Белорусский государственный университет

МОДЕЛИ ДАННЫХ И СУБД

Учебное пособие

Для студентов университетов
Специальностей
«Информатика»,
«Прикладная математика»,
«Компьютерная безопасность»,
«Актуарная математика» и
«Экономическая кибернетика

Минск 2007

Авторы: А. Н. Исаченко, С. П. Бондаренко

Рецензенты: Л.Ф. Зимянин, Н.А. Разоренов

В пособии рассматриваются классические модели данных, этапы проектирования и методы проектирования баз данных, построение семантических моделей данных, проектирование реляционных баз данных на основе принципов нормализации, язык запросов к базам данных SQL, основные функции СУБД, распределенные базы данных, администрирование баз данных. Как пример современной объектно-реляционной СУБД рассматривается СУБД Oracle.

Пособие предназначено для студентов специальностей «Информатика», «Прикладная математика», «Компьютерная безопасность», «Актуарная математика» и «Экономическая кибернетика».

ВВЕДЕНИЕ

Современный мир информационных технологий трудно представить без использования баз данных. Практически все системы в той или иной степени связаны с функциями долговременного хранения и обработки информации. Информация становится фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. Настоящее пособие подготовлено по материалам лекционных курсов, рассматривающих основы теории баз данных, языка запросов SQL и его процедурного расширения – языка PL/SQL системы управления базами данных Oracle, и состоит из 13 разделов.

Раздел 1 пособия посвящен истории возникновения области знаний, связанной с базами данных. Здесь даются определения ключевых понятий, рассматривается классическая трехуровневая архитектура, используемая в системах баз данных.

В разделе 2 приводится классификация моделей, используемых в системах баз данных. Подробно рассматриваются теоретико-графовые модели, которые использовались в ранних системах управления базами данных; реляционная модель, которая является основой практически для всех коммерческих систем управления базами данных (СУБД) и наиболее распространена в настоящий момент. Даются основные положения объектно-реляционных и многомерных моделей.

В разделе 3 дается описание первого языка манипулирования данными, предложенного для реляционной модели ее создателем, американским математиком Е. Ф. Коддом – реляционной алгебры.

Раздел 4 посвящен вопросам проектирования баз данных на основе принципов нормализации, в нем рассматриваются базовые понятия функциональных и многозначных зависимостей между свойствами объектов, которые моделируются в базе данных; рассматриваются нормальные формы схем отношений.

Раздел 5 посвящен семантическим или инфологическим моделям, используемым в современных программных системах поддержки проектирования, называемых CASE-системами (Computer Aided Software Engineering).

Разделы 6 и 7 посвящены изложению структуры СУБД и основным функциям СУБД. Подробно рассмотрены понятие транзакции, которое является базовым при выполнении параллельных запросов к базам данных, свойства транзакций и проблемы, возникающие при параллельном выполнении транзакций.

В разделе 8 излагаются вопросы журнализации и восстановления на ее основе информации базы данных, искаженной в результате сбоев вычислительной системы.

Раздел 9 посвящен вопросам защиты информации в базах данных. Понятие защиты информации в базах данных чаще всего связано с концепцией защиты от несанкционированного доступа. В данном разделе обсуждается общая концепция защиты информации, которая применяется в базах данных, вводится понятие пользователя и рассматриваются вопросы определения прав и привилегий пользователей по работе с отдельными объектами в базе данных.

Раздел 10 посвящен вопросам распределенной обработки данных, здесь рассматривается проектирование распределенных систем обработки данных, уделяется большое внимание фрагментации данных.

В разделе 11 излагаются вопросы, касающиеся основных положений работы с объектно-реляционной СУБД Oracle. Рассматриваются архитектура базы данных, объекты базы данных, архитектура экземпляра Oracle.

Раздел 12 полностью посвящен современному стандартному языку работы с базами данных, языку SQL. Рассматриваются основные элементы языка, создание и модификация объектов базы данных, построение запросов к базе данных.

В разделе 13 рассматривается процедурное расширение языка SQL – язык PL/SQL, который используется в Oracle при разработке триггеров, хранимых процедур и функций, пакетов, а также для работы с объектами их – созданию, хранению в базе данных и выборке.

Материал пособия включает достаточно большое количество примеров и может быть использован для самостоятельного освоения курса «Модели данных и СУБД».

1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ТЕОРИИ БАЗ ДАННЫХ

1.1. ПРИЧИНЫ ВОЗНИКНОВЕНИЯ СИСТЕМ БАЗ ДАННЫХ

Основой решения большинства задач является обработка информации. ***Информация*** – это совокупность фактов, наблюдений, сведений об объектах, явлениях, событиях реального мира. Для обработки информации создаются ***информационные системы***, которые воспринимают информацию из окружающей среды, хранят, обрабатывают ее и выдают в окружающую среду. Как правило, обработке подвергается информация, относящаяся к одной определенной ***предметной области***, т. е. к некоторой области знаний, имеющей практическую ценность для пользователя. В 1960-х гг. появились первые автоматизированные информационные системы (АИС), включающие в свой состав вычислительную технику. Массивы информации в АИС, для их компьютерного хранения и обработки, необходимо оптимальным образом организовывать, обеспечивать их целостность и непротиворечивость. Решение подобных задач с нужной производительностью невозможно осуществлять, используя только функции стандартных файловых систем.

Укажем очевидные недостатки файловой организации данных на внешних носителях.

1. *Изолированность и разделенность данных.* Операционная система контролирует и разграничивает доступ к данным, как правило, на уровне файлов. Поэтому несколько параллельно работающих приложений не смогут одновременно обновлять различные записи в одном и том же файле. Операции совместной обработки нескольких файлов так же достаточно сложны.
2. *Зависимость программ от данных.* Поскольку описание структуры данных задается в прикладной программе, то любое внесение изменений в эту структуру требует, как минимум, перекомпиляции всех программ, использующих этот файл.
3. *Дублирование данных.* В случае, когда различные приложения используют данные, относящиеся к одному и тому же объекту, но хранят эту информацию в своих независимых файлах, возникает неконтролируемое дублирование данных. Это приводит к непроизводительному расходу памяти на внешних устройствах и, что гораздо опаснее, может привести к противоречивости данных. Данные, измененные в одном файле, в другом файле могут остаться в прежнем виде.
4. *Отсутствие описаний данных.* В файлах операционной системы данные, обрабатываемые прикладными программами, хранятся без опи-

сания. Это создает сложности при документировании системы, затрудняет поиск нужных данных и приводит к появлению ошибок.

Для устранения перечисленных недостатков было предложено отделить процесс хранения данных от процесса их обработки, выделив специальную программу-посредник – ***систему управления базами данных*** (СУБД), а сами данные организовать и хранить в виде определенной структуры – ***базы данных***.

1.2. БАЗЫ ДАННЫХ

Предметная область АИС «материализуется» в форме хранимой в памяти ЭВМ структурированной совокупности данных, которые характеризуют состав объектов предметной области, их свойства и взаимосвязи. Такое отражение предметной области принято называть базой данных (БД). Более строгое определение звучит следующим образом.

База данных – это именованная совокупность данных, отражающая состояние объектов и их взаимосвязей (отношений) в рассматриваемой предметной области.

Жизненный цикл системы обработки данных состоит из нескольких этапов:

1) проектирование базы данных;
2) проектирование приложений;
3) реализация базы данных, включающая начальную загрузку и запуск в эксплуатацию;
4) разработка специальных средств администрирования базы данных;
5) эксплуатация базы данных (анализ функционирования, модификация и адаптация).

Процесс проектирования базы данных представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию в терминах некоторой модели и в свою очередь включает несколько этапов.

1. *Системный анализ и словесное описание информационных объектов* предметной области. Основной задачей этого этапа является сбор требований, предъявляемых всеми пользователями к содержимому базы данных и процессу обработки информации. Объединяя частные представления о содержимом БД, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, проектировщики создают обобщенное неформальное описание БД. Оно выполняется с использованием естественного

языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающим над проектированием базы данных.

2. *Проектирование инфологической модели*, представляющей собой частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например ER-модели. В результате выполнения этого этапа строится концептуальная (СУБД-независимая) модель предметной области, которая в дальнейшем будет конкретизирована. Инфологическая модель полностью независима от физических параметров среды хранения данных.

3. *Логическое проектирование базы данных*, т. е. описание базы данных в терминах принятой логической модели данных. На этом этапе выполняется выбор типа СУБД, строится схема базы данных в терминах соответствующей модели данных, подсхемы для различных пользователей, создается набор возможных типовых запросов и спецификации для программного обеспечения.

4. *Физическое проектирование базы данных* заключается в выборе эффективного размещения базы данных на внешних носителях для обеспечения наиболее эффективной работы приложения. Определяется используемая СУБД. Создается реальная БД, программируются и отлаживаются приложения, которые будут работать с БД.

Результатом этапа проектирования являются готовая к заполнению реальной информацией база данных и готовое к работе программное обеспечение.

Современные базы данных, как правило, имеют *трехуровневую архитектуру*, которая образуется в процессе проектирования БД. ***Трехуровневая архитектура***, т. е. инфологический, логический и физический уровни, позволяет обеспечить логическую и физическую независимость хранимых данных от использующих их программ. При необходимости можно переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь физическую модель данных. Можно подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, логическую модель. Указанные изменения физической и логической моделей не будут замечены пользователями системы (окажутся «прозрачными» для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

1.3. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

СУБД – это программный комплекс, обеспечивающий функционирование базы данных. Он отвечает за сохранность, безопасность, целостность, взаимное соответствие данных и обеспечивает доступ пользователей к данным. Дадим строгое определение.

Система управления базами данных представляет собой совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Первые СУБД появились в конце 1960–1970-х гг. Они ориентировались на мэйнфреймы и поддерживали иерархическую и сетевую модели. В процессе дальнейшего развития СУБД постоянно совершенствовались – возникали новые подходы к хранению и обработке данных, организации процесса разработки баз данных и приложений, разрабатывались новые модели данных. Более подробно структура и функции СУБД будут рассмотрены ниже.

2. КЛАССИФИКАЦИЯ МОДЕЛЕЙ ДАННЫХ

2.1. МОДЕЛИРОВАНИЕ ДАННЫХ

Модель данных – это некоторая абстракция, которая, будучи приложима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, т. е. как сведения, содержащие не только данные, но и взаимосвязи между ними.

Возможны следующие связи между объектами предметной области и соответственно описывающими их данными: «один к одному»; «один ко многим»; «многие к одному»; «многие ко многим». Связь «один к одному» (1 : 1) означает, что каждому экземпляру объекта *A* может соответствовать только один экземпляр объекта *B* и наоборот. Связь «один ко многим» (1 : *M*) означает, что могут существовать экземпляры объекта *A*, которым соответствует более одного экземпляра объекта *B*, но каждому экземпляру объекта *B* может соответствовать только один экземпляр объекта *A*. Связь «многие к одному» (*M* : 1) имеет место, когда каждому экземпляру объекта *A* ставится в соответствие ровно один экземпляр объекта *B*, но экземпляру объекта *B* могут соответствовать более одного экземпляра объекта *A*. И наконец, связь «многие ко многим» (*M* : *N*) означает, что нескольким экземплярам объекта *A* могут соответствовать несколько экземпляров объекта *B*.

