1.Определение информационной системы, БД. СУБД. Основные функции СУБД

ИС- представляет собой систему, реализующую автоматич сбор, хранение, обработку и представление информации.

БД- проименованная совокупность данных, организованная по определенным правилам, которые включают общие принципы описания, хранения и манипуляции информации.

СУБД – программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).

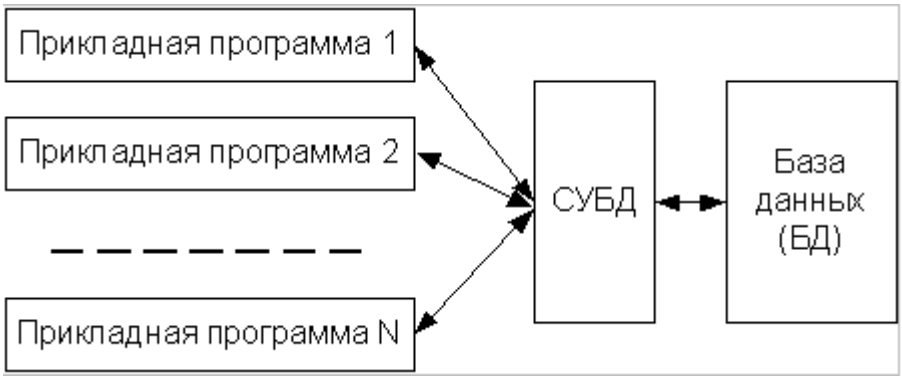

Рис. 2.1. Обеспечение независимости прикладных программ и базы данных

Определим еще одно понятие.

Банк данных – система языковых, алгоритмических, программных, технических и организационных средств поддержки интегрированной совокупности данных, а также сами эти данные, представленные в виде баз данных.

Перечислим основные функции системы управления базами данных.

1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки.

Как правило, создание структуры базы данных происходит в режиме диалога. СУБД последовательно запрашивает у пользователя необходимые данные. В большинстве современных СУБД база данных представляется в виде совокупности таблиц. Рассматриваемая функция позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы.

2. Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).

Такие возможности в СУБД представляются либо на основе использования специального языка программирования, входящего в состав СУБД, либо с помощью графического интерфейса.

3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).

Важнейшим свойством СУБД является возможность поддерживать два независимых взгляда на базу данных – "взгляд пользователя", воплощаемый в логическом представлении данных, и его отражения в прикладных программах; и "взгляд системы" – физическое представление данных в памяти ЭВМ. Обеспечение логической независимости данных предоставляет возможность изменения (в определенных пределах) логического представления базы данных без необходимости изменения физических структур хранения данных.

4. Защита логической целостности базы данных.

5. Защита физической целостности.

6. Управление полномочиями пользователей на доступ к базе данных.

Разные пользователи могут иметь разные полномочия по работе с данными (некоторые данные должны быть недоступны; определенным пользователям не разрешается обновлять данные и т.п.). В СУБД предусматриваются механизмы разграничения полномочий доступа, основанные либо на принципах паролей, либо на описании полномочий.

8. Управление ресурсами среды хранения.

БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память) и удаляются данные (освобождается память). СУБД выделяет ресурсы памяти для новых данных, перераспределяет освободившуюся память, организует ведение очереди запросов к внешней памяти и т.п.

9. Поддержка деятельности системного персонала.

2. Основные понятия иерархической модели данных

Это также одна из наиболее ранних моделей данных. Реализация групповых отношений в иерархической модели, как и в сетевой, может осуществляться с помощью указателей и представляется в виде графа. Однако, в отличие от сетевой модели, здесь существует ряд принципиальных особенностей.

Групповые отношения являются отношениями соподчиненности. Группа (запись) – владелец отношения имеет подчиненные группы – члены отношений. Исходная группа называется предком, подчиненная – потомком.

Групповые отношения образуют иерархическую структуру, которую можно описать как ориентированный граф следующего вида:

1. имеется единственная особая вершина (соответствующая группе), называемая корнем, в которую не заходит ни одно ребро (группа не имеет предков);
2. во все остальные вершины входит только одно ребро (все остальные группы имеют одного предка), а исходит произвольное количество ребер (группы имеют произвольное количество потомков);
3. отсутствуют циклы.

Иерархическая модель данных может представлять совокупность нескольких деревьев. В терминологии иерархической модели деревья, описывающие структуру данных, называются деревьями описания данных, а сами структурированные данные (база данных) – деревьями данных.

Особенностью реализации операций поиска в иерархической модели является то, что операция всегда начинает поиск с корневой вершины и специфицирует иерархический путь (последовательность связанных вершин) от корня до вершины, экземпляры которой удовлетворяют условиям поиска.

Необходимо отметить, что программы, реализующие операции иерархической модели, существенно проще, чем аналогичные программы для сетевой модели, т.к. здесь много легче осуществлять навигацию по структуре. Целесообразность появления иерархической модели обусловлена, конечно, тем, что большинство организационных систем реального мира имеют иерархическую структуру (административное деление страны, организационная структура предприятия и т.п.). Соответствующее концептуальное представление также будет иметь иерархическую структуру и естественным образом может быть описано в терминах иерархической модели. В качестве недостатков иерархической модели можно назвать вышеуказанные недостатки сетевой.

СУБД, поддерживающие иерархическую модель, достаточно широко использовались на вычислительных системах IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать IMS, ОКА и широко тиражируемую в СССР отечественную разработку ИНЕС. Примером иерархической СУБД для персональных ЭВМ является отечественная система НИКА (адаптация системы ИНЕС к IBM PC).

3. Основные понятия сетевой модели данных

Это одна из наиболее ранних моделей данных СУБД. Типовая сетевая модель данных была предложена рабочей группой по базам данных (Data Base Task Group – DBTG) системного комитета CODASYL (Conference of Data System Languages), основными функциями которого были анализ известных фирменных систем обработки управленческих данных с единых позиций и в единой терминологии, обобщение опыта организации таких систем и разработка рекомендаций по созданию соответствующих систем. Структура данных сетевой модели определяется в терминах раздела 6.1 (элемент, запись, группа, групповое отношение, файл, база данных).

Реализация групповых отношений в сетевой модели осуществляется с использованием специально вводимых дополнительных полей - указателей (адресов связи или ссылок), которые устанавливают связь между владельцем и членом группового отношения. Запись может состоять в отношениях разных типов (1:1, 1:N, M:N). Заметим, что если один из вариантов установления связи 1:1 очевиден (в запись – владелец отношения, поля которой соответствуют атрибутам сущности, включается дополнительное поле – указатель на запись – член отношения), то возможность представления связей 1:N и M:N таким же образом весьма проблематична. Поэтому наиболее распространенным способом организации связей в сетевых СУБД является введение дополнительного типа записей (и соответственно, дополнительного файла), полями которых являются указатели.

Рассмотрим для примера представление группового отношения M:N. В модель вводится дополнительная группа (дополнительный вид записей). Элементы этой записи представляют собой указатели на две исходные группы и указатели на экземпляры рассматриваемой дополнительной записи, связывающие их в список (цепь), соответствующий M и (или) N членам группового отношения (рис. 6.1.).

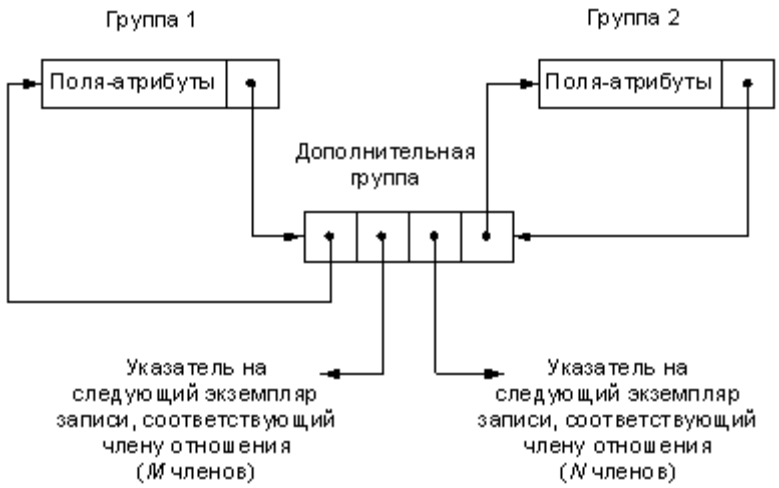

Рис. 6.1. Представление связей типа M:N

Представление связей 1:1, 1:M, N:1 является частным случаем связи типа M:N и осуществляется аналогично рассмотренному выше.

Заметим, что группа может быть членом более чем одного группового отношения. В этом случае вводится несколько дополнительных групп-указателей, а в группе – владельце отношений вводится несколько полей – указателей на дополнительные группы. Тогда множество записей (групп) и связей между ними образует некую сетевую структуру (ориентированный граф общего вида). Вершинами графа являются группы; дугами графа, направленными от владельца к члену группового отношения, – связи между группами.

Сетевая модель данных поддерживает все необходимые операции над данными, реализованные как действия со списковыми структурами. Сетевая модель данных является, вероятно, наиболее общей по возможностям представления концептуальной модели. По сути, любая ER-диаграмма без каких-либо изменений представляется средствами сетевой модели. К недостаткам сетевой модели обычно относят сложность получаемой на её основе концептуальной схемы и большую трудоемкость понимания соответствующей схемы внешним пользователем.

Наиболее существенным недостатком сетевой модели является "жесткость" получаемой концептуальной схемы. Связи закреплены в записях в виде указателей. При появлении новых аспектов использования этих же данных может возникнуть необходимость установления новых связей между ними. Это требует введения в записи новых указателей, т.е. изменения структуры БД, и, соответственно, переформирования всей базы данных.

СУБД, поддерживающие сетевую модель, широко использовались на вычислительных системах серии IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать IDMS, UNIBAD (БАНК), и их аналоги СЕДАН, СЕТОР. На персональных компьютерах сетевые СУБД не получили широкого распространения. Примером сетевой СУБД для персонального компьютера является db_VISTA III. Отметим, что система db_VISTA реализована на языке С и поэтому является переносимой. Система может эксплуатироваться на ПЭВМ типа IBM PC, SUN, Macintosh.

4. Структурная часть РМД

Описывает какие обьекты рассматриваются РМД

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 (для логических значений мы будем попеременно использовать обозначения истина и ложь или true и false). 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 (как, например, на рис. 2.1 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 на рис. 2.1 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 рис. 2.1. Заметим, что в общем случае, как это демонстрируют, в частности, рис. 2.1 и пример предыдущего абзаца, могут существовать такие кортежи tr, которые соответствуют Hr, но не входят в Br.

5.Целостная часть реляционной модели. Реализация условия целостности данных в современных СУБД

Целостная часть РМД описывает ограничения спец вида, которые должны выполняться для любых отношения в любых РМД:

- целостность сущностей
- ссылочная целостность

Напомним, что под целостностью базы данных понимается то, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную часть (правильная) информация. Поддержка целостности в реляционных БД основана на выполнении следующих требований.

1. Первое требование называется требованием целостности сущностей. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е., другими словами, любое отношение должно обладать определенным первичным ключом. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

2. Второе требование называется требованием целостности по ссылкам. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Связь между отношениями осуществляется с помощью миграции ключа.

Требование целостности по ссылкам или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа в ссылающемся отношении в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. (В Access для этого предназначена специальная реализация целочисленного поля – поле типа "Счетчик".) С целостностью по ссылкам дела обстоят несколько более сложно.

Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа.

6. Операции нарушающие ссылочную целостность(СЦ). Стратегии поддержания СЦ

Операции.

СЦ может нарушаться в результате выполнения операций, изменяющих состояние БД. Их 3:

- вставка
- обновление
- удаление кортежа в отношении.

В определении СЦ принимают участие 2 отношения(главный, подчиненный), т о получаем 6 вариантов операций:

	главный	Подчиненый
Вставка	-	+
Обновление	+	+
удаление	+	-

(плюс- СЦ нарушена)

Т о СЦ нарушается при выполнении 4 операций.

Стратегии.

Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

В развитых реляционных СУБД обычно можно выбрать способ поддержания целостности по ссылкам для каждой отдельной ситуации определения внешнего ключа. Конечно, для принятия такого решения необходимо анализировать требования конкретной прикладной области.

Заметим, что все современные СУБД поддерживают и целостность сущностей, и целостность по ссылкам, но позволяют пользователям выключать данные ограничения и, таким образом, строить базы данных, не соответствующие реляционной модели. Опыт показывает, что отход от основных положений реляционной модели приводит к краткосрочному выигрышу – алгоритмы становятся проще, но впоследствии серьезно усложняют задачу, особенно ее сопровождение.

7. Теоретико-множественные операции реляционной алгебры

Хотя в основе теоретико-множественной части реляционной алгебры Кодда лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями.

Операции объединения, пересечения, взятия разности. Совместимость по объединению

Начнем с операции объединения отношений (все, что будет сказано по поводу объединения, верно и для операций пересечения и взятия разности отношений). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Напомним, что в теории множеств:

- результатом объединения двух множеств A{a} и B{b} является такое множество C{c}, что для каждого c либо существует такой элемент a, принадлежащий множеству A, что c=a, либо существует такой элемент b, принадлежащий множеству B, что c=b ;
- пересечением множеств A и B является такое множество C{c}, что для любого c существуют такие элементы a, принадлежащий множеству A, и b, принадлежащий множеству B, что c=a=b ;
- разностью множеств A и B является такое множество C{c}, что для любого c существует такой элемент a, принадлежащий множеству A, что c=a, и не существует такой элемент b, принадлежащий B, что c=b.

Понятие *совместимости отношений по объединению*: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. В развернутой форме это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене (эта развернутая формулировка, вообще говоря, является излишней, но она пригодится нам в следующем абзаце).

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним, что если два отношения "почти" совместимы по объединению, т. е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа объединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

Операция расширенного декартова произведения и совместимость отношений относительно этой операции

Приведем более точное определение операции расширенного декартова произведения. Пусть имеются два отношения R1{a1, a2, …, an} и R2{b1, b2, …, bm}. Тогда результатом операции R1 TIMES R2 является отношение R{a1, a2, …, an, b1, b2, …, bm}, тело которого является множеством кортежей вида {ra1, ra2, …, ran, rb1, rb2, …, rbm} таких, что {ra1, ra2, …, ran} входит в тело R1, а {rb1, rb2, …, rbm} входит в тело R2 .

Два отношения *совместимы по взятию расширенного декартова произведения* в том и только в том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования к одному из этих отношений.

8. Специальные операции реляционной алгебры

В этом разделе мы несколько подробнее рассмотрим специальные реляционные операции реляционной алгебры, такие, как ограничение, проекция, соединение и деление.

Операция ограничения

Операция ограничения WHERE требует наличия двух операндов: ограничиваемого отношения и простого условия ограничения. Простое условие ограничения может иметь:

вид (a comp-op b), где a и b – имена атрибутов ограничиваемого отношения; атрибуты a и b должны быть определены на одном и том же домене, для значений базового типа данных которого поддерживается операция сравнения comp_op, или на базовых типах данных, над значениями которых можно выполнять эту операцию сравнения;

или вид (a comp-op const), где a – имя атрибута ограничиваемого отношения, a const – литерально заданная константа; атрибут a должен быть определен на домене или базовом типе, для значений которого поддерживается операция сравнения comp_op.

