентификатор подразделения, Dept_Name - наименование подразделения, Dept_Kol - количество сотрудников в подразделении. Список сотрудников хранится в таблице PERSON(Pers_Id, Pers_Name, Dept_Id), где Pers_Id - идентификатор сотрудника, Pers_Name - имя сотрудника, Dept_Id - идентификатор подразделения, в котором работает сотрудник:

Dept_Id	Dept_Name	Dept_Kol
1	Кафедра алгебры	3

2	Кафедра программирования	2

Таблица 1 DEPART

Pers_Id	Pers_Name	Dept_Id
1	Иванов	1
2	Петров	2
3	Сидоров	1
4	Пушников	2
5	Шарипов	1

Таблица 2 PERSON

Ограничение целостности этой базы данных состоит в том, что поле Dept_Kol не может заполняться произвольными значениями - это поле должно содержать количество сотрудников, реально числящихся в подразделении.

С учетом этого ограничения можно заключить, что вставка нового сотрудника в таблицу *не может быть выполнена одной операцией*. При вставке нового сотрудника необходимо одновременно увеличить значение поля Dept_Kol:

- Шаг 1. Вставить сотрудника в таблицу PERSON: INSERT INTO PERSON (6, Муфтахов, 1)
- Шаг 2. Увеличить значение поля Dept_Kol: UPDATE DEPART SET Dept=Dept+1 WHERE Dept_Id=1

Если после выполнения первой операции и до выполнения второй произойдет сбой системы, то реально будет выполнена только первая операция и база данных остается в нецелостном состоянии.

Понятие транзакции

Определение 1. ***Транзакция*** - это последовательность операторов манипулирования данными, выполняющаяся *как единое целое* (все или ничего) и переводящая базу данных *из одного целостного состояния в другое целостное состояние*.

Транзакция обладает четырьмя важными свойствами, известными как ***свойства АСИД***:

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

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда COMMIT WORK (зафиксировать транзакцию).
- Подана команда ROLLBACK WORK (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

Команда COMMIT WORK завершает текущую транзакцию и автоматически начинает новую транзакцию. При этом гарантируется, что результаты работы завершенной транзакции фиксируются, т.е. сохраняются в базе данных.

Замечание. Некоторые системы (например, Visual FoxPro), требуют подать явную команду BEGIN TRANSACTION для того, чтобы начать новую транзакцию.

Команда ROLLBACK WORK приводит к тому, что все изменения, сделанные текущей транзакцией откатываются, т.е. отменяются так, *как будто их вообще не было*. При этом автоматически начинается новая транзакция.

При отсоединении пользователя от СУБД происходит автоматическая фиксация транзакций.

При сбое системы происходят более сложные процессы. Кратко суть их сводится к тому, что при последующем запуске системы происходит анализ выполнявшихся до момента сбоя транзакций. Те транзакции, для которых была подана команда COMMIT WORK, но *результаты работы которых не были занесены в базу данных* выполняются снова (накатываются). Те транзакции, для которых не была подана команда COMMIT WORK, откатываются. Более подробно восстановление после сбоев рассматривается далее.

Свойства АСИД транзакций не всегда выполняются в полном объеме. Особенно это относится к свойству И (изоляция). В идеале, транзакции разных пользователей не должны мешать друг другу, т.е. они должны выполняться так, чтобы у пользователя создавалась иллюзия, что он в системе один. Простейший способ обеспечить абсолютную изолированность состоит в том, чтобы выстроить транзакции в очередь и выполнять их строго одну за другой. Очевидно, при этом теряется эффективность работы системы. Поэтому реально одновременно выполняется несколько транзакций (смесь транзакций). Различается несколько уровней изоляции транзакций. На низшем уровне изоляции транзакции могут реально мешать друг другу, на высшем они полностью изолированы. За большую изоляцию транзакций приходится платить большими накладными расходами системы и замедлением работы. Пользователи или администратор системы могут по своему усмотрению задавать различные уровни всех или отдельных транзакций. Более подробно изоляция транзакций рассматривается в следующей главе.