Представление информации в базе данных осуществляется в рамках определенных ограничений, обусловленных используемой информаци-

онной системой, ресурсами, выбранной логической и физической структурами организации данных. Прежде всего, эти ограничения определяют допустимые типы данных и допустимые связи между данными. Ограничения касаются и операций, которые могут выполняться над данными и связями. Существует и множество ограничений, обусловливающих *целостность* базы данных. ***Целостность*** базы данных означает, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация, т. е. отдельные фрагменты данных взаимно согласованны и корректны. *Согласованность* означает, что все порции данных должны быть единообразно смоделированы и включены в систему. *Корректность* – что они достоверны, точны и значимы. Множество допустимых типов данных и связей между ними, множество допустимых операций над данными и связями, множество ограничений целостности в совокупности определяют используемую ***модель данных***. Рассмотрим существующие модели данных.

2.2. ИЕРАРХИЧЕСКАЯ МОДЕЛЬ

Основополагающей логической структурой для ***иерархической модели*** является *ориентированное дерево* с корнем. *Вершины* дерева соответствуют интересующим нас объектам, а *дуги* – связям между объектами. Все вершины дерева, за исключением корня, должны иметь предка. Между двумя вершинами может быть только одна связь. Связи вершины с непосредственно подчиненными вершинами должны иметь определенное упорядочение, как правило, слева направо. Основными типами данных являются два: *запись* и *дерево*. Дерево состоит из одной корневой записи и упорядоченного набора из нуля или более подчиненных записей, каждая из которых в свою очередь может иметь нуль или более подчиненных записей. Каждая вершина дерева может быть представлена в виде некоторой записи или упорядоченного набора записей, а каждая дуга – встроенным в запись указателем (адресом). Иерархическая база данных представляет собой упорядоченную совокупность экземпляров данных типа «дерево».

Основными операциями манипулирования данными в иерархической модели являются: поиск указанного экземпляра дерева; переход от одного дерева к другому; переход от одной записи к другой внутри дерева; вставка новой записи в указанную позицию; удаление текущей записи и т. д. Используются два метода доступа к записям внутри дерева. Прямой порядок обхода начинается с корня с последующей обработкой всего дерева в порядке слева направо. Обратный порядок обхода начинается с

левой висячей вершины с постепенным переходом от одного поддерева к другому слева направо с завершением обработки в корне.

Иерархическая модель поддерживает связи «один к одному» и «один ко многим». Возможна организация связи «многие ко многим» за счет дублирования данных. Основное ограничение целостности заключается в том, что потомок не может существовать без родителя. Поэтому при удалении родительской записи удаляется все определяемое ею поддерево.

К достоинствам иерархической модели относятся эффективное использование памяти ЭВМ, неплохие показатели времени выполнения основных операций над данными, удобство работы с иерархически упорядоченной информацией.

Недостатками иерархической модели являются невозможность хранения экземпляров, не имеющих родительских записей, трудность реализации связей «многие ко многим» и других более сложных иерархических связей.

2.3. СЕТЕВАЯ МОДЕЛЬ

Сетевой подход к организации данных является расширением иерархического. В иерархической модели запись-потомок должна иметь в точности одного предка; в сетевой модели запись-потомок может иметь любое число предков. Для реализации иерархической структуры используются две группы типов данных: *записи* и *набор*. Набор устанавливает именованную связь для записи-предка и одной или нескольких записей-потомков, т. е. поддерживает связи «один к одному» и «один ко многим». Для организации связи «многие ко многим» образуются две связи «один ко многим», объединенные в единую связующую запись. При этом должны выполняться следующие ограничения:

1) только одна запись может быть предком в каждом наборе, но одна и та же запись может быть предком в нескольких различных наборах;

2) одна или больше записей могут быть членами одного и того же набора;

3) запись может входить в несколько наборов;

4) запись может быть предком в одних наборах и потомком в других наборах;

5) между любыми двумя записями может быть определено любое количество наборов;

6) наборы могут быть определены так, что в результате они образуют циклическую структуру;

7) запись необязательно должна быть членом двух экземпляров одного и того же типа набора;

8) запись необязательно должна быть членом какого-либо набора.

Основными операциями при манипулировании данными в сетевой модели являются: поиск записи; создание новой записи; удаление записи; модификация записи; переход от предка к первому потомку; переход от потомка к следующему потомку; переход от потомка к предку; включение записи в набор; исключение записи из набора; перестановка записи в другой набор.

Доступ к типам записи осуществляется путем «перемещения» по структуре и зависит от метода реализации наборов – с помощью цепочек указателей или массивов указателей. Целостность в сетевой модели поддерживается с помощью наборов. Если записи включены в набор, то удаление записи-предка набора приводит к удалению всего набора и каскадному удалению последующих наборов. Если записи-потомки не входят в набор, удаление записи-предка эквивалентно удалению связи.

К достоинствам сетевой модели относится возможность установления произвольных связей между записями. Недостатком сетевой модели является высокая сложность схемы базы данных.

Сложность сетевых и иерархических моделей объясняется тем, что они построены с использованием внутренних физических указателей, связывающих записи между собой.

2.4. РЕЛЯЦИОННАЯ МОДЕЛЬ

Реляционная модель была разработана доктором Э. Ф. Коддом в начале 1970-х гг. С ее созданием начался новый этап в эволюции СУБД. Простота и гибкость модели привлекли к ней внимание разработчиков и снискали ей множество сторонников. Несмотря на некоторые недостатки, реляционная модель стала доминирующей, а реляционные СУБД стали промышленным стандартом де-факто. ***Реляционная модель*** основана на математическом понятии отношения, физическим представлением которого является двухмерная таблица, состоящая из строк одинаковой структуры. Логическая структура данных представляется набором связанных таблиц. Модель поддерживает связи «один к одному» и «один ко многим». Связь «многие ко многим» реализуется с помощью декомпозиции.

Рассмотрим более детально реляционную модель.

Как уже отмечалось, любая база данных состоит из описаний объектов некоторой предметной области, а также содержит информацию о взаимосвязях между объектами. Тип объекта называется ***сущностью***, а характеристики объектов – ***атрибутами***. Таким образом, сущности соответствует определенный набор атрибутов, а каждому конкретному

объекту соответствует набор значений атрибутов. Набор атрибутов, однозначно определяющий каждый объект, называют ***ключом***. Атрибут можно рассматривать как переменную, принимающую значения из некоторого множества значений, называемого ***доменом*** атрибута.

Рассмотрим объект типа T, имеющий набор атрибутов A_1, A_2, ..., A_n. Атрибут A_j может принимать значения из области (домена) D_j, $j = 1, 2, ..., n$. Обозначим через a_{ij} значение атрибута A_j для объекта i, тогда каждому конкретному объекту i типа T соответствует кортеж вида

$$a_i = (a_{i1}, a_{i2}, ..., a_{in}), a_{ij} \in D_j, i = 1, 2, ..., m; j = 1, 2, ..., n;$$

где m – количество объектов типа T. Всему набору рассматриваемых объектов типа T соответствует набор кортежей:

$$\underline{R} = \begin{matrix} a_{11}, a_{12}, ..., a_{1n} \\ a_{21}, a_{22}, ..., a_{2n} \\ ..... \\ a_{m1}, a_{m2}, ..., a_{mn} \end{matrix}$$

Ясно, что $\underline{R} \subseteq D_1 \times D_2 \times ... \times D_n$.

Множество кортежей $\underline{R}$ называют ***отношением***, а количество атрибутов n – ***арностью*** отношения. Количество содержащихся в отношении кортежей называется ***кардинальностью*** отношения. Заметим, что так как отношение – это множество, то порядок следования кортежей в отношении несущественен; отношение не содержит одинаковых элементов – кортежей и, следовательно, обязательно имеет набор атрибутов, являющийся ключом.

Совокупность атрибутов $R = (A_1, A_2, ..., A_n)$ называется ***схемой отношения***. Если обозначить $U = \{A_1, A_2, ..., A_n\}$, то схему отношения можно записать в виде $R = (U)$. Само отношение $\underline{R}$ называется ***текущим значением*** или ***экземпляром*** схемы отношения R. База данных обычно содержит несколько отношений: $\underline{R}_1$, $\underline{R}_2$, …, $\underline{R}_k$; совокупность их схем $R_1 = (U_1)$, $R_2 = (U_2)$, ..., $R_k = (U_k)$ называется ***схемой реляционной базы данных.***

Отношение можно рассматривать как двухмерную таблицу, каждый столбец которой имеет имя – атрибут, а каждая строка содержит данные по одному объекту или данные о связи между несколькими конкретными объектами.

Таким образом, набор кортежей $\underline{R}$ можно записать в виде:

A_1	A_2	…	A_n
a_{11}	a_{12}	…	a_{1n}
a_{21}	a_{22}	…	a_{2n}
…	…	…	…

a_{m1}	a_{m2}	…	a_{mn}

Для таблицы должны выполняться следующие правила: таблица имеет имя, отличное от имен других таблиц; каждая клетка содержит только атомарное значение; каждый столбец имеет уникальное имя; данные для столбца берутся из одного множества значений; порядок следования столбцов не имеет значения; таблица не имеет повторяющихся строк; строки не имеют имен; порядок следования строк не имеет значения.

Любая таблица имеет один или несколько столбцов, значения которых однозначно идентифицируют каждую ее строку. Такой столбец (или совокупность столбцов) называется ***первичным ключом***. Взаимосвязи таблиц в реляционной модели поддерживаются внешними ключами. ***Внешний ключ*** – это столбец (или совокупность столбцов), значения которого однозначно характеризуют сущности, представленные строками некоторого другого отношения, т. е. задают значения их первичного ключа.

Для пользователей АИС необходимо, чтобы база данных отражала предметную область однозначно и непротиворечиво, т. е. удовлетворяла условиям целостности. Для обеспечения выполнения условий целостности на базу данных накладываются некоторые ограничения, которые называют ***ограничениями целостности***. Выделяют два основных типа таких ограничений: ***целостность сущностей*** и ***целостность ссылок***. Ограничение первого типа означает, что любое отношение должно обладать первичным ключом (в принципе для отношения это свойство должно выполняться автоматически). Ограничение ссылочной целостности заключается в том, что внешний ключ не может быть указателем на несуществующую строку в таблице. Контроль целостности осуществляется проверкой ограничений целостности:

1) ключевой столбец не может содержать неопределенное значение (определитель NULL);

2) для связанных таблиц каждой строке основной таблицы соответствует нуль или более строк подчиненной таблицы;

3) для связанных таблиц в подчиненной таблице нет строк, не имеющих родительских строк в основной таблице;

4) для связанных таблиц каждая строка подчиненной таблицы имеет только одну родительскую строку в основной таблице.