Операцией сравнения comp-op могут быть " = ", " ", " > ", " ", " < ", " ". Простые условия вычисляются в трехзначной логике (см. разд. "Реляционная модель данных", лекция 2), и в результате выполнения операции ограничения производится отношение, заголовок которого совпадает с заголовком отношения-операнда, а в тело входят те кортежи отношения-операнда, для которых значением условия ограничения является true. Тем самым, если в некоторых кортежах содержатся неопределенные значения, и по данной причине вычисление простого условия дает значение unknown, то эти кортежи не войдут в результирующее отношение.

Для обозначения вызова операции ограничения будем использовать конструкцию A WHERE comp, где A – ограничиваемое отношение, a comp – простое условие сравнения. Пусть comp1 и comp2 – два простых условия ограничения. Тогда по определению:

- A WHERE (comp1 AND comp2) обозначает то же самое, что и (A WHERE comp1) INTERSECT (A WHERE comp2) ;
- A WHERE (comp1 OR comp2) обозначает то же самое, что и (A WHERE comp1) UNION (A WHERE comp2) ;
- A WHERE NOT comp1 обозначает то же самое, что и A MINUS (A WHERE comp1).

Эти соглашения позволяют задействовать операции ограничения, в которых условием ограничения является произвольное булевское выражение, составленное из простых условий с использованием логических связок AND, OR, NOT и скобок.

Операция взятия проекции

Операция взятия проекции также требует наличия двух операндов – проецируемого отношения A и подмножества множества имен атрибутов, входящих в заголовок отношения A.

Результатом проекции отношения A на множество атрибутов {a1, a2, ..., an}(PROJECT A {a1, a2, ..., an}) является отношение с заголовком, определяемым множеством атрибутов

{a1, a2, ..., an}, и с телом, состоящим из кортежей вида <a1:v1, a2:v2, ..., an:vn> таких, что в отношении A имеется кортеж, атрибут a1 которого имеет значение v1, атрибут a2 имеет значение v2, ..., атрибут an имеет значение vn. Тем самым, при выполнении операции проекции выделяется "вертикальная" вырезка отношения-операнда с естественным уничтожением потенциально возникающих кортежей-дубликатов.

Заметим, что потенциальная потребность удаления дубликатов очень сильно усложняет реализацию операции проекции, поскольку в общем случае для удаления дубликатов требуется сортировка промежуточного результата операции. Основная сложность состоит в том, что этот промежуточный результат в общем случае может быть очень большим, и для сортировки требуется применять дорогостоящие алгоритмы внешней сортировки, выполняемые с применением обменов с внешней памятью. (Под "стоимостью" действия понимается время его выполнения.)

Операция соединения отношений

Общая операция соединения (называемая также соединением по условию) требует наличия двух операндов – соединяемых отношений и третьего операнда – простого условия. Пусть соединяются отношения A и B. Как и в случае операции ограничения, условие соединения comp имеет вид либо (a comp-op b), либо (a comp-op const), где a и b – имена атрибутов отношений A и B, const – литерально заданная константа, и comp-op – допустимая в данном контексте операция сравнения.

Тогда по определению результатом операции соединения A JOIN B WHERE comp совместимых по взятию расширенного декартова произведения отношений A и B является отношение, получаемое путем выполнения операции ограничения по условию comp расширенного декартова произведения отношений A и B .

Если тщательно осмыслить это определение, то станет ясно, что в общем случае применение условия соединения существенно уменьшит мощность результата промежуточного декартова произведения отношений-операндов только в том случае, если условие соединения имеет вид (a comp-op b), где a и b – имена атрибутов разных отношений-операндов. Поэтому на практике обычно считают реальными операциями соединения именно те операции, которые основываются на условии соединения приведенного вида.

9. Понятие архитектуры клиент – сервер

Использование технологии " клиент – сервер " предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети).

Так, архитектура " клиент – сервер " разделяет функции приложения пользователя (называемого клиентом) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД, на структурном языке запросов SQL (Structured Query Languague), являющемся промышленным стандартом в мире реляционных БД. Удаленный сервер принимает запрос и переадресует его SQL-серверу БД. SQL-сервер – специальная программа, управляющая удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Архитектура системы представлена на рис. 3.3.

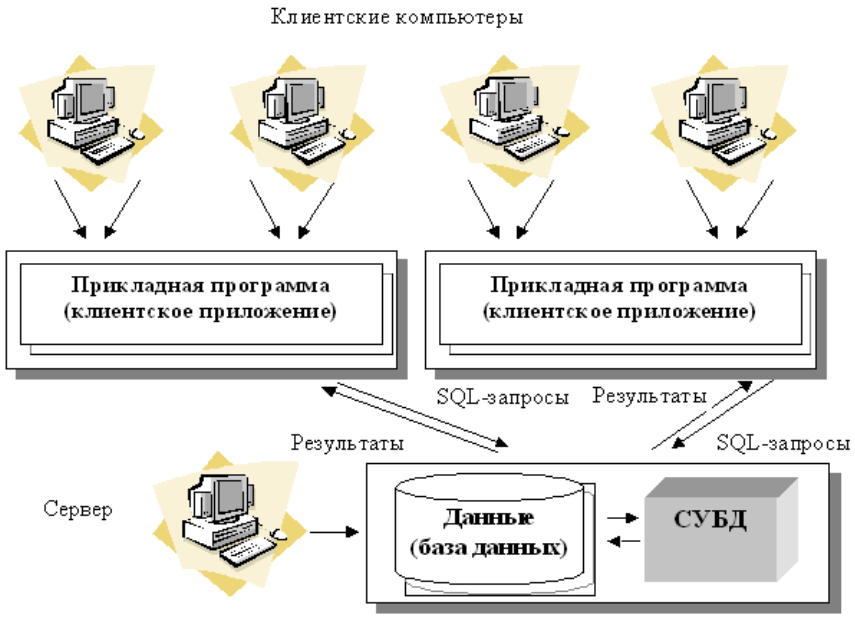

Рис. 3.3. Архитектура "клиент – сервер"

Рассмотрим, как выглядит разграничение функций между сервером и клиентом.

Функции приложения-клиента:

- Посылка запросов серверу.
- Интерпретация результатов запросов, полученных от сервера.
- Представление результатов пользователю в некоторой форме (интерфейс пользователя).

Функции серверной части:

- Прием запросов от приложений-клиентов.

- Интерпретация запросов.
- Оптимизация и выполнение запросов к БД.
- Отправка результатов приложению-клиенту.
- Обеспечение системы безопасности и разграничение доступа.
- Управление целостностью БД.
- Реализация стабильности многопользовательского режима работы.

В архитектуре " клиент – сервер " работают так называемые "промышленные" СУБД. Промышленными они называются из-за того, что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия, организации, банка. К разряду промышленных СУБД принадлежат MS SQL Server, Oracle, Gupta, Informix, Sybase, DB2, InterBase и ряд других [[6]].

Как правило, SQL-сервер обслуживается отдельным сотрудником или группой сотрудников (администраторы SQL-сервера). Они управляют физическими характеристиками баз данных, производят оптимизацию, настройку и переопределение различных компонентов БД, создают новые БД, изменяют существующие и т.д., а также выдают привилегии (разрешения на доступ определенного уровня к конкретным БД, SQL-серверу) различным пользователям [[6]].

Рассмотрим основные достоинства данной архитектуры по сравнению с архитектурой "файл-сервер":

1. Существенно уменьшается сетевой трафик.
2. Уменьшается сложность клиентских приложений (большая часть нагрузки ложится на серверную часть), а, следовательно, снижаются требования к аппаратным мощностям клиентских компьютеров.
3. Наличие специального программного средства – SQL-сервера – приводит к тому, что существенная часть проектных и программистских задач становится уже решенной.
4. Существенно повышается целостность и безопасность БД.

К числу недостатков можно отнести более высокие финансовые затраты на аппаратное и программное обеспечение, а также то, что большое количество клиентских компьютеров, расположенных в разных местах, вызывает определенные трудности со своевременным обновлением клиентских приложений на всех компьютерах-клиентах. Тем не менее, архитектура " клиент – сервер " хорошо зарекомендовала себя на практике, в настоящий момент существует и функционирует большое количество БД, построенных в соответствии с данной архитектурой.

10. Языки баз данных.

все операторы языка SQL разделяются на три составные части: DDL – язык определения данных, DCL – язык управления данными, DML – язык обработки данных.

Приведем примеры основных операторов из вышеуказанных частей (без описания синтаксиса). Описание синтаксиса операторов SQL можно посмотреть в многочисленных книгах по языку SQL, в меню "Справка" конкретных СУБД.

Операторы разграничения доступа пользователей к объектам базы данных (DCL).

GRANT – создание в системе безопасности записи, разрешающей пользователю работать с данными или выполнять определенные операции SQL.

DENY - создание в системе безопасности записи, запрещающей доступ для определенной учетной записи.

Операторы определения данных (язык DDL).

Соответствующие операторы предназначены для создания, удаления, изменения основных объектов модели данных реляционных СУБД: таблиц, представлений, индексов.

CREATE TABLE <имя> - создание новой таблицы в базе данных.

DROP TABLE <имя> - удаление таблицы из базы данных.

ALTER TABLE <имя> - изменение структуры существующей таблицы или ограничений целостности, задаваемых для данной таблицы.

При выполнении аналогичных операций с представлениями или индексами в указанных операторах вместо служебного слова TABLE записывается слово VIEW (представление) или слово INDEX (индекс)

Операторы манипулирования данными (язык DML).

Операторы DML работают с базой данных и используются для изменения данных и получения необходимых сведений.

SELECT – выборка строк, удовлетворяющих заданным условиям. Оператор реализует, в частности, такие операции реляционной алгебры как "селекция" и "проекция".

UPDATE – изменение значений определенных полей в строках таблицы, удовлетворяющих заданным условиям.

INSERT – вставка новых строк в таблицу.

DELETE – удаление строк таблицы, удовлетворяющих заданным условиям. Применение этого оператора учитывает принципы поддержки целостности, поэтому он не всегда может быть выполнен корректно.

11. Создание базы данных в среде MS SQL Server (Create database)

Процесс создания базы данных в системе SQL-сервера состоит из двух этапов: сначала организуется сама база данных, а затем принадлежащий ей журнал транзакций. Информация размещается в соответствующих файлах, имеющих расширения *.mdf (для базы данных) и *.ldf. (для журнала транзакций). В файле базы данных записываются сведения об основных объектах (таблицах, индексах, представлениях и т.д.), а в файле журнала транзакций – о процессе работы с транзакциями (контроль целостности данных, состояния базы данных до и после выполнения транзакций).

Создание базы данных в системе SQL-сервер осуществляется командой CREATE DATABASE. Следует отметить, что процедура создания базы данных в SQL-сервере требует наличия прав администратора сервера.

```
CREATE DATABASE имя_базы_данных

[ON [PRIMARY]

[ <определение_файла>  [,...n] ]

[,<определение_группы> [,...n] ] ]

[ LOG ON {<определение_файла>[,...n] } ]

[ FOR LOAD | FOR ATTACH ]
```

Рассмотрим основные параметры представленного оператора.

При выборе имени базы данных следует руководствоваться общими правилами именования объектов. Если имя базы данных содержит пробелы или любые другие недопустимые символы, оно заключается в ограничители (двойные кавычки или квадратные скобки). Имя базы данных должно быть уникальным в пределах сервера и не может превышать 128 символов.

При создании и изменении базы данных можно указать имя файла, который будет для нее создан, изменить имя, путь и исходный размер этого файла. Если в процессе использования базы данных планируется ее размещение на нескольких дисках, то можно создать так называемые вторичные файлы базы данных с расширением *.ndf. В этом случае основная информация о базе данных располагается в первичном (PRIMARY) файле, а при нехватке для него свободного места добавляемая информация будет размещаться во вторичном файле. Подход, используемый в SQL-сервере, позволяет распределять содержимое базы данных по нескольким дисковым томам.

Параметр ON определяет список файлов на диске для размещения информации, хранящейся в базе данных.

Параметр PRIMARY определяет первичный файл. Если он опущен, то первичным является первый файл в списке.

Параметр LOG ON определяет список файлов на диске для размещения журнала транзакций. Имя файла для журнала транзакций генерируется на основе имени базы данных, и в конце к нему добавляются символы _log.

При создании базы данных можно определить набор файлов, из которых она будет состоять. Файл определяется с помощью следующей конструкции:

<определение_файла>::=

([NAME=логическое_имя_файла,]

FILENAME='физическое_имя_файла'

[,SIZE=размер_файла]

[,MAXSIZE={max_размер_файла |UNLIMITED }]

[, FILEGROWTH=величина_прироста])[,...n]

Здесь логическое имя файла – это имя файла, под которым он будет опознаваться при выполнении различных SQL-команд.

Физическое имя файла предназначено для указания полного пути и названия соответствующего физического файла, который будет создан на жестком диске. Это имя останется за файлом на уровне операционной системы.

Параметр SIZE определяет первоначальный размер файла; минимальный размер параметра – 512 Кб, если он не указан, по умолчанию принимается 1 Мб.

Параметр MAXSIZE определяет максимальный размер файла базы данных. При значении параметра UNLIMITED максимальный размер базы данных ограничивается свободным местом на диске.

При создании базы данных можно разрешить или запретить автоматический рост ее размера (это определяется параметром FILEGROWTH) и указать приращение с помощью абсолютной величины в Мб или процентным соотношением. Значение может быть указано в килобайтах, мегабайтах, гигабайтах, терабайтах или процентах (%). Если указано число без суффикса МБ, КБ или %, то по умолчанию используется значение МБ. Если размер шага роста указан в процентах (%), размер увеличивается на заданную часть в процентах от размера файла. Указанный размер округляется до ближайших 64 КБ.

Дополнительные файлы могут быть включены в группу:

<определение_группы>::=FILEGROUP имя_группы_файлов

<определение_файла>[,...n]

Пример 3.1. Создать базу данных, причем для данных определить три файла на диске C, для журнала транзакций – два файла на диске C.

```
CREATE DATABASE Archive
ON PRIMARY ( NAME=Arch1,
  FILENAME=’c:\user\data\archdat1.mdf’,
SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch2,
  FILENAME=’c:\user\data\archdat2.mdf’,
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch3,
  FILENAME=’c:\user\data\archdat3.mdf’,
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
LOG ON
(NAME=Archlog1,
  FILENAME=’c:\user\data\archlog1.ldf’,
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Archlog2,
  FILENAME=’c:\user\data\archlog2.ldf’,
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

Пример 3.1. Создание базы данных. (html, txt)

12. Создание таблиц посредством CREATE TABLE. Определение ограничений целостности NOT NULL, DEFOULT, CHECK, PRIMARY KEY, FOREIGN KEY.

Создание таблиц выполняется с помощью команды **CREATE TABLE**

```
CREATE TABLE <TABLE-NAME>
      ( <COLUMN name> <DATA type>[(<SIZE>)],
      <COLUMN name> <DATA type> [(<SIZE>)] ... );
```

Создать таблицу P, с информацией о поставщиках

```
CREATE TABLE P
    (pnum int, pname char(10))
```

При создании таблиц могут быть введены определения на вводимые значения. При этом при попытке ввода несовместимых данных будет выведена ошибка. По своей сути ограничения могут быть на один столбец и на неск. столбцов.
Ограничение на один столбец указывается после типа данных в объявлении столбца. Ограничения на несколько столбцов(всю таблицу), указываются после объявления последнего столбца таблицы.

Типы ограничений:

1. NOT NULL
   Запрещает ввод NULL значения. Такое ограничение может быть задано только на один столбец.

   (pnum int NOT NULL, pnum char(20) NOT NULL)

2. DEFOULT
   Ограничение умолчания. Задается только как ограничение на столбец.

   dprice money DEFOULT 0

3. CHECK
   Ограничение этого типа задает множество значений атрибутов. Может быть задано как на столбец, так и на всю таблицу.

   dprice money CHECK (dprice<50)
   dname char(20) CHECK (dname like [А-Я, а-я]%

4. PRIMARY KEY
   Это специальный случай комбинации NOT NULL и UNIQUE. Простой первичный ключ задается ограничением на один столбец.

   CREATE TABLE P
   (pnum int PRIMARY KEY, pname char(10) not NULL UNIQUE)

5. FOREIGN KEY

   Ограничение внешних ключей определяется в начальной таблице и определяет ее связь с родительской.