Свойство Д (долговечность) также не является абсолютными свойством, т.к. некоторые системы допускают вложенные транзакции. Если транзакция Б запущена внутри транзакции А, и для транзакции Б подана команда COMMIT WORK, то фиксация данных транзакции Б является условной, т.к. внешняя транзакция А может откатиться. Результаты работы внутренней транзакции Б будут окончательно зафиксированы только если будет зафиксирована внешняя транзакция А.

Ограничения целостности

Свойство (С) - согласованность транзакций определяется наличием понятия согласованности базы данных.

Определение 2. ***Ограничение целостности*** - это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

Примерами ограничений целостности могут служить следующие утверждения:

Пример 2. Возраст сотрудника не может быть меньше 18 и больше 65 лет.

Пример 3. Каждый сотрудник имеет уникальный табельный номер.

Пример 4. Сотрудник обязан числиться в одном отделе.

Пример 5. Сумма накладной обязана равняться сумме произведений цен товаров на количество товаров для всех товаров, входящих в накладную.

Как видно из этих примеров, некоторые из ограничений целостности являются ограничениями реляционной модели данных (см. гл. 3). Пример 3 представляет ограничение, реализующее целостность сущности. Пример 4 представляет ограничение, реализующее ссылочную целостность. Другие ограничения являются достаточно произвольными утверждениями (примеры 2 и 5). Любое ограничение целостности является *семантическим* понятием, т.е. появляется как следствие определенных свойств объектов предметной области и/или их взаимосвязей.

Определение 3. База данных находится в ***согласованном (целостном) состоянии***, если выполнены (удовлетворены) все ограничения целостности, определенные для базы данных.

В данном определении важно подчеркнуть, что должны быть выполнены не все вообще ограничения предметной области, а только те, которые *определены* в базе данных. Для этого необходимо, чтобы СУБД *обладала* развитыми средствами поддержки ограничений целостности. Если какая-либо СУБД не может отобразить все необходимые ограничения предметной области, то такая база данных хотя и будет находиться в целостном состоянии с точки зрения СУБД, но это состояние не будет правильным с точки зрения пользователя.

Таким образом, согласованность базы данных есть формальное свойство базы данных. База данных не понимает "смысла" хранимых данных. "Смыслом" данных для СУБД является весь набор ограничений целостности. Если все ограничения выполнены, то СУБД считает, что данные корректны.

Вместе с понятием целостности базы данных возникает понятие ***реакции системы на попытку нарушения целостности***. Система должна не только проверять, не нарушаются ли ограничения в ходе выполнения различных операций, но и должным образом реагировать, если операция приводит к нарушению целостности. Имеется два типа реакции на попытку нарушения целостности:

1. *Отказ* выполнить "незаконную" операцию.
2. Выполнение *компенсирующих* действий.

Например, если система знает, что в поле "Возраст_Сотрудника" должны быть целые числа в диапазоне от 18 до 65, то система отвергает попытку ввести значение возраста 66. При этом может генерироваться какое-нибудь сообщение для пользователя.

В противоположность этому, в примере 1 система допускает вставку записи о новом сотруднике (что приводит к нарушению целостности базы данных), но *автоматически* производит компенсирующие действия, изменяя значение поля Dept_Kol в таблице DEPART.

Работу системы по проверке ограничений можно изобразить на следующем рисунке:

В некоторых случаях система может не выполнять проверку на нарушение ограничений, а сразу выполнять компенсирующие операции. Действительно, в примере 1 при вставке или удалении сотрудника проверку производить *не нужно*, т.к. результаты ее известны заранее - ограничение *обязательно* будет нарушено. В этом случае необходимо сразу приступать к компенсированию возникшего нарушения.

Литература и ссылки в Интернет

1. http://citforum.ru/database/osbd/contents.shtml
2. ВВЕДЕНИЕ В СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ. http://www.do.sibsutis.ru/bakalavr/sem7/course90/index1.htm

3.Основы современных баз данных. *С.Д. Кузнецов, информационно-аналитические материалы* http://www.lcard.ru/~nail/database/osbd/contents.htm

4. Введение в реляционные базы данных. Автор С.Д. Кузнецов http://www.intuit.ru/department/database/rdbintro/2/