Основными операциями манипулирования данных являются: добавление строк; модификация строк; удаление строк.

В основе операций над отношениями лежат операции реляционной алгебры и реляционного исчисления, о которых будет рассказано в соответствующем разделе.

Достоинствами реляционной модели являются простота, наглядность, независимость от данных. К тому же, в отличие от сетевых и иерархических моделей, реляционные модели для организации связей между записями применяют не внутренние указатели, а фактические значения атрибута, используя общий атрибут в каждой из записей.

Недостатки реляционной модели связаны с однородностью структуры данных, семантической перегруженностью модели, ограниченным набором операций.

2.5. ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ МОДЕЛЬ

Объектно-ориентированная модель данных учитывает семантику объектов, применяемую в объектно-ориентированном программировании. Основными модельными понятиями являются объекты и литералы. *Объект* обладает уникальным идентификатором, который не изменяется и не используется после удаления объекта. Объекты могут быть разбиты на типы: атомарные, коллекции или структурированные типы. Тип также является объектом. Объект инкапсулирует состояние и поведение. Поведение объекта – это операции, которые могут быть выполнены либо самим объектом, либо над ним. В совокупности эти операции называются методами. Состояние объекта определяется значениями, которые имеются у набора свойств объекта. Имеются два типа свойств – атрибуты и связи. Атрибут определяется для объектов одного типа. Он не является объектом, но может принимать в качестве значений литерал или идентификатор объекта. Объект может хранить все связи, которыми он связан с другими объектами, включая связь «многие ко многим». Связи представлены с помощью ссылочных атрибутов. Запрос одного объекта к другому называют сообщением. Объекты, имеющие одинаковые атрибуты и отвечающие на одни и те же сообщения, образуют класс. Наследование позволяет определить один класс как частный случай более общего класса. Полиморфизм означает допустимость в объектах разных типов иметь методы с одинаковыми именами.

Типы *литералов* можно разбить на атомарные, коллекции, структурированные типы и объекты без типа. Литералы не могут существовать отдельно. Они всегда встроены в объект. С помощью механизма наследования допускается создание новых абстрактных типов данных на основе уже существующих.

Логически структура объектно-ориентированной базы данных похожа на структуру иерархической базы данных. Основное отличие состоит в методах манипулирования данными.

Достоинствами объектно-ориентированной модели являются улучшенные возможности моделирования объектов реального мира. Объектные типы данных, а также объектные таблицы представляют мощный единый уровень интерпретации объектов деловой сферы и позволяют отказаться от деления на части бизнес-данных для хранения их в БД при использовании реляционной модели.

Недостатками модели является высокая понятийная сложность, отсутствие стандарта объектно-ориентированной модели из-за недостаточной ее теоретической разработки.

2.6. ОБЪЕКТНО-РЕЛЯЦИОННАЯ МОДЕЛЬ

В связи с неразработанностью объектно-ориентированной модели на практике применяется ***объектно-реляционная модель***, являющаяся как бы смесью реляционной и объектно-ориентированной методологий для представления данных. Эта модель представляет собой расширенную реляционную модель, в которой сняты ограничения неделимости данных, хранящихся в записях таблиц. Допускаются многозначные поля – поля, значениями которых являются самостоятельные таблицы, встроенные в основную таблицу. Кроме этого поддерживаются такие концепции объектно-ориентированного программирования, как «абстракция», «класс», «экземпляр», «инкапсуляция», «метод», «перегрузка» и «сообщение». Хотя наследование и является одной из наиболее важных характеристик объектов, но в объектно-реляционной модели оно не поддерживается.

В модели вводится специальный *объектный тип*, с помощью которого можно создать абстрактный тип данных любой степени сложности. Используя вложенные объектные типы, можно создавать структуры, в которых используются все виды связей: «один к одному», «один ко многим» и даже «многие ко многим». Хотя это и может привести к определенной избыточности, такой подход дает преимущества по сравнению с использованием множества нормализованных таблиц в чисто реляционной модели.

Преимуществом объектно-реляционной модели является возможность использования существующих реляционных баз данных с вновь разрабатываемыми объектными приложениями.

К недостаткам модели можно отнести сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.

2.7. МНОГОМЕРНАЯ МОДЕЛЬ

Многомерная модель данных является узкоспециализированной моделью, предназначенной для оперативной аналитической обработки информации. В основе модели лежит не двухмерная, как в реляционной модели, а многомерная таблица и многомерное логическое представление структуры информации при описании данных и в операциях манипулирования данными. По сравнению с реляционной моделью многомерная организация данных обладает более высокой наглядностью и информативностью. Модели должны быть присущи агрегируемость, историчность и прогнозируемость данных.

Агрегируемость данных означает рассмотрение информации на различных уровнях ее обобщения.

Историчность данных предполагает обеспечение высокого уровня статичности данных и их взаимосвязей, а также обязательную привязку данных ко времени.

Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным временным интервалам.

К числу основных понятий многомерной модели относятся измерение и ячейка. *Измерение* – это множество однотипных данных, образующих одну из граней многомерной таблицы. *Ячейка* – это поле, значение которого однозначно определяется фиксированным набором измерений.

Используются два основных варианта организации данных: гиперкубическая и поликубическая.

В *гиперкубической* схеме предполагается, что все многомерные таблицы имеют одинаковую размерность и совпадающие измерения.

В *поликубической* схеме может быть определено несколько таблиц с различной размерностью и различными измерениями.

В многомерной модели реализуются такие специальные операции, как формирование «среза», вращение, агрегация и детализация.

Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем.

Недостатком многомерной модели является сложность ее структуры при реализации простейших задач обычной оперативной обработки данных.

3. РЕЛЯЦИОННАЯ АЛГЕБРА И РЕЛЯЦИОННОЕ ИСЧИСЛЕНИЕ

3.1. РЕЛЯЦИОННАЯ АЛГЕБРА

Реляционная алгебра – это теоретический язык операций, которые на

основе одного или нескольких отношений позволяют создавать другое отношение. Реляционную алгебру можно рассматривать как процедурный язык, указывающий, как следует строить требуемое отношение на базе одного или нескольких исходных отношений. Операндами реляционной алгебры являются отношения фиксированной арности. Операции, применяемые к одному отношению, называются *унарными*, применяемые к паре отношений – *бинарными*. Отношения, участвующие в выполнении бинарных операций, в ряде случаев должны быть совместимы по структуре, что означает совместимость имен атрибутов и типов соответствующих доменов.

Отношения реляционной алгебры – это множества, поэтому средства работы с отношениями базируются на традиционных операциях теории множеств, которые дополняются некоторыми специальными операциями, специфичными для баз данных.

К основным операциям относятся следующие: объединение отношений; разность отношений; пересечение отношений; декартово произведение отношений; проекция; выборка; деление отношений; θ-соединение отношений; естественное соединение; внешнее соединение отношений; полусоединение отношений.

Рассмотрим эти операции подробнее.

Объединение. Эта операция почти полностью соответствует операции объединения в теории множеств. Объединение отношений $\underline{R}$ и $\underline{S}$, обозначаемое как $\underline{R} \cup \underline{S}$, представляет собой множество всех таких кортежей, каждый из которых принадлежит $\underline{R}$ или $\underline{S}$, или обоим сразу. Операция объединения применяется только к совместимым отношениям одной арности, поэтому все кортежи в объединении имеют одинаковое число компонентов.

Разность. Разностью отношений $\underline{R}$ и $\underline{S}$, обозначаемой $\underline{R} \setminus \underline{S}$, называется множество всех кортежей, принадлежащих $\underline{R}$, но не принадлежащих $\underline{S}$. Здесь также требуется, чтобы $\underline{R}$ и $\underline{S}$ имели одну и ту же арность и были совместимы.

Пересечение. $\underline{R} \cap \underline{S}$ обозначает множество всех кортежей, принадлежащих одновременно $\underline{R}$ и $\underline{S}$. $\underline{R}$ и $\underline{S}$ должны иметь одинаковую арность и быть совместимы. Очевидно, что $\underline{R} \cap \underline{S} = \underline{R} \setminus (\underline{R} \setminus \underline{S}) = \underline{S} \setminus (\underline{S} \setminus \underline{R})$.

П р и м е р

Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результаты выполнения операций «объединение», «разность» и «пересечение» над этими отношениями можно также представить в виде таблиц.

A	*B*	*C*

A	*B*	*C*

$\underline{R}$ =

1	2	3
4	1	6
3	2	4

$\underline{S}$ =

2	7	1
4	1	6

$\underline{R} \cup \underline{S}$ =

A	*B*	*C*
1	2	3
4	1	6
3	2	4
2	7	1

$\underline{R} \setminus \underline{S}$ =

A	*B*	*C*
1	2	3
3	2	4

$\underline{R} \cap \underline{S}$ =

A	*B*	*C*
4	1	6

Декартово произведение. Пусть $\underline{R}$ и $\underline{S}$ – отношения арности r и s соответственно. Тогда декартовым произведением $\underline{R} \times \underline{S}$ отношений $\underline{R}$ и $\underline{S}$ называется множество всех кортежей длины $r + s$ таких, что первые r компонент образуют кортежи, принадлежащие $\underline{R}$, а последние s компонент – кортежи, принадлежащие $\underline{S}$.

П р и м е р

Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результат выполнения этой операции над этими отношениями можно также представить в виде таблицы.

$\underline{R}$ =

A	*B*	*C*
1	2	3
4	1	6
3	2	4

$\underline{S}$ =

D	*E*	*F*
2	7	1
4	1	6

$\underline{R} \times \underline{S}$ =

A	*B*	*C*	*D*	*E*	*F*
1	2	3	2	7	1
1	2	3	4	1	6
4	1	6	2	7	1
4	1	6	4	1	6
3	2	4	2	7	1
3	2	4	4	1	6

Если имена столбцов в отношениях-сомножителях совпадают, то их помечают именами отношений, например вместо *A*, *B*, *C*, *D*, *E*, *F* можно писать $\underline{R}.A$, $\underline{R}.B$, $\underline{R}.C$, $\underline{S}.D$, $\underline{S}.E$, $\underline{S}.F$.

Проекция. Сущность этой операции заключается в том, что в исход-

ном отношении удаляются некоторые компоненты (атрибуты) и (или) переставляются оставшиеся. Пусть $\underline{R}$ – отношение арности r. Обозначим $\pi_{i_1, i_2, \ldots, i_m}(\underline{R})$, где i_j – целые числа в диапазоне от 1 до r, проекцию $\underline{R}$ на компоненты i_1, i_2, …, i_m, т. е. множество таких кортежей a длины m, что существует некоторый принадлежащий $\underline{R}$ кортеж b_1, b_2, …, b_r, удовлетворяющий условию $a_i = b_{i_j}$.