```
CREATE TABLE PD
(pnum int,
 dnum int,
 volume int not null)

CONSTRAINT PK_PD
PRIMARY KEY (pnum, dnum)
CONSTRAINT FK_PD_P
 FOREIGN KEY pnum REFERENCES P(pnum)
CONSTRAINT FK_PD_D
FOREIGN key clnum REFERENCES D(dnum)
```

13. Выборка данных с помощью оператора SELECT. Использование агрегатных функций в операторе SELECT.

Этот оператор позволяет производить выборку данных из таблиц и преобразовывать к нужному виду полученные результаты. Результатом выполнения оператора SELECT является таблица, к этой таблице вновь может быть применен оператор SELECT, таким образом запросы могут быть вложенными.

Синтаксис

SELECT <список столбцов>
  FROM<список таблиц>
[WHERE<условия выбора строк>]
[GROUP BY<условия группировки строк>]
[HAVING<условия выбора групп>]
[ODER BY<условия сортировки>]

В предложении SELECT перечисляются столбцы, значения которых будут входить в результирующую таблицу. Столбцы размещаются в том же порядке в котором они указаны в SELECT. Если имя столбца содержит разделители, то оно записывается в квадратных скобках. Если столбцы разных таблиц имеют одинаковые имена то такие имена записываются как составные:

<имя таблицы>.<имя столбца>

В предложении FROM перечисляются имена таблиц которые содержат столбцы указанные в предложении SELECT.

Получить всю информацию о деталях

SELECT dnum, dname, dprice
  FROM D

Список всех столбцов заменяет символ *

После служебного слова WHERE указывается условия выбора строк помещаемых в результирующую таблицу, могут быть указаны различные типы условий:

<,>,<=,>=,=,<>, OR, AND

Получить информацию о поставщиках *иванов* и *петров.*

SELECT *
 FROM P
 WHERE pname = ‘Иванов’ OR pname=’Петров’

Проверка на принадлежность множеству выполняется с помощью операции IN

WHERE pname IN (‘Иванов’’Петров’)

Проверка на принадлежность диапазону BETWEEN. Операция определяет max и min границы диапазона в который должно попасть значение. Обе границы считаются принадлежащими диапазону.

Вывести информацию о деталях от 10 до 20р.

```
SELECT*
  FROM D
  WHERE dprice BETWEEN 10 AND 20
```

Использование агрегатных функций.

SELECT может иметь агрегатные функции которые дают единственное значение для целой группы строк таблицы.
Агрегатная функция записывается в виде:

<имя функции>(<имя столбца>)

Используются следующие функции:

- SUM – возвращает сумму значений столбца
- MIN – возвращает минимальное значение
- MAX – возвращает максимальное значение
- FIRST – возвращает первое значение в столбце
- LAST – возвращает последнее значение в столбце
- COUNT – возвращает количество значений в столбце
- AVG – возвращает среднее значение

Пример
Вывести общее количество поставляемых деталей.

```
SELECT SUM(volume) as sum
  FROM PD
```

14. Оператор объединения UNION. Оператор пересечения INTERSECT. Оператор вычитания EXEPT.

UNION

Оператор указывается между запросами. В упрощенном виде это выглядит следующим образом:

<запрос1>
UNION [**ALL**]
<запрос2>
UNION [**ALL**]
<запрос3>
.....;

По умолчанию любые дублирующие записи автоматически скрываются, если не использовано выражение UNION ALL.

Необходимо отметить, что UNION сам по себе не гарантирует порядок строк. Строки из второго запроса могут оказаться в начале, в конце или вообще перемешаться со строками из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать выражение ORDER BY.

Существуют два основных правила, регламентирующие порядок использования оператора UNION:

- Число и порядок извлекаемых столбцов должны совпадать во всех объединяемых запросах;
- Типы данных в соответствующих столбцах должны быть совместимы.

Использование UNION при выборке из двух таблиц

Даны две таблицы:

sales2005

person	amount
Иван	1000
Алексей	2000
Сергей	5000

sales2006

person	amount
Иван	2000
Алексей	2000
Петр	35000

При выполнении следующего запроса:

SELECT * **FROM** sales2005
UNION
SELECT * **FROM** sales2006;

получается результирующий набор, однако порядок строк может произвольно меняться, поскольку ключевое выражение ORDER BY не было использовано:

person	amount
Иван	1000
Алексей	2000
Сергей	5000
Иван	2000
Петр	35000

EXEPT|INTERSECT

```
<запрос1>
{ EXCEPT | INTERSECT }
<запрос2>
```

Оператор EXCEPT возвращает все различные значения, возвращенные левым запросом и отсутствующие в результатах выполнения правого запроса.

Оператор INTERSECT возвращает все различные значения, входящие в результаты выполнения, как левого, так и правого запроса.

Основные правила объединения результирующих наборов двух запросов с оператором EXCEPT или INTERSECT таковы:

- количество и порядок столбцов должны быть одинаковыми во всех запросах;
- типы данных должны быть совместимыми.

Примеры

Вывести N деталей, которые поставляются 1 и 2 поставщиками.

```
SELECT dnum
FROM PD
WHERE pnum=1
INTERSECT
  SELECT dnum
  FROM PD
  WHERE dnum = 2
```

Вывести N поставщиков, которые сейчас не поставляют детали:

```
SELECT pnum
FROM P
EXCEPT
SELECT pnum
FROM PD
```

15. Подзапросы. Классификация подзапросов. Реализация операций вычитания и пересечения посредством подзапросов.

Представляет собой оператор, вложенный в тело другого оператора. Кодирование подзапроса подчиняется тем же правилам, что и кодирование простых запросов. Внешние операции используют результат выполнения внутреннего оператора для определения окончания результата. По количеству возвращаемых значений подзапросы можно разделить на

- скалярные – возвращает одно значение
- табличные – возвращают множество значений

по способу выполнения запроса:

- простые
- сложные

Простой запрос может рассматривается независимо от внешнего запроса СУБД выполняет такой подзапрос один раз, а затем помещают его результат во внешнем запросе.
Сложный подзапрос не может рассматриваться независимо от внешнего. Выполнение начинается с внешнего запроса, который отбирает каждую отдельную строку. Для каждой такой строки подзапрос выполняется 1 раз.

Пример простого скалярного подзапроса.

Вывести наименование деталей, цена которых больше цены болта.

```
SELECT dname
FROM D
WHERE dprice > (SELECT dprice FROM D WHERE dname ‘болт’)
```

Подзапросы можно использовать не только в предложении WHERE, но и в др. Это зависит от диалекта используемого в СУБД.

Табличные подзапросы.

Такие подзапросы возвращают табл. значений поэтому их результат надо обрабатывать спец. образом. Для этого используют операции IN, ANY(SOME), ALL.

Использование IN
Операция IN осуществляет проверку на принадлежность к значению множества, которое получается после выполнения подзапроса.

Пример.
Определить поставщиков, которые поставляют детали в наст. вр.

```
SELECT pname
  FROME P
  WHERE pnum IN(SELECT pnum from PD)
```

Использование ANY
Условие сравнения считается выполненным когда оно выполняется хотя бы для одного значения, полученного после выполнения подзапроса.

Пример.
Определить наимен. деталей, которые поставляются в наст. вр.

```
SELECT dname
FROM D
WHERE dnum = ANY(SELECT dnum FROM PD)
```

Использование ALL и ANY
Если ALL условие сравнения считается выполненным когда всех значений полученных после выполнения подзапроса получено пустое множество.

Пример.
Вывести наимен. деталей с макс. ценой.

```
SELECT dname
FROM D
WHERE dprice >= ALL(SELECT dprice FROM D).
```

Примеры сложных табличных подзапросов.

Для их реализации используются операции EXISTS и NOT EXISTS.
Для операции EXISTS результат = true, если в возвращенной подзапросом таблице есть хотя бы одна сторона.
Если результирующая таблица пуста, то EXISTS возвращает значение false.
Для NOT EXISTS исп-ся обратн. правила обработки. Так как обе операции проверяют лишь наличие строк в таблице подзапросов, то эта таблица может содержать производное количество строк.

Пример. Определить поставщиков, которые поставляют детали в настоящее время.

```
SELECT pname
FROM p
WHERE EXIST(SELECT* FROM PD
WHERE PD.pnum = P.pnum)
```

Реализация пересечения
В SQL нет отдельного оператора реализации пересечения. Поэтому это делается с помощью подзапросов.

Пример.

Вывести наименование поставщиков из таблицы P для которых не существует деталей в таблице D записи о которых не существует в таблице PD.

```
SELECT pname
FROM P
WHERE NOT EXIST(Select dname FROM D Where NOT EXISTS(Select * FROM PD
                                WHERE PD.pnum = P.pnum AND pd.dnum = d.dnum)
```

16. Операторы модификации данных INSERT, UPDATE, DELETE.

зык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью *запросов действия* пользователь может добавить, удалить или обновить блоки данных. Существует три вида *запросов действия*:

- INSERT INTO – *запрос добавления* ;
- DELETE – *запрос удаления* ;
- UPDATE – *запрос обновления*.

Запрос добавления

Оператор INSERT применяется для *добавления записей* в таблицу. Формат оператора:

<оператор_вставки>::=INSERT INTO <имя_таблицы>
[(имя_столбца [,...n])]
{VALUES (значение[,...n])|
<SELECT_оператор>}
Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Первая форма оператора INSERT с параметром VALUES предназначена для вставки единственной строки в указанную таблицу. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERTуказывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL, за исключением тех случаев, когда при описании столбца использовался параметр DEFAULT. Список значений должен следующим образом соответствовать списку столбцов:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
- типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 8.1. Добавить в таблицу ТОВАР новую запись.

INSERT INTO Товар (Название, Тип, Цена)
VALUES(" Славянский ", " шоколад ", 12)
Если столбцы таблицы ТОВАР указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы ТОВАР, оператор можно упростить.

INSERT INTO Товар VALUES (" Славянский ",
" шоколад ", 12)
Вторая форма оператора INSERT с параметром SELECT позволяет скопировать множество строк из одной таблицы в другую. Предложение SELECT может представлять собой любой допустимый

оператор SELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора SELECT, применимы и в этом случае.

Поскольку оператор SELECT в общем случае возвращает множество записей, то оператор INSERT в такой форме приводит к *добавлению* в таблицу аналогичного числа новых записей.

Пример 8.2. Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог
  (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата)
  AS Месяц, Sum(Товар.Цена*Сделка.Количество)
  AS Стоимость
FROM Товар INNER JOIN Сделка
  ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Запрос удаления

Оператор DELETE предназначен для *удаления группы записей* из таблицы.

Формат оператора:

```
<оператор_удаления> ::=DELETE
   FROM <имя_таблицы>[WHERE <условие_отбора>]
```

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Если предложение WHERE присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложение WHERE, из таблицы будут *удалены все записи*, однако сама таблица сохранится.

Пример 8.3. Удалить все прошлогодние сделки.

```
DELETE
FROM Сделка
WHERE Year(Сделка.Дата)=Year(GETDATE())-1
```

Пример 8.3. Удаление всех прошлогодних сделок. (html, txt)

В приведенном примере условие отбора формируется с учетом года (функция Year) от текущей даты (функция GETDATE()).

Запрос обновления

Оператор UPDATE применяется для *изменения* значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

```
<оператор_изменения> ::=
   UPDATE имя_таблицы SET имя_столбца=
     <выражение>[,...n]
   [WHERE <условие_отбора>]
```

Параметр имя_таблицы – это либо имя таблицы базы данных, либо имя обновляемого представления. В предложении SET указываются имена одного и более столбцов, данные в которых необходимо изменить.

Предложение WHERE является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложение WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

Пример Для товаров первого сорта установить цену в значение 140 и остаток – в значение 20 единиц.

```
UPDATE Товар SET Товар.Цена=140, Товар.Остаток=20
WHERE Товар.Сорт=" Первый "
```

ПримерУвеличить цену товаров первого сорта на 25%.

```
UPDATE Товар SET Товар.Цена=Товар.Цена*1.25
WHERE Товар.Сорт=" Первый "
```

Пример 8.6. В сделке с максимальным количеством товара увеличить число товаров на 10%.

```
UPDATE Сделка SET Сделка.Количество=
   Сделка.Количество*1.1
WHERE Сделка.Количество=
   (SELECT Max(Сделка.Количество) FROM Сделка)
```

17. Изменение структуры таблицы с помощью ALTER TABLE.

Оператор ***ALTER TABLE*** обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Синтаксис:

ALTER [IGNORE] TABLE имя_таблицы
  alter_specification
  [, alter_specification ...]
Можно производить следующие изменения в таблице (все они записываются в alter_specification):

- добавление поля:
  - ADD [COLUMN] определение_столбца
  - [FIRST | AFTER имя_столбца]

  или

  ADD [COLUMN] (определение_столбца,
  определение_столбца,...)

Здесь, как и далее, определение_столбца записывается так же, как при создании таблицы.

- добавление индексов:

  ADD *INDEX* [имя_индекса] (имя_индексируемого_столбца,...) или ADD *PRIMARY* KEY (имя_индексируемого_столбца,...) или ADD *UNIQUE*[имя_индекса] (имя_индексируемого_столбца,...) или ADD FULLTEXT [имя_индекса] (имя_индексируемого_столбца,...)

- изменение поля:

  ALTER [COLUMN] имя_столбца {SET DEFAULT literal | DROP DEFAULT} или CHANGE [COLUMN] старое_имя_столбца определение_столбца илиMODIFY [COLUMN] определение_столбца

- удаление поля, индекса, ключа:
  - DROP [COLUMN] имя_столбца
  - DROP PRIMARY KEY
  - DROP INDEX имя_индекса
- переименование таблицы:

  RENAME [TO] новое_имя_таблицы

- переупорядочение полей таблицы:

  ORDER BY поле

  или

  опции_таблицы

Если оператор *ALTER TABLE* используется для изменения определения типа столбца, но DESCRIBE имя_таблицы показывает, что столбец не изменился, то, возможно, *MySQL* игнорирует данную модификацию по одной из причин, описанных в специальном разделе документации. Например,

при попытке изменить столбец VARCHAR на CHAR *MySQL* будет продолжать использовать VARCHAR, если данная таблица содержит другие столбцы с переменной длиной.

Оператор *ALTER TABLE* во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Это делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения *ALTER TABLE* исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица. Следует отметить, что при использовании любой другой опции для ALTER TABLE, кроме RENAME, *MySQL* всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца).

Пример. Добавим в созданную таблицу Persons поле для записи года рождения человека:

```
mysql> ALTER TABLE Persons
   ADD bday INTEGER AFTER last_name;
```

18. Создание и использование представлений в СУБД MS SQL Server.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения *представления* используется только оперативная память. В отличие от других объектов базы данных*представление* не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого *представления*.

Создания и изменения *представлений* в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

<определение_представления> ::=
{ CREATE| ALTER} VIEW имя_представления
[(имя_столбца [,...n])]
[WITH ENCRYPTION]
AS SELECT_оператор
[WITH CHECK OPTION]
Рассмотрим назначение основных параметров.

По умолчанию имена столбцов в *представлении* соответствуют именам столбцов в исходных таблицах. Явное указание имени столбца требуется для вычисляемых столбцов или при объединении нескольких таблиц, имеющих столбцы с одинаковыми именами. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в *представлении*.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. Если при определении *представления* необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных, необходимо применить этот аргумент.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через *представление*, на соответствие критериям, определенным в операторе SELECT. Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из*представления*. Такое случается, если для *представления* установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам. Использование аргумента WITH CHECK OPTION гарантирует, что сделанные изменения будут отображены в *представлении*. Если пользователь пытается выполнить изменения, приводящие к исключению строки из *представления*, при заданном аргументе WITH CHECK OPTIONсервер выдаст сообщение об ошибке и все изменения будут отклонены.

Пример 10.1. Показать в представлении клиентов из Москвы.

Создание представления:

```
CREATE  VIEW view1 AS
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
```

Пример 10.1. Представление клиентов из Москвы. (html, txt)
Выборка данных из представления:

```
SELECT * FROM view1
```

Обращение к *представлению* осуществляется с помощью оператора SELECT как к обычной таблице.

Представление можно использовать в команде так же, как и любую другую таблицу.
К *представлению* можно строить запрос, модифицировать его (если оно отвечает определенным требованиям), соединять с другими таблицами. Содержание *представления* не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. *Представления* значительно расширяют возможности управления данными. В частности, это прекрасный способ разрешить доступ к информации в таблице, скрыв часть данных.

Так, в примере 10.1 *представление* просто ограничивает доступ пользователя к данным таблицы Клиент, позволяя видеть только часть значений.

Выполним команду:

INSERT INTO view1 VALUES (12,'Петров', 'Самара')
Это допустимая команда в *представлении*, и строка будет добавлена с помощью *представления* view1 в таблицу Клиент. Однако, когда информация будет добавлена, строка исчезнет из *представления*, поскольку название города отлично от Москвы. Иногда такой подход может стать проблемой, т.к. данные уже находятся в таблице, но пользователь их не видит и не в состоянии выполнить их удаление или модификацию. Для исключения подобных моментов служит WITH CHECK OPTION в определении *представления*. Фраза размещается в определении *представления*, и все команды модификации будут подвергаться проверке.

```
ALTER VIEW view1
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
WITH CHECK OPTION
```
Для такого *представления* вышеупомянутая вставка значений будет отклонена системой.

Таким образом, *представление* может изменяться командами модификации DML, но фактически модификация воздействует не на само *представление*, а на базовую таблицу.

Представление удаляется командой:

DROP VIEW имя_представления [,...n]

19. Создание и выполнение хранимых процедур в СУБД MS SQL Server

ХП представляю собой последовательность операторов на языке SQL которые хранятся в БД и выполняются как единое целое.

Существует два типа ХП:

- Системные ХП. Предназначены для получения информации из системных таблиц и выполнения служебных операций.
- Пользовательские ХП., которые создаются разработчиками БД.

При первом вызове ХП СУБД генерирует и оптимизирует план выполнения ХП. ХП сохраняется в БД в откомпилированном виде поэтому скорости из выполнения значительно выше обычных SQL запросов. ХП выполняются на сервере.

Создание ХП:

```
CREATE PROC<имя ХП>[<список параметров>]
[WITH{ENCRYPTION|RECOMPILE}]
AS
<SQL операторы>
GO
```

Описание параметров в списке выполняется следующим образом:
@<имя><тип данных>[=<значение по умолчанию][OUTPUT]

Пример. Разработать ХП для вывода наименования поставщика по заданному номеру.

```
CREATE PROC P1 (@pnum int)
AS
  DECLARE @pname char(10)
  SELECT @pname = pname
  FROM P
  WHERE pnum = @ pnum
  PRINT @pname
GO
EXEC P1
```

Выполнять ХП позволяет оператор EXEC <имя ХП><список параметров>
Для возвращаемых параметров указывается слово OUTPUT. Чтобы использовать возвращаемый параметр надо объявить переменную для его хранения в некотором скрипте который возвращает данную ХП.

Пример.

```
DECLARE @dprice money
EXEC P2 3, $, @price OUTPUT
PRINT ‘цена в валюте=’ + CONVERT(char(10), @dprice)
```

Если в операторе EXEC перечисляются не все параметры которые были определены в операторе CREATE PROC то надо указать имена формальных параметров.

В теле ХП можно использовать оператор RETURN

RETURN[<целочисленное выражение>] для завершения выполнения ХП

Получить возвращаемое оператором RETURN значение переменной можно:

EXEC @<имя переменной>=<имя ХП><список параметров>

Текст ХП можно просматривать выполнив процедуру
EXEC sp_helptext<имя ХП>, но если указанный параметр WITH ENCRYPTION.

В некоторых случаях требуется чтобы ХП перекомпилировалась при каждом выполнении
WITH RECOMPILE
Удаление ХП: DROP PROC <имя ХП>

20. Создание и выполнение функций в СУБД MS SQL Server

Пользовательские функции сходны с *хранимыми процедурами*, но, в отличие от них, могут применяться в запросах так же, как и системные*встроенные функции*. *Пользовательские функции*, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а *пользовательские функции* способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Функции Scalar

Создание и изменение *функции* данного типа выполняется с помощью команды:

```
<определение_скаляр_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.

Функция может содержать один или несколько *входных параметров* либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой *функции* имя и начинаться с символа " @ ". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове *функции*.

С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое *функцией* значение.

Дополнительные параметры, с которыми должна быть создана *функция*, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания *функции*, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы *функции*. Кроме того, в теле *функции* может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объектов может привести к нарушению работы *функции*. Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой *функции* ключевое слово SCHEMABINDING.

Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом *функции*.

Когда в ходе выполнения кода *функции* встречается ключевое слово RETURN, выполнение *функции* завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле *функции* разрешается использование множества командRETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные

выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого словаRETURNS.

Пример Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

```
CREATE FUNCTION
   user1.sales(@data DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @c INT
SET @c=(SELECT SUM(количество)
      FROM Сделка
      WHERE дата=@data)
RETURN (@c)
END
```

Функции Inline

Создание и изменение *функции* этого типа выполняется с помощью команды:

```
<определение_табл_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS TABLE
[ WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
RETURN [(] SELECT_оператор [)]
```

Основная часть параметров, используемых при создании *табличных функций*, аналогична параметрам *скалярной функции*. Тем не менее создание*табличных функций* имеет свою специфику.

После ключевого слова RETURNS всегда должно указываться ключевое слово TABLE. Таким образом, *функция* данного типа должна строго возвращать значение *типа данных TABLE*. Структура возвращаемого значения *типа TABLE* не указывается явно при описании собственно типа данных. Вместо этого сервер будет автоматически использовать для возвращаемого значения TABLE структуру, возвращаемую запросом SELECT, который является единственной командой *функции*.

Особенность *функции* данного типа заключается в том, что структура значения TABLE создается автоматически в ходе выполнения запроса, а не указывается явно при определении типа после ключевого слова RETURNS.

Возвращаемое *функцией* значение *типа TABLE* может быть использовано непосредственно в запросе, т.е. в разделе FROM.

Пример Создать и применить функцию табличного типа для определения двух наименований товара с наибольшим остатком.

```
CREATE FUNCTION user1.itog()
RETURNS TABLE
AS
RETURN (SELECT TOP 2 Товар.Название
  FROM Товар INNER JOIN Склад
  ON Товар.КодТовара=Склад.КодТовара
  ORDER BY Склад.Остаток DESC)
```

Использовать *функцию* для получения двух наименований товара с наибольшим остатком можно следующим образом:

```
SELECT Название
FROM user1.itog()
```

Функции Multi-statement

Создание и изменение *функций* типа *Multi-statement* выполняется с помощью следующей команды:

```
<определение_мульти_функции>::=
{CREATE | ALTER }FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS @имя_параметра TABLE
   <определение_таблицы>
[WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN
END
```

Использование большей части параметров рассматривалось при описании предыдущих *функций*.

Отметим, что *функции* данного типа, как и *табличные*, возвращают значение *типа TABLE*. Однако, в отличие от *табличных функций*, при создании*функций Multi-statement* необходимо явно задать структуру возвращаемого значения. Она указывается непосредственно после ключевого слова TABLEи, таким образом, является частью определения возвращаемого типа данных. Синтаксис конструкции <определение_таблицы> полностью соответствует одноименным структурам, используемым при создании обычных таблиц с помощью команды CREATE TABLE.

Набор возвращаемых данных должен формироваться с помощью команд INSERT, выполняемых в теле *функции*. Кроме того, в теле *функции* допускается использование различных конструкций языка SQL, которые могут контролировать значения, размещаемые в выходном наборе строк. При работе с командой INSERT требуется явно указать имя того объекта, куда необходимо вставить строки. Поэтому в *функциях* типа *Multi-statement*, в отличие от*табличных*, необходимо присвоить какое-то имя объекту с *типом данных TABLE* – оно и указывается как возвращаемое значение.

Завершение работы *функции* происходит в двух случаях: если возникают ошибки выполнения и если появляется ключевое слово RETURN. В отличие от*функций скалярного типа*, при использовании команды RETURN не нужно указывать возвращаемое значение. Сервер автоматически возвратит набор данных *типа TABLE*, имя и структура которого была указана после ключевого слова RETURNS. В теле *функции* может быть указано более одной командыRETURN.

Необходимо отметить, что работа *функции* завершается только при наличии команды RETURN. Это утверждение верно и в том случае, когда речь идет о достижении конца тела *функции* – самой последней командой должна быть команда RETURN.

21. Создание и использование триггеров MS SQL Server. Триггеры типа INSTEAD OF и типа AFTER

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми *триггеры* связаны. Каждый *триггер* привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные *триггером*.

Создает *триггер* только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому *триггеры* необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, *триггер* выполняется неявно в каждом случае возникновения *триггерного события*, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском *триггера*. С помощью *триггеров* достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:

```
<Определение_триггера>::=
 CREATE TRIGGER имя_триггера
 BEFORE | AFTER <триггерное_событие>
 ON <имя_таблицы>
 [REFERENCING
  <список_старых_или_новых_псевдонимов>]
 [FOR EACH { ROW | STATEMENT}]
 [WHEN(условие_триггера)]
 <тело_триггера>
```

Реализация триггеров в среде MS SQL Server

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения *триггера*:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
[WITH ENCRYPTION ]
```

```
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
  sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
 [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
 бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри *триггера* к другим базам данных, в том числе и расположенным на удаленном сервере.

Рассмотрим назначение аргументов из команды CREATE | ALTER TRIGGER.

Имя *триггера* должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

При указании аргумента WITH ENCRYPTION сервер выполняет шифрование кода *триггера*, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров

В SQL Server существует два параметра, определяющих поведение *триггеров*:

- AFTER. *Триггер* выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, *триггер* не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение*триггера* осуществляется в теле одной транзакции: если произойдет откат *триггера*, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER -триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение несколькихAFTER -триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все *триггеры* являются AFTER -триггерами.
- INSTEAD OF. *Триггер* вызывается вместо выполнения команд. В отличие от AFTER -триггера INSTEAD OF -триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF -триггер.

Триггеры различают по типу команд, на которые они реагируют.

Существует три типа *триггеров*:

- INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.
- UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.
- DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать *триггер*. При его создании должна быть указана хотя бы одна команда. Допускается *создание триггера*, реагирующего на две или на все три команды.

Аргумент WITH APPEND позволяет создавать несколько *триггеров* каждого типа.

При *создании триггера* с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске *триггера*.

Отметим, что внутри *триггера* не допускается выполнение ряда операций, таких, например, как:

- создание, изменение и удаление базы данных;
- восстановление резервной копии базы данных или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется *триггер*. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых *триггеров*. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

Программирование триггера

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: *inserted* и *deleted* . В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется *триггер*. Для каждого *триггера* создается свой комплект таблиц inserted и deleted, поэтому никакой другой *триггер* не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение *триггера*, содержимое таблиц inserted и deleted может быть разным:

- команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения *триггера* все строки из таблицы inserted переместятся в исходную таблицу;
- команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; *триггер* может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;
- команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении*триггера*. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения *триггера*.

Для получения информации о количестве строк, которое будет изменено при успешном завершении *триггера*, можно использовать функцию@@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что *триггер* запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому *триггер*должен обрабатывать все эти строки.

Если *триггер* обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определенная транзакция, поэтому внутри *триггера* допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения *триггера* и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Для получения списка столбцов, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение *триггера*, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

Для *удаления триггера* используется команда

DROP TRIGGER {имя_триггера} [,...n]

22. Создание и использование курсоров в СУБД MS SQL Server.

Курсор в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к базе данных, в памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой *курсора*. Указанная область в памяти поименована и доступна для прикладных программ.

В соответствии со стандартом SQL при работе с *курсорами* можно выделить следующие основные *действия*:

- создание или *объявление курсора* ;
- ***открытие курсора,*** т.е. наполнение его данными, которые сохраняются в многоуровневой памяти ;
- *выборка из курсора* и *изменение* с его помощью строк данных;
- *закрытие курсора*, после чего он становится недоступным для пользовательских программ;
- *освобождение курсора*, т.е. удаление *курсора* как объекта, поскольку его *закрытие* необязательно освобождает ассоциированную с ним память.

SQL Server поддерживает три *вида курсоров*:

- *курсоры* SQL применяются в основном внутри триггеров, хранимых процедур и сценариев;
- *курсоры* сервера действуют на сервере и реализуют программный интерфейс приложений для ODBC, OLE DB, DB_Library;
- *курсоры* клиента реализуются на самом клиенте. Они выбирают весь результирующий набор строк из сервера и сохраняют его локально, что позволяет ускорить операции обработки данных за счет снижения потерь времени на выполнение сетевых операций.

Управление курсором в среде MS SQL Server

Управление курсором реализуется путем выполнения следующих команд:

- DECLARE – создание или *объявление курсора* ;
- OPEN – *открытие курсора*, т.е. наполнение его данными;
- FETCH – *выборка из курсора* и *изменение* строк данных с помощью курсора;
- CLOSE – *закрытие курсора* ;
- DEALLOCATE – *освобождение курсора*, т.е. удаление курсора как объекта.

Объявление курсора

В стандарте SQL для создания *курсора* предусмотрена следующая команда:

```
<создание_курсора>::=
  DECLARE имя_курсора
    [INSENSITIVE][SCROLL] CURSOR
  FOR SELECT_оператор
  [FOR { READ_ONLY | UPDATE
    [OF имя_столбца[,...n]]}]
```

При использовании ключевого слова INSENSITIVE будет создан *статический курсор*. *Изменения данных* не разрешаются, кроме того, не отображаются*изменения*, сделанные другими пользователями. Если ключевое слово INSENSITIVE отсутствует, создается *динамический курсор*.

При указании ключевого слова SCROLL созданный *курсор* можно прокручивать в любом направлении, что позволяет применять любые команды *выборки*. Если этот аргумент опускается, то *курсор* окажется *последовательным*, т.е. его просмотр будет возможен только в одном направлении – от начала к концу.

SELECT-оператор задает тело запроса SELECT, с помощью которого определяется результирующий набор строк *курсора*.

При указании аргумента FOR READ_ONLY создается *курсор* "только для чтения", и никакие модификации данных не разрешаются. Он отличается от*статического*, хотя последний также не позволяет менять данные. В качестве курсора "только для чтения" может быть объявлен *динамический курсор*, что позволит отображать *изменения*, сделанные другим пользователем.

Создание *курсора* с аргументом FOR UPDATE позволяет выполнять в *курсоре изменение данных* либо в указанных столбцах, либо, при отсутствии аргумента OF имя_столбца, во всех столбцах.

В среде MS SQL Server принят следующий синтаксис команды создания *курсора*:

<создание_курсора>::=
 DECLARE имя_курсора CURSOR [LOCAL | GLOBAL]
 [FORWARD_ONLY | SCROLL]
 [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
 [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
 [TYPE_WARNING]
 FOR SELECT_оператор
 [FOR UPDATE [OF имя_столбца[,...n]]]

При использовании ключевого слова LOCAL будет создан локальный *курсор*, который виден только в пределах создавшего его пакета, триггера, хранимой процедуры или пользовательской функции. По завершении работы пакета, триггера, процедуры или функции *курсор* неявно уничтожается. Чтобы передать содержимое *курсора* за пределы создавшей его конструкции, необходимо присвоить его параметру аргумент OUTPUT.

Если указано ключевое слово GLOBAL, создается глобальный *курсор* ; он существует до закрытия текущего соединения.

При указании FORWARD_ONLY создается *последовательный курсор* ; *выборку* данных можно осуществлять только в направлении от первой строки к последней.

При указании SCROLL создается *прокручиваемый курсор* ; обращаться к данным можно в любом порядке и в любом направлении.

При указании STATIC создается *статический курсор*.

При указании KEYSET создается ключевой курсор.

При указании DYNAMIC создается *динамический курсор*.

Если для *курсора* READ_ONLY указать аргумент FAST_FORWARD, то созданный *курсор* будет оптимизирован для быстрого доступа к данным. Этот аргумент не может быть использован совместно с аргументами FORWARD_ONLY и OPTIMISTIC.

В *курсоре*, созданном с указанием аргумента OPTIMISTIC, запрещается *изменение* и *удаление строк*, которые были изменены после *открытия курсора*.

При указании аргумента TYPE_WARNING сервер будет информировать пользователя о неявном изменении типа *курсора*, если он несовместим с запросомSELECT.

Открытие курсора

Для *открытия курсора* и наполнения его данными из указанного при создании *курсора* запроса SELECT используется следующая команда:

```
OPEN {{[GLOBAL]имя_курсора }
    |@имя_переменной_курсора}
```

После *открытия курсора* происходит выполнение связанного с ним оператора SELECT, выходные данные которого сохраняются в многоуровневой памяти.

Выборка данных из курсора

Сразу после *открытия курсора* можно выбрать его содержимое (результат выполнения соответствующего запроса) посредством следующей команды:

```
FETCH [[NEXT | PRIOR | FIRST | LAST
  | ABSOLUTE {номер_строки
  | @переменная_номера_строки}
  | RELATIVE {номер_строки |
    @переменная_номера_строки}]
 FROM ]{{[GLOBAL ]имя_курсора }|
   @имя_переменной_курсора }
 [INTO @имя_переменной [,...n]]
```

При указании FIRST будет возвращена самая первая строка полного результирующего набора *курсора*, которая становится текущей строкой.

При указании LAST возвращается самая последняя строка *курсора*. Она же становится текущей строкой.

При указании NEXT возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Теперь она становится текущей. По умолчанию команда FETCH использует именно этот способ *выборки* строк.

Ключевое слово PRIOR возвращает строку, находящуюся перед текущей. Она и становится текущей.

Аргумент ABSOLUTE {номер_строки | @переменная_номера_строки} возвращает строку по ее абсолютному порядковому номеру в полном результирующем наборе *курсора*. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

Аргумент RELATIVE {кол_строки | @переменная_кол_строки} возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет возвращена строка, находящаяся за указанное количество строк перед текущей. При указании нулевого значения возвратится текущая строка. Возвращенная строка становится текущей.

Чтобы *открыть глобальный курсор*, перед его именем требуется указать ключевое слово GLOBAL. Имя *курсора* также может быть указано с помощью переменной.

В конструкции INTO @имя_переменной [,...n] задается список переменных, в которых будут сохранены соответствующие значения столбцов возвращаемой строки. Порядок указания переменных должен соответствовать порядку столбцов в *курсоре*, а тип данных переменной – типу данных в столбце *курсора*. Если конструкция INTO не указана, то поведение команды FETCH будет напоминать поведение команды SELECT – данные выводятся на экран.

Изменение и удаление данных

Для выполнения изменений с помощью *курсора* необходимо выполнить команду UPDATE в следующем формате:

```
UPDATE имя_таблицы SET {имя_столбца={
  DEFAULT | NULL | выражение}}[,...n]
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

За одну операцию могут быть изменены несколько столбцов текущей строки *курсора*, но все они должны принадлежать одной таблице.

Для удаления данных посредством *курсора* используется команда DELETE в следующем формате:

```
DELETE имя_таблицы
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

В результате будет удалена строка, установленная текущей в *курсоре*.

Закрытие курсора

```
CLOSE {имя_курсора | @имя_переменной_курсора}
```

После *закрытия курсор* становится недоступным для пользователей программы. При *закрытии* снимаются все блокировки, установленные в процессе его работы. *Закрытие* может применяться только к открытым *курсорам*. Закрытый, но не *освобожденный курсор* может быть повторно открыт. Не допускается закрывать неоткрытый *курсор*.

Освобождение курсора

Закрытие курсора необязательно освобождает ассоциированную с ним память. В некоторых реализациях нужно явным образом освободить ее с помощью оператора DEALLOCATE. После *освобождения курсора* освобождается и память, при этом становится возможным повторное использование имени *курсора*.

```
DEALLOCATE { имя_курсора |
  @имя_переменной_курсора }
```

Для контроля достижения конца *курсора* рекомендуется применять функцию: @@FETCH_STATUS

Функция @@FETCH_STATUS возвращает:

0, если *выборка* завершилась успешно;

-1, если *выборка* завершилась неудачно вследствие попытки *выборки* строки, находящейся за пределами *курсора* ;

-2, если *выборка* завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

23. Проектирование баз данных с помощью метода нормальных форм. Определения нормальных форм. Понятия функциональной и транзитивной зависимостей.

При проектировании базы данных решаются две основные проблемы.

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было, по возможности, лучшим (эффективным, удобным и т. д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных, т. е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создания каких дополнительных структур (например, индексов) потребовать и т. д.? Эту проблему обычно называют проблемой физического проектирования баз данных.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- *первая нормальная форма (1NF)*;
- *вторая нормальная форма (2NF)*;
- *третья нормальная форма (3NF)*;
- *нормальная форма Бойса-Кодда (BCNF)*;
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм состоят в следующем:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей нормальной формы;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе процесса проектирования лежит метод нормализации, т. е. декомпозиции отношения, находящегося в предыдущей нормальной форме, на два или более отношений, которые удовлетворяют требованиям следующей нормальной формы.

В этой лекции мы обсудим первые шаги *процесса нормализации*, в которых учитываются функциональные зависимости между атрибутами отношений. Хотя мы и называем эти шаги первыми, именно они имеют основную практическую важность, поскольку позволяют получить схему реляционной базы данных, в большинстве случаев удовлетворяющую потребности приложений.

Минимальные функциональные зависимости и вторая нормальная форма

Пусть имеется переменная отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП, ПРО_НОМ, СЛУ_ЗАДАН}. Новые атрибуты СЛУ_УРОВ иСЛУ_ЗАДАН содержат, соответственно, данные о разряде служащего и о задании, которое выполняет служащий в данном проекте. Будем считать, что разряд служащего определяет размер его заработной платы и что каждый служащий может участвовать в нескольких проектах, но в каждом проекте он выполняет только одно задание. Тогда очевидно, что единственно возможным ключом отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ является составной атрибут {СЛУ_НОМ,

ПРО_НОМ}. Диаграмма минимального множества FD показана на рис. 7.1, а возможное тело значения отношения – на рис. 7.2.

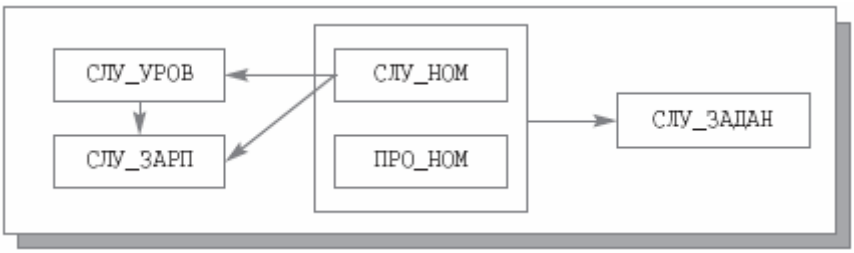

Рис. 7.1. Диаграмма множества FD отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП	ПРО_НОМ	СЛУ_ЗАДАН
2934	2	22400.00	1	A
2935	3	29600.00	1	B
2936	1	20000.00	1	C
2937	1	20000.00	1	D
2934	2	22400.00	2	D
2935	3	29600.00	2	C
2936	1	20000.00	2	B
2937	1	20000.00	2	A

Рис. 7.2. Возможное значение переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ

Аномалии обновления, возникающие из-за наличия неминимальных функциональных зависимостей

Во множество FD отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ входит много FD, в которых детерминантом является не возможный ключ отношения (соответствующие стрелки в диаграмме начинаются не с {СЛУ_НОМ, ПРО_НОМ}, т. е. некоторые функциональные зависимости атрибутов от возможного ключа не являются минимальными). Это приводит к так называемым *аномалиям обновления*. Под ***аномалиями обновления*** понимаются трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE). Обсудим сначала *аномалии обновления*, вызываемые наличием FD СЛУ_НОМ⟶СЛУ_УРОВ (эти *аномалии* связаны с избыточностью хранения значений атрибутов СЛУ_УРОВ и СЛУ_ЗАРП в каждом кортеже, описывающем задание служащего в некотором проекте).

- **Добавление кортежей.** Мы не можем дополнить отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данными о служащем, который в данное время еще не участвует ни в одном проекте (ПРО_НОМ является частью первичного ключа и не может содержать неопределенных значений). Между тем часто бывает, что сначала служащего принимают на работу, устанавливают его разряд и размер зарплаты, а лишь потом назначают для него проект.
- **Удаление кортежей.** Мы не можем сохранить в отношении СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данные о служащем, завершившем участие в своем последнем проекте (по той причине, что значение атрибута ПРО_НОМ для этого служащего становится неопределенным). Между тем

характерна ситуация, когда между проектами возникают перерывы, не приводящие к увольнению служащих.

- **Модификация кортежей.** Чтобы изменить разряд служащего, мы будем вынуждены модифицировать все кортежи с соответствующим значением атрибута СЛУ_НОМ. В противном случае будет нарушена естественная FD СЛУ_НОМ ⟶СЛУ_УРОВ (у одного служащего имеется только один разряд).

Возможная декомпозиция

Для преодоления этих трудностей можно произвести декомпозицию переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ на две переменных отношений –СЛУЖ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП} и СЛУЖ_ПРО_ЗАДАН {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. На основании теоремы Хита эта декомпозиция является декомпозицией без потерь, поскольку в исходном отношении имелась FD {СЛУ_НОМ, ПРО_НОМ}⟶СЛУ_ЗАДАН. На рис. 7.3 показаны диаграммы множеств FD этих отношений, а на рис. 7.4 – их значения.

Рис. 7.3. Диаграммы FD в переменных отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН

Теперь мы можем легко справиться с операциями обновления.

- **Добавление кортежей.** Чтобы сохранить данные о принятом на работу служащем, который еще не участвует ни в каком проекте, достаточно добавить соответствующий кортеж в отношение СЛУЖ.
- **Удаление кортежей.** Если кто-то из служащих прекращает работу над проектом, достаточно удалить соответствующий кортеж из отношенияСЛУЖ_ПРО_ЗАДАН. При увольнении служащего нужно удалить кортежи с соответствующим значением атрибута СЛУ_НОМ из отношений СЛУЖ иСЛУЖ_ПРО_ЗАДАН.
- **Модификация кортежей.** Если у служащего меняется разряд (и, следовательно, размер зарплаты), достаточно модифицировать один кортеж в отношении СЛУЖ.

Значение переменной отношения СЛУЖ

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП
2934	2	22400.00
2935	3	29600.00
2936	1	20000.00
2937	1	20000.00

Значение переменной отношения СЛУЖ_ПРО_ЗАДАН

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2935	1	B
2936	1	C
2937	1	D
2934	2	D
2935	2	C
2936	2	B
2937	2	A

Рис. 7.4. Значения переменных отношений

Вторая нормальная форма

Как видно, на рис. 7.3 отсутствуют FD, не являющиеся **минимальными**. Наличие таких FD на рис. 7.1 вызывало *аномалии обновления*. Проблема заключалась в том, что атрибут СЛУЖ_УРОВ относился к сущности служащий, в то время как первичный ключ идентифицировал сущностьзадание_служащего_в_проекте.

Переменная отношения находится во ***второй нормальной форме (2NF)*** тогда и только тогда, когда она находится в *первой нормальной форме*, и каждый неключевой атрибут[1)] минимально функционально зависит от первичного ключа[2)].

Переменные отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН находятся в *2NF* (все неключевые атрибуты отношений минимально зависят от первичных ключейСЛУ_НОМ и {СЛУ_НОМ, ПРО_НОМ} соответственно). Переменная отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ не находится в *2NF* (например, FD {СЛУ_НОМ, ПРО_НОМ}⟶СЛУ_УРОВ не является минимальной). Любая переменная отношения, находящаяся в *1NF*, но не находящаяся в *2NF*, может быть приведена к набору переменных отношений, находящихся в *2NF*. В результате декомпозиции мы получаем набор проекций исходной переменной отношения, естественное соединение значений которых воспроизводит значение исходной переменной отношения (т. е. это декомпозиция без потерь). Для переменных отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН исходное отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ воспроизводится их естественным соединением по общему атрибуту СЛУ_НОМ.

Заметим, что допустимое значение переменной отношения СЛУЖ может содержать кортежи, информационное наполнение которых выходит за пределы допустимых значений переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ. Например, в теле отношения СЛУЖ может находиться кортеж с данными о служащем с номером 2938, который еще не участвует ни в одном проекте. Наличие такого кортежа не влияет на результат естественного соединения, тело которого все равно будет совпадать с телом допустимого значения переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ.

Нетранзитивные функциональные зависимости и третья нормальная форма

В произведенной декомпозиции переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ множество FD переменной отношения СЛУЖ_ПРО_ЗАДАН предельно просто – в единственной нетривиальной функциональной зависимости детерминантом является возможный ключ. При использовании этой переменной отношения какие-либо *аномалии обновления* не возникают. Однако переменная отношения СЛУЖ не является такой же совершенной.