Например, для построения $\pi_{3,1}(\underline{R})$ нужно из каждого кортежа, принадлежащего $\underline{R}$, сформировать кортеж длины 2 из третьего и первого его компонентов в указанном порядке, т. е. выписать 3-й, затем 1-й компоненты. При этом из каждой группы одинаковых кортежей в результирующем отношении оставляется только один кортеж (отношение – это множество кортежей, и оно не может содержать одинаковых, т. е. совпадающих по всем компонентам кортежей).

Вместо номеров компонент (столбцов) часто используют атрибуты.

Выборка. Пусть F – формула, образованная:

1) операндами, являющимися константами или номерами компонент (атрибутами);

2) операциями сравнения: <, =, >, ≤, ≥;

3) логическими операциями Λ (И – конъюнкция), ∨ (ИЛИ – дизъюнкция), ¬ (НЕТ – отрицание).

В этом случае $\sigma_F(\underline{R})$ есть множество всех таких кортежей t, принадлежащих $\underline{R}$, что при подстановке i-го компонента вместо всех вхождений i (или соответствующего атрибута) в формулу F она станет истинной.

Например, $\sigma_{2>3}$ обозначает множество кортежей, принадлежащих $\underline{R}$, второй компонент которых больше третьего.

Заметим, что константы в формулах должны быть заключены в кавычки, это позволит отличить их от номеров или имен столбцов.

П р и м е р

Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результаты выполнения операций проекции и выборки над этими отношениями можно также представить в виде таблиц.

$\underline{R}$ =

A	*B*	*C*
1	2	3
4	1	6
3	2	4

$\pi_{A,C}(\underline{R})$ =

A	*C*
1	3

$\sigma_{B='2'}(\underline{R})$ =

A	*B*	*C*
1	2	3

4	6
3	4

3	2	4

Деление. Пусть $\underline{R}$ и $\underline{S}$ – отношения арности r и s соответственно, причем $r > s$ и $\underline{S} \neq \varnothing$. Предположим, что $\underline{R}$ определено на множестве атрибутов A, а $\underline{S}$ на множестве атрибутов B и $B \subseteq A$. $\underline{R} \div \underline{S}$ (частное) есть множество кортежей t длины $r - s$, таких, что для всех кортежей $u \in \underline{S}$, кортеж tu принадлежит $\underline{R}$ (здесь tu означает кортеж, полученный приписыванием справа к кортежу t компонентов кортежа u).

П р и м е р

Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результат выполнения операции деления над этими отношениями можно также представить в виде таблицы.

$\underline{R} =$

A	*B*	*C*	*D*
1	2	3	4
1	2	5	6
2	3	5	6
5	4	3	4
5	4	5	6
1	2	4	5

$\underline{S} =$

C	*D*
3	4
5	6

$\underline{R} \div \underline{S} =$

A	*B*
1	2
5	4

θ-соединение. θ-соединение отношений $\underline{R}$ и $\underline{S}$ по столбцам i и j, записываемое $\underline{R}|><|_{i\theta j}\underline{S}$, где θ – операция сравнения (=, < и т. д.), есть краткая запись выражения: $\sigma_{i\theta j}(\underline{R} \times \underline{S})$. Таким образом, θ-соединение $\underline{R}$ и $\underline{S}$ представляет собой множество всех таких кортежей в декартовом произведении $\underline{R}$ и $\underline{S}$, что i-й атрибут $\underline{R}$ находится в отношении θ с j-м атрибутом отношения $\underline{S}$. Если θ является операцией «=», то эта операция называется эквисоединением.

Естественное соединение. Эта операция играет фундаментальную роль в теории проектирования реляционных баз данных. Естественное соединение (обозначается $\underline{R}\,|><|\,\underline{S}$) применимо лишь тогда, когда отношения имеют один или несколько общих атрибутов.

Вычислить $\underline{R}\,|><|\,\underline{S}$ можно следующим образом:

1) вычислить $\underline{R} \times \underline{S}$;

2) для каждого атрибута A, который именует некоторый столбец в $\underline{R}$ и какой-либо столбец в $\underline{S}$, выберем только те кортежи из $\underline{R} \times \underline{S}$, у которых совпадают значения в столбцах $\underline{R}.A$ и $\underline{S}.A$;

3) для каждого указанного выше атрибута A удалим столбец $\underline{R}.A$.

Формально, если $A_1, A_2, \ldots, A_k$ являются общими атрибутами для $\underline{R}$ и $\underline{S}$,

то

$$\underline{R} \mid >< \mid \underline{S} = \pi_{i_1, i_2, \ldots, i_m} (\sigma_{\mathrm{R.A}_1 = S.A_1 \wedge \ldots \wedge R.A_k = S.A_k} (\underline{R} \times \underline{S})),$$

где i_1, i_2, …, i_m − упорядоченный список всех атрибутов $\underline{R} \times \underline{S}$, за исключением атрибутов $\underline{S}.A_1$, $\underline{S}.A_2$, …, $\underline{S}.A_k$.

Рассмотрим ряд примеров, реализующих вышеописанные операции.

П р и м е ры

1. Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результат выполнения операции θ-соединения над этими отношениями можно также представить в виде таблицы.

$\underline{R} =$

A	*B*	*C*
1	2	3
4	5	6
7	8	9

$\underline{S} =$

D	*E*
3	1
6	2

$\underline{R} \mid >< \mid_{B<D} \underline{S} =$

A	*B*	*C*	*D*	*E*
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

2. Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результат выполнения операции естественного соединения над этими отношениями можно также представить в виде таблицы.

$\underline{R} =$

A	*B*	*C*
1	2	3
4	2	3
2	2	6
3	1	4

$\underline{S} =$

B	*C*	*D*
2	3	4
2	3	5
1	4	2

$\underline{R} \mid >< \mid \underline{S} =$

A	*B*	*C*	*D*
1	2	3	4
1	2	3	5
4	2	3	4
4	2	3	5
3	1	4	2

Внешнее соединение. ***Левым внешним соединением*** $\underline{R} \supset \mid \underline{S}$ отношений $\underline{R}$ и $\underline{S}$ называется отношение, содержащее все кортежи отношения

$\underline{R}|><|\underline{S}$, а также кортежи $\underline{R}$, не имеющие совпадающих значений в общих столбцах с отношением $\underline{S}$. Для обозначения отсутствующих значений во втором отношении используется определитель NULL.

Аналогично вводится операция ***правого внешнего соединения*** $\underline{R}|>\subset \underline{S}$. Существует также ***полное внешнее соединение*** $\underline{R} \supset\subset \underline{S}$, в результирующем отношении которого помещаются все кортежи из обоих отношений и в котором для обозначения несовпадающих значений кортежей используются определители NULL.

Полусоединение. Эту операцию можно определить с помощью операций «проекция» и «θ-соединение». А именно $\underline{R}|>_{i\theta j}\underline{S} = \pi_u(\underline{R}|><|_{i\theta j}\underline{S})$. Здесь u – это набор всех атрибутов отношения $\underline{R}$. Аналогично определяются полусоединение по эквивалентности (когда θ есть равенство) и естественное полусоединение $\underline{R}|> \underline{S}$.

П р и м е р ы

1. Пусть отношения $\underline{R}$ и $\underline{S}$ представлены следующими таблицами, тогда результаты выполнения операций левого внешнего, правого внешнего и полного внешнего соединения над этими отношениями можно также представить в виде таблиц.

$\underline{R} =$

A	*B*	*C*
1	2	3
4	2	3
2	2	6
3	1	4

$\underline{S} =$

B	*C*	*D*
2	3	4
2	3	5
1	4	2
3	5	4

$\underline{R} \supset|< \underline{S} =$

A	*B*	*C*	*D*
1	2	3	4
1	2	3	5
4	2	3	4
4	2	3	5
3	1	4	2
2	2	6	NULL

$\underline{R}|>\subset \underline{S} =$

A	*B*	*C*	*D*
1	2	3	4
1	2	3	5
4	2	3	4
4	2	3	5
3	1	4	2
NULL	3	5	4

$\underline{R} \supset\subset \underline{S} =$

A	*B*	*C*	*D*
1	2	3	4
1	2	3	5
4	2	3	4

4	2	3	5
3	1	4	2
2	2	6	NULL
NULL	3	5	4

2. Пример, демонстрирующий выполнение θ-полусоединения отношений $\underline{R}$ и $\underline{S}$.

$\underline{R} =$

A	*B*	*C*
1	2	3
4	3	2
2	7	1
1	2	6

$\underline{S} =$

D	*E*
4	5
3	2

$\underline{R} \mid >_{C<D} \underline{S} =$

A	*B*	*C*
1	2	3
4	3	2
2	7	1

3. Пример, демонстрирующий выполнение естественного полусоединения отношений $\underline{R}$ и $\underline{S}$.

$\underline{R} =$

A	*B*	*C*
7	3	1
2	4	1

$\underline{S} =$

B	*C*	*D*
4	3	1
4	1	2

$\underline{R} \mid > \underline{S} =$

A	*B*	*C*
2	4	1

3.2. РЕЛЯЦИОННОЕ ИСЧИСЛЕНИЕ.

В отличие от реляционной алгебры, которая указывает, как получить требуемое отношение из имеющихся отношений, ***реляционное исчисление*** указывает свойства искомого отношения без конкретизации процедуры его получения. Математической основой реляционного исчисления является исчисление предикатов – один из разделов математической логики. В теории исчисления предикатов под *предикатом* понимается функция, значения которой являются высказываниями. Высказывание может быть истинным или ложным. Чтобы задать *n*-местный предикат $P(x_1, \ldots, x_n)$, следует указать множества $D_1, \ldots, D_n$ – области изменения предикатных переменных $x_1, \ldots, x_n$. Синтаксически задание *n*-местного

предиката осуществляется указанием формулы логико-математического языка, содержащей n переменных. В наиболее распространенном случае такой язык содержит предикатные переменные x, y, z, … функциональные символы f, g, h, … с различным количеством аргументных мест и предикатные символы P, Q, R, … также с различным количеством аргументных мест. Из переменных и функциональных символов конструируются термы языка, содержательно интерпретируемые как имена объектов исследования. Если P есть n-местный предикатный символ, $n \geq 0$, а t_1, …, t_n – термы, то $P(t_1, \ldots, t_n)$ есть, по определению, атомарная формула. Содержательно $P(t_1, \ldots, t_n)$ означает, что истинно высказывание, гласящее, что t_1, …, t_n связаны отношением P. Из атомарных формул с помощью пропозициональных связок и кванторов конструируются формулы языка. Обычный набор связок состоит из операторов сравнения, а набор кванторов включает конъюнкцию, дизъюнкцию, импликацию, отрицание, квантор «для всех», квантор «существует». Вхождения переменной x в формулу φ называется *связанным*, если x входит в часть φ вида $\exists x\varphi$ или $\forall x\varphi$. Остальные вхождения называются *свободными*.