Аномалии обновлений, возникающие из-за наличия транзитивных функциональных зависимостей

Функциональные зависимости переменной отношения СЛУЖ по-прежнему порождают некоторые *аномалии обновления*. Они вызываются наличием транзитивной FD СЛУ_НОМ ⟶СЛУ_ЗАРП (через FD СЛУ_НОМ⟶СЛУ_УРОВ и СЛУ_УРОВ⟶СЛУ_ЗАРП).
Эти *аномалии* связаны с избыточностью хранения значения атрибута СЛУ_ЗАРП в каждом кортеже, характеризующем служащих с одним и тем же разрядом.

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

Возможная декомпозиция

Для преодоления этих трудностей произведем декомпозицию переменной отношения СЛУЖ на две переменных отношений – СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ}и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}. По теореме Хита, это снова декомпозиция без потерь по причине наличия, например, FD СЛУ_НОМ⟶СЛУ_УРОВ. На рис. 7.5 показаны диаграммы FD этих переменных отношений, а на рис. 7.6 – их возможные значения.

Рис. 7.5. Диаграммы FD в отношениях СЛУЖ1 и УРОВ

Как видно из рис. 7.6, это преобразование обратимо, т. е. любое допустимое значение исходной переменной отношения СЛУЖ является естественным соединением значений отношений СЛУЖ1 и УРОВ. Также можно заметить, что мы избавились от трудностей при выполнении операций обновления.

- **Добавление кортежей.** Чтобы сохранить данные о новом разряде, достаточно добавить соответствующий кортеж к отношению УРОВ.
- **Удаление кортежей.** При увольнении последнего служащего, обладающего данным разрядом, удаляется соответствующий кортеж из отношенияСЛУЖ1, и данные о разряде сохраняются в отношении УРОВ.

- **Модификация кортежей.** При изменении размера зарплаты, соответствующей некоторому разряду, изменяется значение атрибута СЛУ_ЗАРП ровно в одном кортеже отношения УРОВ.

Третья нормальная форма

Значение переменной отношения СЛУЖ1

СЛУ_НОМ	СЛУ_УРОВ
2934	2
2935	3
2936	1
2937	1

Значение переменной отношения УРОВ

СЛУ_УРОВ	СЛУ_ЗАРП
2	22400.00
3	29600.00
1	20000.00

Рис. 7.6. Тела отношений СЛУЖ1 и УРОВ

Трудности, которые мы испытывали, были связаны с наличием транзитивной FD СЛУ_НОМ⟶СЛУ_ЗАРП. Наличие этой FD на самом деле означало, что атрибут СЛУ_ЗАРП характеризовал не сущность служащий, а сущность разряд.

Переменная отношения находится в ***третьей нормальной форме (3NF)*** в том и только в том случае, когда она находится во *второй нормальной форме*, и каждый неключевой атрибут нетранзитивно[1)] функционально зависит от первичного ключа[2)].

Отношения СЛУЖ1 и УРОВ оба находятся в *3NF* (все неключевые атрибуты нетранзитивно зависят от первичных ключей СЛУ_НОМ и СЛУ_УРОВ). ОтношениеСЛУЖ не находится в *3NF* (FD СЛУ_НОМ⟶СЛУ_ЗАРП является транзитивной). Любое отношение, находящееся в *2NF*, но не находящееся в *3NF*, может быть приведено к набору отношений, находящихся в *3NF*. Мы получаем набор проекций исходного отношения, естественное соединение которых воспроизводит исходное отношение (т. е. это декомпозиция без потерь). Для отношений СЛУЖ1 и УРОВ исходное отношение СЛУЖ воспроизводится их естественным соединением по общему атрибуту СЛУ_УРОВ.

Заметим, что допустимые значения отношения УРОВ могут содержать кортежи, информационное наполнение которых выходит за пределы тела отношения СЛУЖ. Например, в теле отношения УРОВ может находиться кортеж с данными о разряде 4, который еще не присвоен ни одному служащему. Наличие такого кортежа не влияет на результат естественного соединения, который все равно будет являться допустимым значением отношения СЛУЖ.

Независимые проекции отношений. Теорема Риссанена

Обратите внимание, что для переменной отношения СЛУЖ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП}, кроме декомпозиции на отношения СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}, возможна и декомпозиция на отношения СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и СЛУЖ_ЗАРП {СЛУ_НОМ, СЛУ_ЗАРП}[3)]. Оба отношения, полученные путем второй декомпозиции, находятся в *3NF*, и эта декомпозиция также является декомпозицией без потерь. Тем не менее вторая декомпозиция, в отличие от первой, не

устраняет проблемы, связанные с обновлением отношения СЛУЖ. Например, по-прежнему невозможно сохранить данные о разряде, которым не обладает ни один служащий. Посмотрим, с чем это связано.

Отношения СЛУЖ1 и УРОВ могут обновляться независимо (являются *независимыми проекциями*), и при этом результат их естественного соединения всегда будет таким, как если бы обновлялось исходное отношение СЛУЖ. Это происходит потому, что FD отношения СЛУЖ трансформировались в индивидуальные ограничения первичного ключа отношений СЛУЖ1 и УРОВ. При второй декомпозиции FD СЛУ_УРОВ⟶ СЛУ_ЗАРП трансформируется в ограничение целостности сразу для двух отношений (такого рода ограничения целостности называются ограничениями базы данных, и их поддержка гораздо более накладна с технической точки зрения). Понятно, что в *процессе нормализации* декомпозиция отношения на *независимые проекции*является предпочтительной. Необходимые и достаточные условия *независимости проекций отношения* обеспечивает *теорема Риссанена*.

Теорема Риссанена

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

- каждая FD в отношении r логически следует[4)] из FD в r1 и r2;
- общие атрибуты r1 и r2 образуют возможный ключ хотя бы для одного из этих отношений.

Мы не будем приводить доказательство этой *теоремы*, но продемонстрируем ее верность на примере двух показанных выше декомпозиций отношенияСЛУЖ. В первой декомпозиции (на проекции СЛУЖ1 и УРОВ) общий атрибут СЛУ_УРОВ является возможным (и первичным) ключом отношения УРОВ, а единственная дополнительная FD отношения СЛУЖ (СЛУ_НОМ⟶СЛУ_ЗАРП) логически следует из FD СЛУ_НОМ⟶ СЛУ_УРОВ и СЛУ_УРОВ⟶СЛУ_ЗАРП, выполняемых для отношений СЛУЖ1 и УРОВ соответственно. Вторая декомпозиция удовлетворяет второму условию *теоремы Риссанена* (СЛУ_НОМявляется первичным ключом в каждом из отношений СЛУЖ1 и СЛУ_ЗАРП), но FD СЛУ_УРОВ⟶СЛУ_ЗАРП не выводится из FD СЛУ_НОМ ⟶СЛУ_УРОВ и СЛУ_НОМ⟶СЛУ_ЗАРП.

Атомарным отношением называется отношение, которое невозможно декомпозировать на *независимые проекции*. Далеко не всегда для неатомарных (не являющихся *атомарными*) отношений требуется декомпозиция на атомарные проекции. Например, отношение СЛУЖ2 {СЛУ_НОМ, СЛУ_ЗАРП, ПРО_НОМ} с множеством FD {СЛУ_НОМ ⟶СЛУ_ЗАРП, СЛУ_НОМ⟶ПРО_НОМ} не является *атомарным* (возможна декомпозиция на *независимые проекции*СЛУЖ3 {СЛУ_НОМ, СЛУ_ЗАРП} и СЛУЖ4 {СЛУ_НОМ, ПРО_НОМ}). Но эта декомпозиция не улучшает свойства отношения СЛУЖ2 и поэтому не является осмысленной. Другими словами, при выборе способа декомпозиции нужно стремиться к получению *независимых проекций*, но не обязательно атомарных.

ерекрывающиеся возможные ключи и нормальная форма Бойса-Кодда

До сих пор в определениях нормальных форм мы предполагали, что у декомпозируемого отношения имеется только один возможный ключ. На практике чаще всего бывает именно так. Но имеется один частный случай, который (почти) удовлетворяет требованиям *2NF* и *3NF*, но, тем не менее, порождает*аномалии*

обновления. Это тот случай, когда у отношения имеется несколько возможных ключей, и некоторые из этих возможных ключей «перекрываются», т. е. содержат общие атрибуты.

Аномалии обновлений, связанные с наличием перекрывающихся возможных ключей

Например, пусть имеется переменная отношения СЛУЖ_ПРО_ЗАДАН1 {СЛУ_НОМ, СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН} с множеством FD, показанным на рис. 7.7.

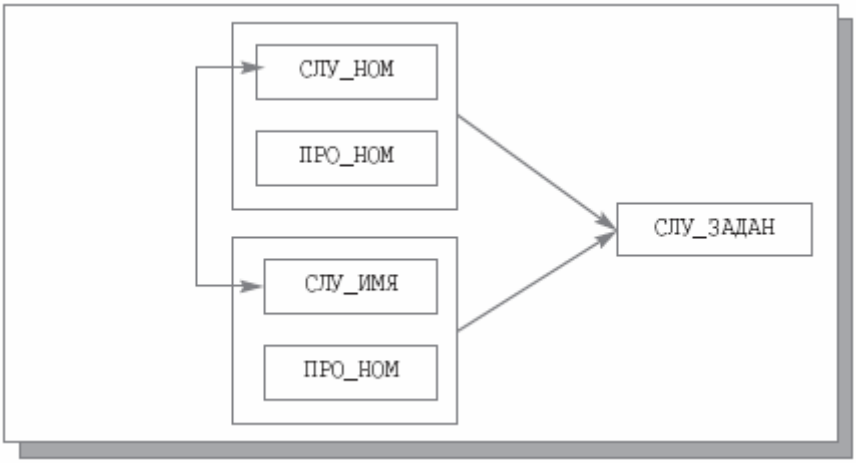

Рис. 7.7. Диаграмма FD отношения СЛУЖ_ПРО_ЗАДАН1

В отношении СЛУЖ_ПРО_ЗАДАН1 служащие уникально идентифицируются как по номерам удостоверений, так и по именам. Следовательно, существуют FD СЛУ_НОМ⟶ СЛУ_ИМЯ и СЛУ_ИМЯ⟶СЛУ_НОМ. Но один служащий может участвовать в нескольких проектах, поэтому возможными ключами являются{СЛУ_НОМ, ПРО_НОМ} и {СЛУ_ИМЯ, ПРО_НОМ}. На рис. 7.8 показано возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН1.

СЛУ_НОМ	СЛУ_ИМЯ	ПРО_НОМ	СЛУ_ЗАДАН
2934	Иванов	1	A
2941	Иваненко	2	B
2934	Иванов	2	B
2941	Иваненко	1	A

Рис. 7.8. Возможное значение переменной отношения СЛУЖ_ПРО_ЗАДАН1

Очевидно, что, хотя в отношении СЛУЖ_ПРО_ЗАДАН1 все FD неключевых атрибутов от возможных ключей являются минимальными и транзитивные FD отсутствуют, этому отношению свойственны *аномалии обновления*. Например, в случае изменения имени служащего требуется обновить атрибутСЛУ_ИМЯ во всех кортежах отношения СЛУЖ_ПРО_ЗАДАН1, соответствующих данному служащему. Иначе будет нарушена FD СЛУ_НОМ⟶СЛУ_ИМЯ, и база данных окажется в несогласованном состоянии.

Нормальная форма Бойса-Кодда

Причиной отмеченных *аномалий* является то, что в требованиях *2NF* и *3NF* не требовалась минимальная функциональная зависимость от первичного ключа

атрибутов, являющихся компонентами других возможных ключей. Проблему решает нормальная форма, которую исторически принято называть*нормальной формой Бойса-Кодда* и которая является уточнением *3NF* в случае наличия нескольких *перекрывающихся возможных ключей*.

Переменная отношения находится в ***нормальной форме Бойса-Кодда (BCNF)*** в том и только в том случае, когда любая выполняемая для этой переменной отношения нетривиальная и минимальная FD имеет в качестве детерминанта некоторый возможный ключ данного отношения.

Переменная отношения СЛУЖ_ПРО_ЗАДАН1 может быть приведена к *BCNF* путем одной из двух декомпозиций: СЛУЖ_НОМ_ИМЯ {СЛУ_НОМ, СЛУ_ИМЯ} иСЛУЖ_НОМ_ПРО_ЗАДАН {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН} с множеством FD и значениями, показанными на рис. 7.9, и СЛУЖ_НОМ_ИМЯ {СЛУ_НОМ, СЛУ_ИМЯ}и СЛУЖ_ИМЯ_ПРО_ЗАДАН {СЛУ_ИМЯ, ПРО_НОМ, СЛУ_ЗАДАН} (FD и значения результирующих переменных отношений выглядят аналогично).

Очевидно, что каждая из декомпозиций устраняет трудности, связанные с обновлением отношения СЛУЖ_ПРО_ЗАДАН1.

Всегда ли следует стремиться к BCNF?

Предположим теперь, что в организации все проекты включают разные задания, и по-прежнему каждый служащий может участвовать в нескольких проектах, но может выполнять в каждом проекте только одно задание. Одно задание в каждом проекте могут выполнять несколько служащих. Тогда переменная отношения СЛУЖ_ПРО_ЗАДАН имеет множество FD, показанное на рис. 7.10, и может содержать значение, представленное на том же рисунке.

В этом отношении существуют два возможных ключа: {СЛУ_НОМ, ПРО_НОМ} и {СЛУ_НОМ, СЛУ_ЗАДАН}. Отношение удовлетворяет требованиям *3NF*: отсутствуют неминимальные FD неключевых атрибутов от возможных ключей (поскольку нет неключевых атрибутов) и отсутствуют транзитивные FD. Однако из-за наличия FD СЛУ_ЗАДАН⟶ ПРО_НОМ это отношение не находится в *BCNF*. Поэтому отношению СЛУ_ПРО_ЗАДАН снова свойственны *аномалии обновления*. Например (поскольку СЛУ_НОМ является компонентом обоих возможных ключей), невозможно удалить данные о единственном служащем, выполняющем задание в некотором проекте, не утратив информацию об этом задании.

Рис. 7.9. Диаграммы FD и значения переменных отношений СЛУЖ_НОМ_ИМЯ и СЛУЖ_НОМ_ПРО_ЗАДАН

Можно привести отношение СЛУЖ_ПРО_ЗАДАН к *BCNF*, выполнив его декомпозицию на отношения СЛУЖ_НОМ_ЗАДАН {СЛУ_НОМ, СЛУ_ЗАДАН}[1] иПРО_НОМ_ЗАДАН {СЛУ_ЗАДАН, ПРО_НОМ}, и эта декомпозиция решает обозначенные проблемы (теперь можно хранить данные о задании проекта, не выполняемом ни одним служащим). Значения переменных отношений СЛУЖ_НОМ_ЗАДАН и ПРО_НОМ_ЗАДАН показаны на рис. 7.11.

Однако возникают новые трудности. Например, система должна запретить добавление в отношение СЛУЖ_НОМ_ЗАДАН кортежа <2934, D>, поскольку задание D относится к проекту 1, а служащий с номером 2934 уже выполняет задание в этом проекте. Так происходит, потому что исходная FD{СЛУ_НОМ, ПРО_НОМ}⟶СЛУ_ЗАДАН не выводится из единственной (нетривиальной) действующей для этих проекций FD СЛУ_ЗАДАН⟶ ПРО_НОМ, и соответствующее ограничение целостности становится ограничением базы данных.

Значение переменной отношения **СЛУЖ_ПРО_ЗАДАН** (новый вариант)

СЛУ_НОМ	ПРО_НОМ	ПРО_ЗАДАН
2934	1	A
2935	1	A
2941	2	B
2934	2	C
2941	1	D

Рис. 7.10. Новый вариант переменной отношения СЛУЖ_ПРО_ЗАДАН

Тем самым, проекции СЛУЖ_НОМ_ЗАДАН и ПРО_НОМ_ЗАДАН не являются *независимыми*, а отношение СЛУЖ_ПРО_ЗАДАН *атомарно*, хотя и не находится в*BCNF*. Из этого следует, что при проектировании реляционной базы данных приведение отношения к *BCNF* не должно быть самоцелью. Нужно внимательно оценивать положительные и отрицательные последствия нормализации.

Наконец, приведем пример, когда наличие двух *перекрывающихся возможных ключей* не мешает отношению находиться в *BCNF*. Предположим, что в организации проекты включают одни и те же задания, каждый служащий может участвовать в нескольких проектах, но может выполнять в каждом проекте только одно задание. Тогда переменная отношения СЛУЖ_НОМ_ЗАДАН имеет множество FD, показанное на рис. 7.12, и может содержать значение, показанное на том же рисунке.

В третьем варианте отношения СЛУЖ_НОМ_ЗАДАН имеются *перекрывающиеся возможные ключи* ({СЛУ_НОМ, ПРО_НОМ} и {ПРО_НОМ, СЛУ_ЗАДАН}), однако оно находится в *BCNF*, поскольку эти ключи являются единственными детерминантами. Легко убедиться, что отношению СЛУЖ_НОМ_ЗАДАН *аномалии обновления* не свойственны.

Значение переменной отношения **СЛУЖ_НОМ_ЗАДАН**

СЛУ_НОМ	ПРО_ЗАДАН
2934	A
2935	A
2941	B
2934	C
2941	D

Значение переменной отношения **ПРО_НОМ_ЗАДАН**

ПРО_НОМ	ПРО_ЗАДАН
1	A
2	B
2	C
1	D

Рис. 7.11. Значения переменных отношений СЛУЖ_НОМ_ЗАДАН и ПРО_НОМ_ЗАДАН

Диаграмма FD отношения **СЛУЖ_НОМ_ЗАДАН** (третий вариант)

Возможное значение переменной отношения **СЛУЖ_НОМ_ЗАДАН** (третий вариант)

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2935	1	B
2941	2	B
2934	2	A
2941	1	D

Рис. 7.12. Третий вариант отношения СЛУЖ_НОМ_ЗАДАН

Заключение

В этой лекции мы обсудили три начальные нормальные формы отношений – *вторую* и *третью нормальные формы* и *нормальную форму Бойса-Кодда*, – которые производятся путем декомпозиции без потерь исходного отношения на две проекции, где отсутствуют *аномалии* изменений, существовавшие в исходном отношении по причине наличия функциональных зависимостей с нежелательными свойствами.

Нормализация схемы базы данных способствует более эффективному выполнению системой управления базами данных операций обновления базы данных, поскольку сокращается число проверок и вспомогательных действий, поддерживающих целостность базы данных. При проектировании реляционной базы данных почти всегда добиваются *второй нормальной формы* всех входящих в базу данных отношений. В часто обновляемых базах данных обычно стараются обеспечить *третью нормальную*

форму отношений. На *нормальную форму Бойса-Кодда* внимание обращают гораздо реже, поскольку на практике ситуации, в которых у отношения имеется несколько составных *перекрывающихся возможных ключей*, встречаются нечасто.

24. Инфологическое проектирование. Основные этапы проэктирования БД с помощью метода «сущность-связь».

Основными понятиями метода сущность-связь являются следующие:

- сущность,
- атрибут сущности,
- ключ сущности,
- связь между сущностями,
- степень связи,
- класс принадлежности экземпляров сущности,
- диаграммы ER-экземпляров,
- диаграммы ER-типа.

Сущность представляет собой объект, информация о котором хранится в БД. Экземпляры сущности отличаются друг от друга и однозначно идентифицируются. Названиями сущностей являются, как правило, *существительные,* например: ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА, КАФЕДРА, ГРУППА.

Атрибут представляет собой свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так, атрибутами сущности ПРЕПОДАВАТЕЛЬ может быть его Фамилия, Должность, Стаж (преподавательский) и т. д.

Ключ сущности - атрибут или набор атрибутов, используемый для идентификации экземпляра сущности. Как видно из определения, понятие ключа сущности аналогично понятию ключа отношения.

Связь двух или более сущностей - предполагает зависимость между атрибутами этих сущностей. Название связи обычно представляется *глаголом.* Примерами связей между сущностями являются следующие: ПРЕПОДАВАТЕЛЬ *ВЕДЕТ* ДИСЦИПЛИНУ (Иванов ВЕДЕТ «Базы данных»), ПРЕПОДАВАТЕЛЬ *ПРЕПОДАЕТ-В* ГРУППЕ (Иванов ПРЕПОДАЕТ-В 256 группе), ПРЕПОДАВАТЕЛЬ *РАБОТАЕТ-НА* КАФЕДРЕ (Иванов РАБОТАЕТ-НА 25 кафедре).

Приведенные определения сущности и связи не полностью формализованы, но приемлемы для практики. Следует иметь в виду, что в результате проектирования могут быть получены несколько вариантов одной БД. Так, два разных проектировщика, рассматривая одну и ту же проблему с разных точек зрения, могут получить различные наборы сущностей и связей. При этом оба варианта могут быть рабочими, а выбор лучшего из них будет результатом личных предпочтений.

С целью повышения наглядности и удобства проектирования для представления сущностей, экземпляров сущностей и связей между ними используются следующие графические средства:

- *диаграммы ER-экземпляров,*
- *диаграммы ER-типа,* или *ER-диаграммы.*

Степень связи является характеристикой связи между сущностями, которая может быть типа: 1:1, 1:M, M:1, M:M.

Класс принадлежности (КП) сущности может быть: *обязательным* и *ие-обязателъным.*

Класс принадлежности сущности является *обязательным,* если все экземпляры этой сущности обязательно участвуют в рассматриваемой связи, *и* противном случае класс принадлежности сущности является *необязательным.*

Этапы проектирования.

Процесс проектирования базы данных является итерационным - допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений и включает следующие этапы:

1. Выделение сущностей и связей между ними.
2. Построение диаграмм ER-типа с учетом всех сущностей и их связей.
3. Формирование набора предварительных отношений с указанием пред полагаемого первичного ключа для каждого отношения и использованием диаграмм ER-типа.

4. Добавление неключевых атрибутов в отношения.

5. Приведение предварительных отношений к нормальной форме Бойса - Кодда, например, с помощью метода нормальных форм.

6. Пересмотр ER-диаграмм в следующих случаях:

- некоторые отношения не приводятся к нормальной форме Бойса - Кодда;
- некоторым атрибутам не находится логически обоснованных мест в пред варительных отношениях.

После преобразования ER-диаграмм осуществляется повторное выполнение предыдущих этапов проектирования (возврат к этапу 1).

Одним из узловых этапов проектирования является этап формирования отношений. Рассмотрим процесс формирования предварительных отношений, составляющих первичный вариант схемы БД.

В рассмотренных выше примерах связь ВЕДЕТ всегда соединяет две сущности и поэтому является *бинарной.* Сформулированные ниже правила формирования отношений из диаграмм ER

25. Даталогическое проектирование. Правила формирования отношений по ER-диаграммам.

Даталогическое проектирование

В реляционных БД даталогическое или логическое проектирование приводит к разработке схемы БД, то есть совокупности схем отношений, которые адекватно моделируют абстрактные объекты предметной области и семантические связи между этими объектами. Основой анализа корректности схемы являются так называемые функциональные зависимости между атрибутами БД. Некоторые зависимости между атрибутами отношений являются нежелательными из-за побочных эффектов и аномалий, которые они вызывают при модификации БД. При этом под процессом модификации БД мы понимаем внесение новых данных в БД или удаление некоторых данных из БД, а также обновление значений некоторых атрибутов.

Однако этап логического или даталогического проектирования не заканчивается проектированием схемы отношений. В общем случае в результате выполнения этого этапа должны быть получены следующие результирующие документы:

- Описание концептуальной схемы БД в терминах выбранной СУБД.
- Описание внешних моделей в терминах выбранной СУБД.
- Описание декларативных правил поддержки целостности базы данных.
- Описание процедур поддержки семантической целостности базы данных.

Однако перед тем как описывать построенную схему в терминах выбранной СУБД, нам надо выстроить эту схему. Именно этому процессу и посвящен данный раздел. Мы должны построить корректную схему БД, ориентируясь на реляционную модель данных.

ОПРЕДЕЛЕНИЕ
Корректной назовем схему БД, в которой отсутствуют нежелательные зависимости между атрибутами отношений.

Процесс разработки корректной схемы реляционной БД называется *логическим проектированием БД*.

Проектирование схемы БД может быть выполнено двумя путями:

- *путем декомпозиции (разбиения)*,когда исходное множество отношений, входящих в схему БД заменяется другим множеством отношений (число их при этом возрастает), являющихся проекциями исходных отношений;
- *путем синтеза*,то есть путем компоновки из заданных исходных элементарных зависимостей между объектами предметной области схемы БД.

Классическая технология проектирования реляционных баз данных связана с теорией нормализации, основанной на анализе функциональных зависимостей между атрибутами отношений. Понятие функциональной зависимости является фундаментальным в теории нормализации реляционных баз данных. Мы определим его далее, а пока коснемся смысла этого понятия. Функциональные зависимости определяют устойчивые отношения между объектами и их свойствами в рассматриваемой предметной области. Именно поэтому процесс поддержки функциональных зависимостей, характерных для данной предметной области, является базовым для процесса проектирования.

Процесс проектирования с использованием декомпозиции представляет собой процесс последовательной нормализации схем отношений, при этом каждая последующая

итерация соответствует нормальной форме более высокого уровня и обладает лучшими свойствами по сравнению с предыдущей.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса—Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или форма проекции-соединения (5NF или PJNF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе классического процесса проектирования лежит последовательность переходов от предыдущей нормальной формы к последующей. Однако в процессе декомпозиции мы сталкиваемся с проблемой *обратимости*,то есть возможности восстановления исходной схемы. Таким образом, декомпозиция должна
сохранять *эквивалентность* схем БД при замене одной схемы на другую.

ОПРЕДЕЛЕНИЕ
Схемы БД называются *эквивалентными*,если содержание исходной БД может быть получено путем естественного соединения отношений, входящих в результирующую схему, и при этом не появляется новых кортежей в исходной БД.

При выполнении эквивалентных преобразований сохраняется множество исходных фундаментальных функциональных зависимостей между атрибутами отношений.

Функциональные зависимости определяют не текущее состояние БД, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые присущи реальному объекту, который моделируется с помощью БД.

Поэтому определить функциональные зависимости по текущему состоянию БД можно только в том случае, если экземпляр БД содержит абсолютно полную информацию (то есть никаких добавлений и модификации БД не предполагается). В реальной жизни это требование невыполнимо, поэтому набор функциональных зависимостей задает разработчик, системный аналитик, исходя из глубокого системного анализа предметной области.

Приведем ряд основных определений.

Функциональной зависимостью набора атрибутов B отношения `R` от набора атрибутов `A` того же отношения, обозначаемой как `R.A -> R.B` или `A -> B`

называется такое соотношение проекций `R[A]` и `R[B]`, при котором в каждый момент времени любому элементу проекции `R[A]` соответствует только один элемент проекции `R[B]`, входящий вместе с ним в какой-либо кортеж отношения `R`.

Функциональная зависимость `R.A -> R.B` называется *полной*,если набор атрибутов `B` функционально зависит от `A` и не зависит функционально от любого подмножества `A`, то есть `R.A -> R.B` называется полной, если:

$$\forall A1 \subseteq A \rightarrow R.A - / \rightarrow R.B$$,

что читается следующим образом:

для любого `A1`, являющегося подмножеством `A`, `R.B` функционально не зависит от `R.A`, в противном случае зависимость `R.A -> R.B` называется неполной.

Функциональная зависимость `R.A -> R.B` называется *транзитивной*, если существует набор атрибутов C такой, что:

1. C не является подмножеством `A`.
2. C не включает в себя `B`.
3. Существует функциональная зависимость `R.A -> R.C`.
4. Не существует функциональной зависимости `R.C -> R.A`.
5. Существует функциональная зависимость `R.C -> R.B`.

Возможным ключом отношения называется набор атрибутов отношения, который полностью и однозначно (функционально полно) определяет значения всех остальных атрибутов отношения, то есть возможный ключ — это набор атрибутов, однозначно определяющий кортеж отношения, и при этом при удалении любого атрибута из этого набора его свойство однозначной идентификации кортежа теряется.

А может ли быть ситуация, когда отношение не имеет возможного ключа? Давайте вспомним определение отношения: *отношение — это подмножество декартова произведения множества доменов*. И в полном декартовом произведении все наборы значений различны, тем более в его подмножестве. Значит, обязательно для каждого отношения всегда существует набор атрибутов, по которому можно однозначно определить кортеж отношения. В вырожденном случае это просто полный набор атрибутов отношения, потому что если мы зададим для всех атрибутов конкретные значения, то, по определению отношения, мы получим только один кортеж.

В общем случае в отношении может быть несколько возможных ключей.

Среди всех возможных ключей отношения обычно выбирают один, который считается главным и который называют *первичным ключом отношения*.

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав ни одного возможного ключа отношения.

Взаимно-независимые атрибуты — это такие атрибуты, которые не зависят функционально один от другого.

Если в отношении существует несколько функциональных зависимостей, то каждый атрибут или набор атрибутов, от которого зависит другой атрибут, называется *детерминантом* отношения.

Для функциональных зависимостей как фундаментальной основы проекта БД были проведены исследования, позволяющие избежать избыточного их представления. Ряд зависимостей могут быть выведены из других путем применения правил, названных аксиомами Армстронга, по имени исследователя, впервые сформулировавшего их. Это три основных аксиомы:

1. *Рефлексивность*: если B является подмножеством A, то A -> B
2. *Дополнение*: если A->B, то A.C -> B.C
3. *Транзитивность*: если A -> B и B -> C, то A -> C.

Доказано, что данные правила являются полными и исчерпывающими, то есть, применяя их, из заданного множества функциональных зависимостей можно вывести все возможные функциональные зависимости.

Множество всех возможных функциональных зависимостей, выводимое из заданного набора исходных функциональных зависимостей, называется его замыканием.

ОПРЕДЕЛЕНИЕ
Отношение находится в первой нормальной форме тогда и только тогда, когда на пересечении каждого столбца и каждой строки находятся только элементарные значения атрибутов.