Для баз данных исчисление предикатов существует в двух формах: реляционного исчисления кортежей и реляционного исчисления доменов. В *реляционном исчислении кортежей* задача состоит в нахождении таких кортежей, для которых предикат является истинным. Это исчисление основано на переменных кортежа, т. е. таких, для которых допустимыми значениями могут быть только кортежи данного отношения. Описательную часть исчисления можно представить в виде

RANGE OF <переменная> IS <список>.

Здесь <переменная> – это идентификатор переменной кортежа, <список> – конструкция вида $X_1[, X_2[, \ldots, X_n]\ldots]$. Список содержит элементы, каждый из которых является либо отношением, либо выражением над отношениями. Область допустимых значений <переменной> образуется путем объединения значений всех элементов списка. Выражение реляционного исчисления, формирующего запрос на языке исчисления кортежей, упрощенно можно записать в виде

$(Y_1[, Y_2[\ldots,Y_m]\ldots])$ [WHERE wff].

Здесь Y_i – это запись вида

{<переменная> |<переменная>.<атрибут>} [AS <атрибут>]

Соответственно wff – well-formed formula (правильно построенная формула) – это предикат, который записывается одним из следующих типов:

<условие>

NOT wff
<условие> AND wff
<условие> OR wff
IF <условие> THEN wff
EXISTS <переменная> (wff)
FORALL <переменная> (wff)
(wff)

Например, предположим, что имеется три отношения О1, О2, О3,

ПР	КАФ	ФАК
П1	К1	Ф1
П2	К1	Ф1
П3	К2	Ф1

отношение О1

СТУД	КУРС	ГР
С1	3	2
С2	3	1
С3	4	2
С4	4	1

отношение О2

ПР	СТУД	ЧАС
П1	С1	10
П2	С2	10
П2	С3	10
П3	С1	10
П3	С4	20

отношение О3

в первом из которых указываются сведения о преподавателях (преподаватель, кафедра, факультет), во втором – сведения о студентах (студент, курс, группа), в третьем – сведения о количестве часов консультаций (преподаватель, студент, часы). Создадим список преподавателей, работающих на кафедре К1:

RANGE OF S IS O1
(S.ПР) WHERE S.КАФ=’К1’

Создадим список преподавателей, студентов четвертого курса и часов:

RANGE OF P IS O2
RANGE OF V IS O3
(V) WHERE EXISTS P (V.СТУД=P.СТУД AND P.КУРС=4)

Этот же список можно сформировать следующим образом:

RANGE OF P IS O2
RANGE OF V IS O3
RANGE OF VP IS (V) WHERE EXISTS P (V.СТУД=P.СТУД AND P.КУРС=4)
(VP)

В *реляционном исчислении доменов* используются переменные, значения которых берутся из доменов, а не из кортежей отношений. Исчисление доменов поддерживает дополнительную форму условий, называе-

мую условием принадлежности. В общем виде условие принадлежности записывается в виде $\underline{R}(A_1 : p_1, A_2 : p_2,\ldots)$, где A_i – атрибут отношения $\underline{R}$, а p_i – переменная домена или литерал. Условие считается истинным тогда и только тогда, когда в отношении $\underline{R}$ имеется кортеж со значениями p_i для атрибутов A_i.

Для приведенных выше в примере первого списка выражения исчисления доменов будут иметь вид

(S) WHERE O1 (КАФ : ‘К1’)

Здесь S – переменная домена атрибута ПР, объявленная каким-либо образом, подобно оператору RANGE исчисления кортежей.

В заключение заметим, что если реляционное исчисление ограничивается только безопасными (т. е. имеющими смысл) выражениями, то реляционное исчисление доменов эквивалентно реляционному исчислению кортежей, а оно в свою очередь – реляционной алгебре.

4. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ НА ОСНОВЕ НОРМАЛИЗАЦИИ

4.1. НОРМАЛИЗАЦИЯ ОТНОШЕНИЙ, ЦЕЛИ НОРМАЛИЗАЦИИ

Как уже отмечалось, задача проектирования реляционной базы данных заключается в выборе схемы базы из множества альтернативных вариантов, т. е., по сути, требуется определить набор схем отношений базы данных. Для удовлетворения этих требований необходимо определить, из каких отношений должна состоять база данных и какие атрибуты должны входить в эти отношения. Основной подход заключается в следующем. Предполагается существование некоторого универсального отношения, содержащего все атрибуты базы данных, затем на основе анализа связей между атрибутами пытаются осуществить декомпозицию универсального отношения, т. е. перейти к нескольким отношениям меньшей размерности, удовлетворяющих определенным условиям, причем исходное отношение должно восстанавливаться с помощью операции естественного соединения реляционной алгебры. Прежде чем перейти к рассмотрению методики построения оптимальной схемы базы данных, рассмотрим наиболее часто встречающиеся недостатки схем отношений, или аномалии отношений. Например, рассмотрим схему отношения:

Изготовители = (назв_изгот, адрес_изгот, изделие, колич_за_год, цена)

Эта схема имеет ряд недостатков.

1. *Избыточность*. Адрес изготовителя повторяется для каждого изготовляемого изделия.

2. *Потенциальная противоречивость* (аномалии обновления). Вследствие избыточности возможно обновление адреса изготовителя в одном кортеже, оставляя его неизменным в другом, т. е. возможна ситуация, когда база данных содержит различные адреса для одного изготовителя.

3. *Аномалии включения*. В базу данных не может быть записан адрес изготовителя, если не известно, какие изделия и в каком количестве он изготовляет.

4. *Аномалии удаления*. Обратная проблема появляется при необходимости удаления всех изделий, изготавливаемых определенным изготовителем, вследствие чего мы теряем его адрес, что не всегда желательно.

В этом примере все перечисленные недостатки исчезают, если заменить исходное отношение двумя отношениями:

Изг_адр = (назв_изгот, адрес_изгот)
Изг_изд = (назв_изгот, изделие, колич_за_год, цена)

Однако приведенная декомпозиция имеет существенный недостаток: чтобы получить адреса изготовителей, требуется выполнить операцию естественного соединения, которая работает сравнительно медленно, тем не менее, приведенная декомпозиция явно предпочтительнее исходной схемы отношения.

Теория реляционных баз данных обладает мощным инструментом, который способен помочь разработчику оптимальным образом спроектировать структуру отношений базы данных. Этот инструмент – метод нормализации отношений. ***Нормализация отношений*** – пошаговый процесс разложения (декомпозиции) исходных отношений базы данных на более простые. Каждая ступень этого процесса приводит схему отношений базы данных в последовательные нормальные формы. Каждая следующая нормальная форма обладает «лучшими» свойствами, чем предыдущая. Каждой нормальной форме соответствует некоторый набор ограничений. Отношение находится в определенной нормальной форме, если оно удовлетворяет набору ограничений этой формы.

Процесс нормализации основан на понятии функциональной зависимости атрибутов.

4.2. СТРУКТУРА ФУНКЦИОНАЛЬНЫХ ЗАВИСИМОСТЕЙ

4.2.1. Функциональные зависимости и их свойства

Пусть $R = (U)$ – схема отношения, $U = \{A_1, A_2, ..., A_n\}$ – множество ат-

рибутов, X, $Y \subseteq U$. Говорят, что X функционально определяет Y или что Y функционально зависит от X, и обозначают это $X \to Y$, если в любом отношении $\underline{R}$, являющемся текущим значением схемы R, не могут содержаться два кортежа, компоненты которых совпадают по всем атрибутам, принадлежащим множеству X, но не совпадают хотя по одному атрибуту, принадлежащему множеству Y.

Функциональные зависимости возникают различным образом, например если R содержит описание набора объектов и A_1, A_2, ..., A_n – атрибуты этого типа объектов, а X – множество атрибутов, образующих его ключ, то можно утверждать, что $X \to Y$ для любого $Y \subseteq \{ A_1, A_2, ..., A_n\}$. Это следует из того, что кортежи $\underline{R}$ представляют объекты, а объекты однозначно идентифицируются значениями атрибутов ключа, следовательно, два кортежа, совпадающие по атрибутам, принадлежащим X, должны представлять один и тот же объект и поэтому являются одним и тем же кортежем.

Следует отметить, что функциональные зависимости являются утверждениями обо всех отношениях, которые могут быть значениями схемы R, т. е. функциональная зависимость – это свойство схемы, а не конкретного экземпляра отношения. Следовательно, невозможно, анализируя конкретное текущее значение схемы R, определить, какие зависимости имеют место для R, в лучшем случае можно утверждать о некоторых зависимостях, что они не имеют места в схеме R. Единственный способ определения функциональных зависимостей для схемы отношения заключается в том, чтобы проанализировать семантику атрибутов. В этом смысле зависимости являются фактически высказываниями о предметной области, они не могут быть доказаны формальными средствами проектирования схем баз данных, хотя, как будет показано ниже, можно выводить новые (не заданные явно при описании схемы) зависимости из уже заданных.

Пусть задано множество атрибутов $U = \{ A_1, A_2, ..., A_n\}$ и некоторое множество функциональных зависимостей F, записанных в виде пар подмножеств (X, Y), $X, Y \subseteq U$, таких, что $X \to Y$; соответствующую схему отношения будем записывать в виде $R = (U, F)$. Говорят, что зависимость $A \to B$ ***логически следует*** из F, если для каждого экземпляра отношения $\underline{R}$ со схемой R, удовлетворяющего зависимостям F, выполняется также зависимость $A \to B$. Например, легко показать, что если $X \to Y$ и $Y \to Z$, то $X \to Z$.

Пусть F^+ обозначает ***замыкание*** F, т. е. множество всех функциональных зависимостей, которые логически следуют из F (включая, естест-

венно, само F); F при этом называют системой образующих структуры функциональных зависимостей. Если $F^+ = F$, то F называют ***замкнутым*** множеством зависимостей. Теперь рассмотрим задачу поиска зависимостей, логически следующих из заданного набора F.

Как показал Армстронг, совокупность всех пар (X, Y), таких, что $X, Y \subseteq U$ и $X \to Y$, образует структуру функциональных зависимостей отношения R, которая характеризуется следующим набором аксиом, называемых ***аксиомами Армстронга***:

1) если $X \supseteq Y$, то $X \to Y$ (рефлексивность);

2) если $X \to Y$ и $W \supseteq Z$, то $X \cup W \to Y \cup Z$ (продолжение);

3) если $X \to Y$ и $Y \to Z$, то $X \to Z$ (транзитивность).

Легко видеть, что аксиомы 2) и 3) могут быть объединены в одну аксиому:

4) если $X \to Y$ и $Y \cup W \to Z$, то $X \cup W \to Z$ (псевдотранзитивность).

Полезны также следующие свойства, вытекающие из 1)–3):

5) если $X \to Y$ и $X \to Z$, то $X \to Y \cup Z$ (аддитивность);

6) если $X \to Y$ и $Z \subseteq Y$, то $X \to Z$ (декомпозиция).