В некотором смысле это определение избыточно, потому что собственно оно определяет само отношение в теории реляционных баз данных. Однако в силу исторически сложившихся обстоятельств и для преемственности такое определение первой нормальной формы существует и мы должны с ним согласиться. Отношения, находящиеся в первой нормальной форме, часто называют просто нормализованными отношениями. Соответственно, ненормализованные отношения могут интерпретироваться как таблицы с неравномерным заполнением, например таблица "Расписание", которая имеет вид:

Преподаватель	День недели	Номер пары	Название дисциплины	Тип занятий	Группа
Петров В. И.	Понед.	1	Теор. выч. проц.	Лекция	4906
	Вторник	1	Комп. графика	Лаб. раб..	4907
	Вторник	2	Комп. графика	Лаб. раб	4906
Киров В. А.	Понед.	2	Теор. информ.	Лекция	4906
	Вторник	3	Пр-е на C++	Лаб. раб.	4907
	Вторник	4	Пр-е на C++	Лаб. раб.	4906
Серов А. А.	Понед.	3	Защита инф.	Лекция.	4944
	Среда	3	Пр-е на VB	Лаб. раб	4942
	Четверг	4	Пр-е на VB	Лаб. раб.	4922

Здесь на пересечении одной строки и одного столбца находится целый набор элементарных значений, соответствующих набору дней, перечню пар, набору дисциплин, по которым проводит занятия один преподаватель.

Для приведения отношения "Расписание" к первой нормальной форме необходимо дополнить каждую строку фамилией преподавателя.

Преподаватель	День недели	Номер пары	Название дисциплины	Тип занятий	Группа
Петров В. И.	Понед.	1	Теор. выч. проц.	Лекция	4906
Петров В. И.	Вторник	1	Комп. графика	Лаб. раб..	4907
Петров В. И.	Вторник	2	Комп. графика	Лаб. раб	4906
Киров В. А.	Понед.	2	Теор. информ.	Лекция	4906
Киров В. А.	Вторник	3	Пр-е на C++	Лаб. раб.	4907
Киров В. А.	Вторник	4	Пр-е на C++	Лаб. раб.	4906
Серов А. А.	Понед.	3	Защита инф.	Лекция.	4944
Серов А. А.	Среда	3	Пр-е на VB	Лаб. раб	4942
Серов А. А.	Четверг	4	Пр-е на VB	Лаб. раб.	4922

ОПРЕДЕЛЕНИЕ
Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в первой нормальной форме и не содержит неполных функциональных зависимостей непервичных атрибутов от атрибутов первичного ключа.

Рассмотрим отношение, моделирующее сдачу студентами текущей сессии. Структура этого отношения определяется следующим набором атрибутов:

(ФИО, Номер зач.кн., Группа, Дисциплина, Оценка)
Так как каждый студент сдает целый набор дисциплин в процессе сессии, то первичным ключом отношения может быть (Номер. зач.кн., Дисциплина), который однозначно определяет каждую строку отношения. С другой стороны, атрибуты ФИО и Группа зависят только от части первичного ключа — от значения атрибута Номер зач. кн., поэтому мы должны констатировать наличие неполных функциональных зависимостей в данном отношении. Для приведения данного отношения ко второй нормальной форме следует разбить его на проекции, при этом должно быть соблюдено условие восстановления исходного отношения без потерь. Такими проекциями могут быть два отношения:

(ФИО, Номер.зач.кн., Группа) (Номер зач.кн., Дисциплина, Оценка)
Этот набор отношений не содержит неполных функциональных зависимостей, и поэтому эти отношения находятся во второй нормальной форме.

А почему надо приводить отношения ко второй нормальной форме? Иначе говоря, какие аномалии или неудобства могут возникнуть, если мы оставим исходное отношение и не будем его разбивать на два? Давайте рассмотрим ситуацию, когда студент переведен из одной группы в другую. Тогда в первом случае (если мы не разбивали исходное отношение на два) мы должны найти все записи с данным студентом и в них изменить значение атрибута Группа на новое. Во втором же случае меняется только один кортеж в первом отношении. И конечно, опасность нарушения корректности (непротиворечивости содержания) БД в первом случае выше. Может получиться так, что часть кортежей поменяет значения атрибута Группа, а часть по причине сбоя в работе аппаратуры останется в старом состоянии. И тогда наша БД будет содержать записи, которые относят одного студента одновременно к разным группам. Чтобы этого не произошло, мы должны принимать дополнительные непростые меры, например организовывать процесс согласованного изменения с использованием сложного механизма транзакций, который мы будем рассматривать в лекциях, посвященных вопросам распределенного доступа к БД. Если же мы перешли ко второй нормальной форме, то мы меняем только один кортеж. Кроме того, если у нас есть студенты, которые еще не сдавали экзамены, то в исходном отношении мы вообще не можем хранить о них информацию, а во второй схеме информация о студентах и их принадлежности к конкретной группе хранится отдельно от информации, которая связана со сдачей экзаменов, и поэтому мы можем в этом случае отдельно работать со студентами и отдельно хранить и обрабатывать информацию об успеваемости и сдаче экзаменов, что в действительности и происходит.

ОПРЕДЕЛЕНИЕ
Отношение находится в третьей нормальной форме тогда и только тогда, когда оно находится во второй нормальной форме и не содержит транзитивных зависимостей.

Рассмотрим отношение, связывающее студентов с группами, факультетами и специальностями, на которых он учится.

```
(ФИО, Номер зач.кн., Группа, Факультет, Специальность, Выпускающая кафедра)
```

Первичным ключом отношения является `Номер зач.кн.`, однако рассмотрим остальные функциональные зависимости. Группа, в которой учится студент, однозначно определяет факультет, на котором он учится, а также специальность и выпускающую кафедру. Кроме того, выпускающая кафедра однозначно определяет факультет, на котором обучаются студенты, выпускаемые по данной кафедре. Но если мы предположим, что одну специальность могут выпускать несколько кафедр, то специальность не определяет выпускающую кафедру. В этом случае у нас есть следующие функциональные зависимости:

```
Номер зач.кн. -> ФИО
Номер зач.кн. -> Группа
Номер зач.кн. -> Факультет
Номер зач.кн. -> Специальность
Номер зач.кн. -> Выпускающая кафедра
Группа -> Факультет
Группа -> Специальность
Группа -> Выпускающая кафедра
Выпускающая кафедра -> Факультет
```

И эти зависимости образуют транзитивные группы. Для того чтобы избежать этого, мы можем предложить следующий набор отношений:

```
(Номер.зач.кн., ФИО, Специальность, Группа)
(Группа, Выпускающая кафедра)
(Выпускащая кафедра, Факультет)
```

Первичные ключи отношений выделены.

Теперь необходимо удостовериться, что при естественном соединении мы не потеряем ни одной строки и не получим лишних кортежей. И это упражнение я предлагаю выполнить вам самостоятельно.

Полученный набор отношений находится в третьей нормальной форме.

ОПРЕДЕЛЕНИЕ
Отношение находится в нормальной форме Бойса—Кодда, если оно находится в третьей нормальной форме и каждый детерминант отношения является возможным ключом отношения.

Рассмотрим отношение, моделирующее сдачу студентом текущих экзаменов. Предположим, что студент может сдавать экзамен по одной дисциплине несколько раз, если он получил неудовлетворительную оценку. Допустим, что во избежание возможных полных однофамильцев мы можем однозначно идентифицировать студента номером его зачетной книги, но, с другой стороны, у нас ведется электронный учет текущей успеваемости студентов, поэтому каждому студенту -присваивается в период его обучения в вузе уникальный номер-идентификатор. Отношение, которое моделирует сдачу текущей сессии, имеет следующую структуру:

```
(Номер зач.кн., Идентификатор_студента, Дисциплина, Дата, Оценка)
```

Возможными ключами отношения являются `Номер_зач.кн, Дисциплина, Дата` и `Идентификатор_студента, Дисциплина, Дата`.

Какие функциональные зависимости у нас имеются?

```
Номер_зач.кн, Дисциплина, Дата -> Оценка;
```

```
Идентификатор_студента, Дисциплина, Дата -> Оценка;
Номер зач.кн. -> Идентификатор_студента;
Идентификатор_студента -> Номер зач.кн.
```

- Откуда взялись две последние функциональные зависимости? Но ведь мы предварительно описали, что каждому студенту ставится в соответствие один номер зачетной книжки и один `Идентификатор_студента`, поэтому по значению `Номер зач.кн`. можно однозначно определить`Идентификатор_студента` (это третья зависимость) и обратно (и это четвертая зависимость). Оценим это отношение.
- Это отношение находится в третьей нормальной форме, потому что неполных функциональных зависимостей непервичных атрибутов от атрибутов возможного ключа здесь не присутствует и нет транзитивных зависимостей. А как же третья и четвертая зависимости, разве они не являются неполными? Нет, потому что зависимым не является непервичный атрибут, то есть атрибут, не входящий ни в один возможный ключ. Поэтому придраться к этому мы не можем. Но вот под четвертую нормальную форму наше отношение не подходит, потому что у нас есть два детерминанта`Номер зач.кн`.и `Идентификатор_студента`, которые не являются возможными ключами отношения. Для приведения отношения к нормальной форме Бойса—Кодда надо разделить отношение, например, на два со следующими схемами:
  - `(Идентификатор_студента, Дисциплина, Дата, Оценка)`
    `(Номер зач.кн., Идентификатор_студента)`

    или наоборот:

    ```
    (Номер зач.кн., Дисциплина, Дата, Оценка)
    (Номер зач.кн., Идентификатор_студента)
    ```

    Эти схемы равнозначны с точки зрения теории нормализации, поэтому выбирать проектировщикам следует исходя из некоторых дополнительных рассуждений. Ну, например, если учесть, что зачетные книжки могут теряться, то как они будут восстанавливаться: если с тем же самым номером, то нет разницы, но если с новым номером, то тогда первая схема предпочтительней.

В большинстве случаев достижение третьей нормальной формы или даже формы Бойса—Кодда считается достаточным для реальных проектов баз данных, однако в теории нормализации существуют нормальные формы высших порядков, которые уже связаны не с функциональными зависимостями между атрибутами отношений, а отражают более тонкие вопросы семантики предметной области и связаны с другими видами зависимостей. Прежде чем перейти к рассмотрению нормальных форм высших порядков, дадим еще несколько определений.

ОПРЕДЕЛЕНИЕ
В отношении `R (A, B, C)` существует *многозначная зависимость (multi valid dependence, MVD)* `R.A ->> R.B` в том и только в том случае, если множество значений B, соответствующее паре значений `A` и `C`, зависит только от `A` и не зависит от `C`.

Когда мы рассматривали функциональные зависимости, то каждому значению детерминанта соответствовало только одно значение зависимого от него атрибута. При рассмотрении многозначных зависимостей мы выделяем случаи, когда одному значению некоторого атрибута соответствует устойчиво постоянное множество значений другого атрибута. Когда это может быть? Рассмотрим конкретную ситуацию, понятную всем студентам. Пусть дано отношение, которое моделирует предстоящую сдачу экзаменов на сессии. Допустим, оно имеет вид:

```
(Номер зач.кн., Группа, Дисциплина)
```

Перечень дисциплин, которые должен сдавать студент, однозначно определяется не его фамилией, а номером группы (то есть специальностью, на которой он учится).

В данном отношении существуют следующие две многозначные зависимости:

```
Группа ->> Дисциплина
Группа ->> Номер зач.кн.
```
Это означает, что каждой группе однозначно соответствует перечень дисциплин по учебному плану и номер группы определяет список студентов, которые в этой группе учатся.

Если мы будем работать с исходным отношением, то мы не сможем хранить информацию о новой группе и ее учебном плане — перечне дисциплин, которые должна пройти группа до тех пор, пока в нее не будут зачислены студенты. При изменении перечня дисциплин по учебному плану, например при добавлении новой дисциплины, внести эти изменения в отношение для всех студентов, занимающихся в данной группе, весьма затруднительно. С другой стороны, если мы добавляем студента в уже существующую группу, то мы должны добавить множество кортежей, соответствующих перечню дисциплин для данной группы. Эти аномалии модификации отношения как раз и связаны с наличием двух многозначных зависимостей.

В теории реляционных баз данных доказывается, что в общем случае в отношении `R (A, B, C)` существует многозначная зависимость `R.A ->> R.B` в том и только в том случае, когда существует многозначная зависимость `R.A ->> R.C`.

Дальнейшая нормализация отношений, подобных нашему, основывается на теореме Фейджина.

ТЕОРЕМА ФЕЙДЖИНА
Отношение `R (A, B, C)` можно спроецировать без потерь в отношения `R1 (A, B)` и `R2 (A, C)` в том и только в том случае, когда существует MVD `A ->> B | C` (что равнозначно наличию двух зависимостей `A ->> B` и `A ->> C`).

Под проецированием без потерь понимается такой способ декомпозиции отношения путем применения операции проекции, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений. Практически теорема доказывает наличие эквивалентной схемы для отношения, в котором существует несколько многозначных зависимостей.

ОПРЕДЕЛЕНИЕ
Отношение `R` находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости `A ->> B` все остальные атрибуты `R` функционально зависят от `A`.

В нашем примере можно произвести декомпозицию исходного отношения в два отношения:

```
(Номер зач.кн., Группа)
(Группа, Дисциплина)
```
Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий. Действительно, обе операции модификации теперь упрощаются: добавление нового студента связано с добавлением всего одного кортежа в первое отношение, а добавление новой дисциплины выливается в добавление одного кортежа во второе отношение, кроме того, во втором отношении мы можем хранить любое количество групп с определенным перечнем дисциплин, в которые пока еще не зачислены студенты.

Последней нормальной формой является пятая нормальная форма 5NF, которая связана с анализом нового вида зависимостей, *зависимостей "проекции соединения" (project-join зависимости*,обозначаемые как *PJ-зависимости)*.Этот вид

зависимостей является в некотором роде обобщением многозначных зависимостей.

ОПРЕДЕЛЕНИЕ
Отношение R (X, Y,..., Z) удовлетворяет зависимости соединения (X, Y,..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z. Здесь X, Y, ..., Z — наборы атрибутов отношения R.

Наличие PJ-зависимости в отношении делает его в некотором роде избыточным и затрудняет операции модификации.

ОПРЕДЕЛЕНИЕ
Отношение R находится *в пятой нормальной форме* (нормальной форме проекции-соединения — PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Рассмотрим отношение R1:

R1(Преподаватель, Кафедра, Дисциплина)
Предположим, что каждый преподаватель может работать на нескольких кафедрах и на каждой кафедре может вести несколько дисциплин. В этом случае ключом отношения является полный набор из трех атрибутов. В отношении отсутствуют многозначные зависимости, и поэтому отношение находится в 4NF.

Введем следующие обозначения наборов атрибутов:

ПК (Преподаватель, Кафедра)
ПД (Преподаватель, Дисциплина)
КД (Кафедра, Дисциплина)
Допустим, что отношение R1 удовлетворяет зависимости проекции соединения (ПК, ПД, КД). Тогда отношение R1 не находится в NF/PJ, потому что единственным ключом его является полный набор атрибутов, а наличие зависимости PJ связано с наборами атрибутов, которые не составляют возможные ключи отношения R1. Для того чтобы привести это отношение к NF/PJ, его надо представить в виде трех отношений:

R2 (Преподаватель, Кафедра)
R3 (Преподаватель, Дисциплина)
R4 (Кафедра, Дисциплина)
Пятая нормальная форма редко используется на практике. В большей степени она является теоретическим исследованием. Очень тяжело определить само наличие зависимостей "проекции—соединения", потому что утверждение о наличии такой зависимости делается для всех возможных состояний БД, а не только для текущего экземпляра отношения R1. Однако знание о возможном наличии подобных зависимостей, даже теоретическое, нам все же необходимо.