Аксиомы Армстронга можно рассматривать как правила вывода, позволяющие выводить функциональные зависимости, логически следующие из заданного набора зависимостей.

Аксиомы Армстронга являются *надежными* и *полными*, иными словами, если зависимость $X \to Y$ выведена из F по этим аксиомам, то она выполняется в любом отношении, в котором выполняются все зависимости из F, и наоборот, если некоторая зависимость $X \to Y$ не выводится из F по аксиомам Армстронга, то можно построить экземпляр отношения, для которого выполняются все зависимости из F и не выполняется зависимость $X \to Y$.

Пусть задана схема отношения $R = (U, F)$, $X \subseteq U$, ***замыканием*** множества атрибутов X относительно набора зависимостей F называется множество всех таких атрибутов $A_i \in U$, что зависимость $X \to A_i$ выводится из F по аксиомам Армстронга; замыкание X обозначают X^+.

Таким образом, X^+ – максимальное по включению подмножество множества U, функционально зависимое от X при заданном наборе функций F. Ясно, что зависимость $(X \to Y) \in F$ тогда и только тогда, когда $X^+ \supseteq Y$.

Отметим следующие свойства замыканий:

1) $X^+ \supseteq X$;

2) $X \supseteq Y \Rightarrow X^+ \supseteq Y^+$;

3) $(X^+)^+ = X^+$.

Аналогичные свойства имеют и замыкания наборов функций.

Вычисление F^+ – довольно трудоемкая задача, поскольку F^+ может быть довольно большим, даже если F мало, и может содержать порядка 2^n элементов, где n – количество атрибутов в отношении. Фактически вычисление F^+ сводится к вычислению замыканий подмножеств атрибутов.

Аксиомы Армстронга не являются достаточно удобными для непосредственного применения при вычислении замыканий подмножеств атрибутов, и обычно используется описанный ниже алгоритм, его правильность легко обосновывается теми же аксиомами Армстронга. Этот алгоритм требует времени пропорционально длине всех выписанных зависимостей из набора F.

Пусть задана схема $R = (U, F)$ и $X \subset U$, требуется вычислить X^+.

Шаг 1: положить $X^{(0)} := X$, $i := 0$;

Шаг 2: найти $\Delta X^{(i+1)}$ – множество всех таких атрибутов $y \in U \setminus X^{(i)}$, что существует некоторая зависимость $V \to W \in F$, такая что $X^{(i)} \supseteq V$ и $y \in W$;

Шаг 3: если $\Delta X^{(i+1)} = \varnothing$, то $X^{(i)} = X^+$ и конец работы, иначе $X^{(i+1)} := X^{(i)} \cup \Delta X^{(i+1)}$, $i := i + 1$ и перейти к шагу 2.

Поскольку множество атрибутов U конечно, то алгоритм всегда заканчивает работу за конечное число итераций.

П р и м е р

Пусть $R = (U, F)$ – схема отношения, где $U = \{A_1, A_2, \ldots, A_7\}$,
$F = \{A_1, A_4 \to A_2; A_2 \to A_3; A_3 \to A_4; A_4, A_7 \to A_5; A_5 \to A_6; A_6 \to A_7\}$,
множество $X = \{A_3, A_5\}$.

Требуется вычислить X^+.

Выпишем последовательность действий для каждого этапа.

$X^{(0)} = (A_3, A_5)$;

$\Delta X^{(1)} = \{A_4, A_6\}$, $X^{(1)} = \{A_3, A_4, A_5, A_6\}$;

$\Delta X^{(2)} = \{A_7\}$, $X^{(2)} = \{A_3, A_4, A_5, A_6, A_7\}$;

$\Delta X^{(3)} = \varnothing$, следовательно, $X^+ = X^{(2)} = \{A_3, A_4, A_5, A_6, A_7\}$.

4.2.2. Ключи схем отношений

Рассматривая наборы объектов, мы предполагали, что существует ключ – набор атрибутов, однозначно определяющий объект. Аналогичное понятие существует и для отношения с заданным множеством функциональных зависимостей, т. е. набор атрибутов, однозначно определяющий кортеж отношения.

Пусть задана схема отношения $R = (U, F)$, $X \subseteq U$;

Множество X называют ***сверхключом*** схемы отношения R, если $X^+ = U$, т. е. зависимость $X \to U$ принадлежит F^+ .

Минимальный по включению сверхключ называется ***ключом***, т. е. X – ключ схемы R, если:

1) $X^+ = U$;

2) ни для какого собственного подмножества $Y \subset X$, $Y^+ \neq U$.

В литературе часто такой ключ называют возможным ключом, а собственно ключом называют некоторый определенный возможный помеченный ключ; в свою очередь такой помеченный ключ иногда называют первичным, а остальные – просто ключами. Здесь эти термины («возможный ключ», «первичный ключ») не используются, поскольку они связаны с семантикой базы.

В принципе отношение может иметь несколько ключей. Если атрибут содержится в некотором ключе, то его называют ***ключевым*** или ***первичным*** атрибутом отношения, в противном случае атрибут называют ***непервичным***.

Задача нахождения произвольного «первого попавшегося» ключа отношения решается довольно просто. Это можно сделать следующим образом: рассматривается все множество атрибутов U, из него вычеркивается любой атрибут, получаем множество V; если $V^+ = U$, то вычеркнутый атрибут так и остается вычеркнутым, иначе возвращаем его на место. Далее пытаемся вычеркнуть следующий атрибут, так продолжаем до тех пор, пока ни один атрибут нельзя будет вычеркнуть. Оставшийся набор атрибутов является ключом отношения.

Вычисление всех ключей отношения – задача довольно трудоемкая, поскольку ключей может быть экспоненциально много. Задачи нахождения минимального по количеству атрибутов ключа, а также проверки, является ли конкретный атрибут ключевым (первичным), относятся к классу *NP*-трудных, т. е. не имеют алгоритмов решения с полиномиальной оценкой трудоемкости.

П р и м е р

Для схемы отношения $R = (U, F)$, где $U = \{A_1, A_2, \ldots, A_8\}$,
$F = \{A_1 \to A_2; A_2 \to A_3; A_3 \to A_1; A_2, A_3 \to A_4; A_4, A_7 \to {}_{A6};$
$A_6, A_5 \to A_7; A_7 \to A_8\}$ требуется найти все ключи, наборы первичных и непервичных атрибутов и сверхключ. В результате решения получаем:

1) Все ключи отношения:

$K_1 = \{A_1, A_5, A_6\}$, $K_2 = \{A_1, A_5, A_7\}$, $K_3 = \{A_2, A_5, {}_{A6}\}$,

$K_4 = \{A_2, A_5, A_7\}$, $K_5 = \{A_3, A_5, A_6\}$, $K_6 = \{A_3, A_5, A_7\}$.

2) Множество первичных атрибутов: $\{A_1, A_2, A_3, A_5, A_6, A_7\}$.

3) Множество непервичных атрибутов: $\{A_4, A_8\}$.

Ясно, что сверхключом является любое множество атрибутов, содержащее ключ. В приведенном примере сверхключом является множество $\{A_1, A_2, A_5, A_6\}$.

4.2.3. Полные и неполные функциональные зависимости

Зависимость $X \to Y$ называется ***неполной***, если существует такой атрибут $x \in X$, что $(X \setminus \{x\})^+ \supseteq Y$, т. е., если, вычеркнув некоторый атрибут x из X, получим X' такое, что $(X' \to Y) \in F^+$. Если такого атрибута x в левой части зависимости не существует, зависимость называется ***полной***. В дальнейшем будем рассматривать только такие неполные зависимости $X \to Y$, где $Y \not\subset X$.

Система образующих структуры функциональных зависимостей называется ***элементарной***, если она содержит только полные зависимости.

Перейти от заданной зависимости $X \to Y$ к полной можно вычеркиванием из X «избыточных» атрибутов, т. е. вычеркнув некоторый атрибут x и получив X', проверяем $(X')^+ \supseteq Y$, если да, то атрибут x остается вычеркнутым, если нет – возвращается на место. Так проверяются все атрибуты множества X. Ясно, что по одной заданной зависимости возможно получение нескольких полных зависимостей (в зависимости от порядка вычеркивания атрибутов).

П р и м е р

Для схемы $R = (U, F)$, где $U = \{A_1, A_2, A_3, A_4\}$, $F = \{A_1 \to A_2; A_2 \to A_3; A_3 \to A_4; A_1, A_2 \to A_4\}$ определить неполные зависимости, входящие в множество зависимостей F.

Зависимость $A_1, A_2 \to A_4$ не является полной, поскольку зависимость $(A_1 \to A_4) \in F^+$ и зависимость $(A_2 \to A_4) \in F^+$.

4.2.4. Покрытие множества зависимостей

Пусть F_1 и F_2 – множества функциональных зависимостей на множестве атрибутов U. Говорят, что F_1 и F_2 ***эквивалентны***, если $F_1^+ = F_2^+$. В этом случае говорят также, что F_1 покрывает F_2 (и F_2 покрывает F_1). Легко проверить, являются ли F_1 и F_2 эквивалентными. Для этого не обязательно строить замыкания F_1^+ и F_2^+. Достаточно для каждой зависимости $(X \to Y) \in F_1$ проверить, содержится ли эта зависимость в F_2^+. Для этого проверяют, содержится ли Y в $X_{F_1}^+$, (индекс внизу означает, что замыкание строится относительно набора функций F_2); в свою очередь

для каждой зависимости $(V \to W) \in F_2$ проверяют, содержится ли она в F_1^+. Если эти условия выполняются, то F_1 и F_2 эквивалентны, в противном случае – неэквивалентны.

Заметим, что когда от заданных зависимостей переходим к полным, то получаем набор функций, эквивалентный исходному.

Кроме того, всегда можно перейти к набору функций, эквивалентному исходному и такому, что в правой части находится только один атрибут.

Говорят, что множество зависимостей F является ***минимальным покрытием*** или ***элементарным функциональным базисом*** структуры функциональных зависимостей, если:

1) правая часть каждой зависимости из F содержит единственный атрибут;

2) ни для какой зависимости $(X \to Y) \in F$ множество $F \setminus (X \to Y)$ не эквивалентно F;

3) все зависимости набора F полные.

При определении функционального базиса иногда не требуют выполнения условий 1) и 3), а функциональный базис, удовлетворяющий 3), называют элементарным.

Заметим, что одному набору функций F может соответствовать несколько элементарных функциональных базисов.

П р и м е р

Для схемы $R = (U, F)$, где $U = \{A_1, A_2, A_3, A_4\}$,

$F = \{A_1, A_2 \to A_3; A_3 \to A_4; A_1 \to A_4; A_3, A_4 \to A_1; A_1, A_4 \to A_2; A_2 \to A_1; A_3 \to A_1; A_4 \to A_1\}$ функциональными базисами являются, например, следующие множества функций:

$F_1 = \{A_1 \to A_2; A_2 \to A_3; A_3 \to A_4; A_4 \to A_1\}$,

$F_2 = \{A_1 \to A_2; A_1 \to A_3; A_1 \to A_4; A_2 \to A_1; A_3 \to A_1; A_4 \to A_1\}$.

Легко видеть, что $F_1^+ = F_2^+ = F^+$ и при этом F_1 и F_2 удовлетворяют приведенным выше трем условиям.

4.2.5. Декомпозиция схем отношений

Декомпозицией схемы отношения $R = (U, F)$ называется замена ее совокупностью $\rho = \{R_1, R_2, ..., R_k)$, где $R_i = (U_i)$, $i = 1, 2, ..., k$, такой, что $U_1 \cup U_2 \cup \ldots \cup U_k = U$. При этом не требуется, чтобы U_i были непересекающимися. R_i будем называть ***подсхемами*** отношения R.

Соединение без потерь. Пусть задана схема отношения $R = (U, F)$ и ее декомпозиция $\rho = (R_1, R_2, \ldots, R_k)$. Говорят, что эта декомпозиция обладает свойством ***соединения без потерь*** относительно F, если каждое от-

ношение $\underline{R}$ для R, удовлетворяющее F, может быть представлено в виде

$$\underline{R} = \pi_{R_1}(\underline{R}) |><| \pi_{R_2}(\underline{R}) |><| \ldots |><| \pi_{R_k}(\underline{R}),$$

т. е. $\underline{R}$ является естественным соединением его проекций на все R_i. Ясно, что свойство соединения без потерь весьма желательно, поскольку в этом случае может быть восстановлено исходное отношение.

Рассмотрим основные свойства отображения проекция-соединение.

Если $\rho = (R_1, R_2, ..., R_k)$, обозначим через $M\rho$ отображение, которое определяется соотношением

$$M\rho(\underline{R}) = \pi_{R_1}(\underline{R}) |><| \pi_{R_2}(\underline{R}) |><| \ldots |><| \pi_{R_k}(\underline{R}).$$

Таким образом, условие соединения без потерь относительно F может быть выражено следующим образом: для всех $\underline{R}$, удовлетворяющих F, $\underline{R} = M\rho(\underline{R})$.

Для любой декомпозиции выполняются следующие условия:

а) $\underline{R} \subseteq M\rho(\underline{R})$;

б) если $\underline{S} = M\rho(\underline{R})$, то $\pi_{R_i}(\underline{S}) = R_i$;

в) $M\rho(M\rho(\underline{R})) = M\rho(\underline{R})$.

Свойство а) означает, что декомпозиция представляет, вообще говоря, некоторое большее по количеству кортежей отношение, чем исходное, и если не выполнено условие соединения без потерь, то мы можем получить кортежи, которых нет в исходном отношении, что естественно нарушает адекватность базы данных.

Для проверки свойства соединения без потерь можно использовать следующий алгоритм.

Пусть заданы схема отношения $R = (\{A_1, A_2, ..., A_n\}, F)$ и декомпозиция $\rho = (R_1, R_2, ..., R_k)$, $R_i = (U_i)$, $i = 1, 2, ..., k$.

Строим таблицу с n столбцами и k строками, столбец j соответствует атрибуту A_j, а строка i – схеме отношения R_i. На пересечении строки i и столбца j поместим символ A_j, если $A_j \in U_i$. В противном случае поместим туда символ *.

Просматриваем каждую из зависимостей $X \to Y$. Рассматривая зависимость $X \to Y$, изменяем строки, которые совпадают по всем столбцам, соответствующим атрибутам из X. При обнаружении двух таких строк отождествляем их символы в столбцах, соответствующих атрибутам из Y. Если при этом один из символов в одной из строк равен A_j, а символ другой строки в этом же столбце равен *, то заменяем * на A_j. Повторяем просмотр зависимостей до тех пор, пока: либо некоторая строка станет равной $A_1, A_2, ..., A_n$; либо больше изменений в таблице провести нельзя.

В первом случае декомпозиция ρ обладает свойством соединения без потерь. Во втором – нет.

П р и м е р ы

Пусть для схемы $R = (\{A_1, A_2, A_3, A_4, A_5\}, \{A_1 \to A_2; A_2 \to A_3; A_3, A_4 \to A_5; A_2 \to A_4\})$ получена декомпозиция $\rho = (R_1, R_2, R_3)$, где

$R_1 = (\{A_1, A_2\})$, $R_2 = (\{A_2, A_3, A_4\})$, $R_3 = (\{A_3, A_4, A_5\})$.

Требуется проверить, обладает ли она свойством соединения без потерь. Построим следующие таблицы:

1-я (начальная таблица):

A_1	A_2	*	*	*
*	A_2	A_3	A_4	*
*	*	A_3	A_4	A_5

2-я таблица:

A_1	A_2	A_3	A_4	*
*	A_2	A_3	A_4	A_5
*	*	A_3	A_4	A_5

3-я таблица:

A_1	A_2	A_3	A_4	A_5
*	A_2	A_3	A_4	A_5
*	*	A_3	A_4	A_5

В последней таблице первая строка представляет собой все A_j, следовательно, декомпозиция обладает свойством соединения без потерь.

Теперь пусть для схемы

$R = (\{A_1, A_2, A_3, A_4, A_5\}, \{A_1 \to A_2; A_2 \to A_3; A_3, A_5 \to A_4\})$ получена декомпозиция $\rho = (R_1, R_2)$, где $R_1 = (\{A_1, A_2\})$; $R_2 = (\{A_2, A_3, A_4, A_5\})$. Построим следующие таблицы:

1-я таблица:

A_1	A_2	*	*	*
*	A_2	A_3	A_4	A_5

2-я таблица:

A_1	A_2	A_3	*	*
*	A_2	A_3	A_4	A_5

Больше никаких изменений в таблице сделать нельзя, и строку, содержащую только A_i, мы не получили, значит, декомпозиция в этом слу-

чае не обладает свойством соединения без потерь.

Справедливо следующее утверждение.

Если $\rho = (R_1(U_1), R_2(U_2))$ – декомпозиция $R = (U, F)$, то ρ обладает свойством соединения без потерь относительно F тогда и только тогда, когда $((U_1 \cap U_2) \to U_1 \setminus U_2) \in F^+$ или $((U_1 \cap U_2) \to U_2 \setminus U_1) \in F^+$.

Это утверждение дает довольно простой способ проверки свойства соединения без потерь при декомпозиции на две подсхемы.

4.2.6. Декомпозиции, сохраняющие зависимости

Как уже отмечалось, желательно, чтобы декомпозиция обладала свойством соединения без потерь. Это является гарантией того, что любое отношение, являющееся текущим значением схемы, может быть восстановлено из его проекций на подсхемы декомпозиции. Другое важное свойство декомпозиции $\rho = (R_1, R_2, ..., R_k)$ схемы отношения R заключается в том, чтобы множество зависимостей F, заданных для R, было выводимым из проекций F на подсхемы R_i.

Формально *проекцией F на множество атрибутов Z*, обозначаемой $\pi_Z(F)$, называется множество зависимостей $X \to Y$ в F^+, таких, что $X, Y \subseteq Z$.

Часто удобнее вместо $\pi_Z(F)$ рассматривать его минимальное покрытие. Например, если имеется $R = (\{A_1, A_2, A_3, A_4\}, \{A_1 \to A_2; A_2 \to A_3; A_3 \to A_4\})$, $Z = \{A_1, A_3, A_4\}$, то $\pi_Z(F) = \{A_1 \to A_3; A_3 \to A_4\}$. Здесь записано минимальное покрытие проекции F.

Декомпозиция ρ сохраняет множество зависимостей F, если из объединения всех зависимостей, принадлежащих $\pi_{R_i}(F)$, $i = 1, 2, \ldots, k$, логически следуют все зависимости F.

Стремление к тому, чтобы ρ сохраняла F, естественно. Зависимости в F могут рассматриваться как ограничения целостности для схемы R. Если из спроецированных зависимостей не следует F, то возможны такие текущие значения Ri, представляющие отношение R в декомпозиции ρ, которые не удовлетворяют F, даже тогда, когда ρ обладает свойством соединения без потерь относительно F. Чтобы избежать подобных аномалий в случае обновления базы данных необходимо осуществлять операцию естественного соединения, позволяющую проверить, не нарушаются ли зависимости из F.

Следует отметить, что декомпозиция может обладать свойством соединения без потерь, но не сохранять F, например:

Для схемы $R = (\{A_1, A_2, A_3\}, \{A_1, A_2 \to A_3; A_3 \to A_1\})$ получена декомпозиция $\rho = (R_1, R_2)$, где $R_1 = (\{A_1, A_2\})$, $R_2 = (\{A_2, A_3\})$.

Декомпозиция $\rho = (R_1, R_2)$ обладает свойством соединения без потерь, это следует из того, что $\{A_1, A_2\} \cap \{A_2, A_3\} = \{A_2\}$ и $A_2 \to A_3 = U_2 \setminus U_1$. В то же время зависимость $A_1, A_2 \to A_3$ не выводится из проекций F на множества U_1 и U_2.

Возможен и обратный случай, когда декомпозиция сохраняет зависимости, но не обладает свойством соединения без потерь. Например, это имеет место для следующей схемы R и ее декомпозиции:

$R = (\{A1, A2, A3, A4\}, \{A1 \to A2; A3 \to A4\}), \ \rho = (\{A1, A2\}, \{A3, A4\})$.

В теории проектирования реляционных баз данных разработано несколько нормальных форм схем отношений, в различной степени устраняющих перечисленные ранее возможные недостатки исходных схем.

4.3 НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ

4.3.1. Первая и вторая нормальные формы схем отношений

Говорят, что отношение нормализовано или находится в ***первой нормальной форме***, если домен каждого атрибута состоит из неделимых значений, а не множеств или кортежей из элементарного домена или доменов, т. е. значения атрибутов – это некоторые элементарные величины, не имеющие структуры. В дальнейшем будем рассматривать только нормализованные отношения.

Понятие второй нормальной формы тесно связано с понятием полной функциональной зависимости.

Вторая нормальная форма схемы отношения – это либо данная схема, если она нормализована и содержит только полные функциональные зависимости непервичных атрибутов от ключей, либо декомпозиция схемы, каждая из подсхем которой обладает указанными свойствами, а сама декомпозиция имеет свойство соединения без потерь.

Заметим, что если в схеме отношения все атрибуты первичные или же любой ключ состоит только из одного атрибута, то схема заведомо находится во второй нормальной форме.

Вторая нормальная форма существует для любой схемы отношения, это следует из следующего простого утверждения.

Теорема Хита. Пусть задана схема отношения $R = (U, F)$ и $U = X \cup Y \cup Z$, $X \to Y$, тогда декомпозиция $R = (R_1, R_2)$, где $R_1 = (X \cup Y)$, $R_2 = (X \cup Z)$, обладает свойством соединения без потерь.

Это утверждение дает очевидный способ приведения схемы отношения с неполной функциональной зависимостью набора непервичных атрибутов B от ключа K ко второй нормальной форме.

Пусть U – набор всех атрибутов отношения R. Присутствие в отношении неполной зависимости B от ключа K означает, что $\exists\, A\ ((A \subset K) \wedge (B \not\subset A) \wedge (A \to B))$. Построим декомпозицию $R_1 = (A \cup B)$, $R_2 = (U \setminus B)$. Отношения R_1 и R_2 уже не содержат указанной неполной зависимости, и полученная декомпозиция обладает свойством соединения без потерь. Далее аналогичные проверки применяются к отношениям R_1 и R_2, так продолжается до тех пор, пока не получим декомпозицию $\rho = (R_1, R_2, ..., R_k)$, каждая из подсхем которой не содержит неполных зависимостей непервичных атрибутов от ключей. Декомпозиция ρ и является искомой второй нормальной формой схемы R.

Описанный способ обосновывается также следующими свойствами декомпозиции.

а) Пусть задана схема отношения $R = (U, F)$, $\rho = (R_1, R_2, ..., R_k)$ – декомпозиция R, обладающая свойством соединения без потерь относительно F, а F_i для каждого i означает проекцию U на R_i. Допустим, что $\sigma = (S_1, S_2, ..., S_m)$ – декомпозиция R_i, обладающая свойством соединения без потерь относительно F_i. Тогда декомпозиция R в подсхемы $(R_1, ..., R_{i-1}, S_1, ..., S_m, R_{i+1}, ..., R_k)$ – также обладает свойством соединения без потерь относительно F.

б) Предположим, что R, F, ρ те же, что в пункте а). Пусть $\tau = (R_1, R_2, ..., R_k, R_{k+l}, ..., R_t)$ – декомпозиция R в некоторое множество схем, которое включает все схемы из ρ. Тогда τ также обладает свойством соединения без потерь относительно F.

Таким образом, если декомпозиция обладает свойством соединения без потерь, то «разбив» некоторую схему декомпозиции на еще более мелкие подсхемы, обладающие свойством соединения без потерь относительно разбиваемой подсхемы, опять получим декомпозицию исходной схемы, обладающую этим свойством. Добавление к декомпозиции любого количества подсхем не нарушает свойства соединения без потерь. Эти свойства часто используются также для приведения отношений к третьей и усиленной третьей (рассмотрены ниже) нормальным формам.

Подробнее остановимся также на задаче проектирования набора функций F на схемы декомпозиции. В общем случае это задача довольно трудоемкая, но при приведении отношений к нормальным формам чаще всего в отдельные подсхемы выделяются некоторые зависимости из F. Например можно поступать следующим образом.

Пусть задана схема $R = (U, F)$, считаем, что все зависимости в F в правой части содержат только один атрибут (всегда можно преобразовать F к такому виду). Допустим, строится декомпозиция $R_1 = (U_1)$ и $R_2 = (U_2)$, причем $U_1 = X \cup \{y\}, X \subset U, y \in U, (X \to y) \in F^+$; $U_2 = U \setminus \{y\}$.

Для получения проекции F на U_2 достаточно найти все зависимости вида $X_i \to y$, содержащиеся в F. Затем каждую зависимость, содержащую атрибут y в левой части заменить группой зависимостей, подставляя вместо y множества X_i. Те зависимости, где атрибут y находится в правой части, удаляются, в результате получим проекцию F на U_2 (точнее, покрытие проекции). Для получения проекции F на U_1 придется просмотреть замыкания подмножеств множества U_1 относительно F, но множество U_1 обычно невелико.

Заметим также, что всегда удобнее работать, если все зависимости F полные (такое преобразование F сделать не сложно), модифицированные зависимости при получении проекции F на U_2 также имеет смысл приводить к полным. При рассмотрении U_1 в случае, если $X \to y$ – полная зависимость, следует просматривать только замыкания подмножеств, содержащих y.

Декомпозицию схемы отношения при переходе ко второй нормальной форме удобно строить в виде дерева. Пусть задана схема $R = (U, F)$. Вначале дерево состоит только из одной вершины, соответствующей отношению R. Далее находим зависимость непервичного атрибута y от части ключа X, $(X \to y) \in F$, и строим две нижестоящие вершины: одна из них соответствует отношению $R_1 = (U \setminus \{y\})$, вторая – отношению $R_2 = (X \cup \{y\})$, далее, если необходимо, аналогичную декомпозицию осуществляем для висящих вершин (листьев) дерева; этот процесс продолжается до тех пор, пока все висящие вершины не будут соответствовать подсхемам во второй нормальной форме, совокупность этих подсхем и является искомой декомпозицией исходной схемы отношения.

В принципе можно рассматривать вместо одного атрибута y в правых частях выделяемых зависимостей множества атрибутов, но тогда несколько усложнится задача проектирования F на подсхемы.

Желательно также рассматривать только полные функциональные зависимости.

П р и м е р

Для схемы $R = (U, F)$, $U = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$,

$F = \{A_1 \to A_3; A_3 \to A_5; A_2 \to A_4; A_4 \to A_6; A_5, A_6 \to A_7\}$ требуется получить вторую нормальную форму схемы отношения R. Отношение имеет единственный ключ $K = \{A_1, A_2\}$, множество непервичных атрибутов $\{A_3, A_4, A_5, A_6, A_7\}$.

Зависимость $A_1 \to A_3$ является нежелательной, так как непервичный атрибут A_3 зависит от части ключа. Выделяем эту зависимость в отдельное отношение, получаем:

$R_1 = (\{A_1, A_2, A_4, A_5, A_6, A_7\},$

$\{A_1 \to A_5; A_2 \to A_4; A_4 \to A_6; A_5, A_6 \to A_7\})$.

Зависимость $A_1 \to A_5$ получена в результате проецирования F на подсхему R_1; ключом отношения R_1 является $\{A_1, A_2\}$.

$R_2 = (\{A_1, A_3\}, \{A_1 \to A_3\})$, единственный ключ $R_2 - \{A_1\}$.

R_2 находится во второй нормальной форме, в $R1$ нежелательной является зависимость $A_1 \to A_5$. Получаем декомпозицию R_1:

$R_{11} = (\{A_1, A_2, A_4, A_6, A_7\}, \{A_2 \to A_4; A_4 \to A_6; A_1, A_6 \to A_7\})$, ключ $R_{11} - \{A_1, A_2\}$,

$R_{12} = (\{A_1, A_5\}, \{A_1 \to A_5\})$, ключ $R_{12} - \{A_1\}$.

R_{12} находится во второй нормальной форме, в R_{11} нежелательной является зависимость $A_2 \to A_4$, получаем декомпозицию R_{11}:

$R_{111} = (\{A_1, A_2, A_6, A_7\}, \{A_2 \to A_6; A_1, A_6 \to A_7\})$, ключ – $\{A_1, A_2\}$,

$R_{112} = (\{A_2, A_4\}, \{A_2 \to A_4\})$, ключ $R_{112} - \{A_2\}$.

R_{112} находится во второй нормальной форме, в R_{111} нежелательной является зависимость $A_2 \to A_6$, получаем декомпозицию R_{111}:

$R_{1111} = (\{A_1, A_2, A_7\}, \{A_1, A_2 \to A_7\})$, ключ $R_{1111} - \{A_1, A_2\}$,

$R_{1112} = (\{A_2, A_6\}, \{A_2 \to A_6\})$, ключ $R_{1112} - \{A_2\}$.

R_{1111} и R_{1112} находятся во второй нормальной форме, искомая декомпозиция (вторая нормальная форма отношения R) имеет вид

$\rho = (R_2, R_{12}, R_{112}, R_{1111}, R_{1112})$.

Дерево декомпозиции имеет вид:

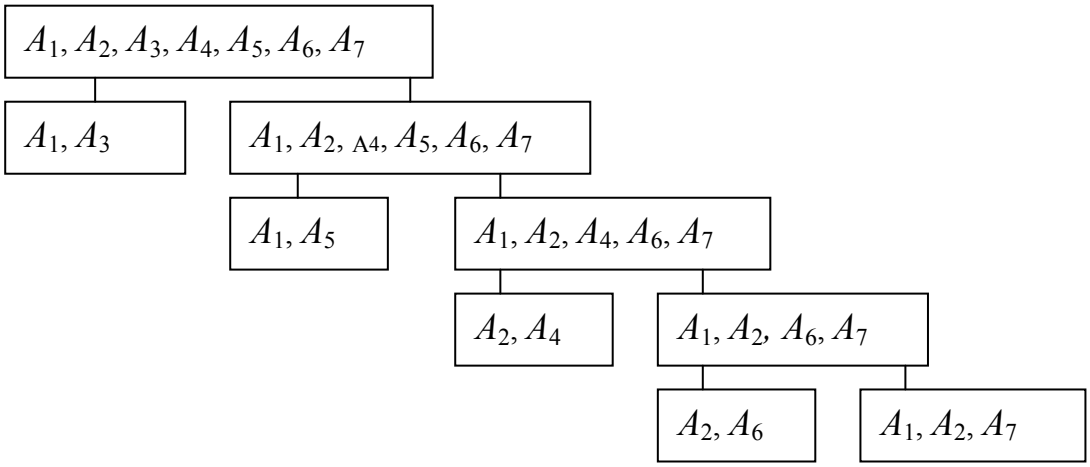


Схема отношения может иметь более одной второй нормальной формы. Так, в приведенном примере второй нормальной формой исходной схемы является также декомпозиция

$\rho_1 = (\{A_1, A_3\}, \{A_3, A_5\}, \{A_2, A_4\}, \{A_4, A_6\}, \{A_5, A_6, A_7\})$,

причем ρ удачнее, чем ρ_1, в смысле минимизации аномалий базы данных.

В приведенном примере ситуация упрощалась тем, что отношение имеет единственный ключ (рассматривать надо неполные зависимости непервичных атрибутов от всех ключей).

4.3.2. Третья нормальная форма схем отношений.

Понятие третьей нормальной формы схемы отношения тесно связано с понятием транзитивной зависимости, которая имеет вид цепочки функциональных зависимостей с определенными ограничениями.

Пусть задана схема отношения $R = (U, F)$, A, $C \subseteq U$. Функциональная зависимость $A \to C$ называется *транзитивной*, если существует подмножество атрибутов $B \subset U$ такое, что

$$A \not\supseteq B, B \not\supseteq C, A \to B, B \nrightarrow A \text{ и } B \to C.$$

Отметим, что наличие или отсутствие зависимости $C \to B$ значения не имеет. Если такой зависимости нет, то транзитивная зависимость называется *строгой* транзитивной зависимостью. Например, $R = (\{A_1, A_2, A_3, A_4, A_5\}, \{A_1 \to A_2; A_2 \to A_3; A_1 \to A_4; A_4 \to A_5; A_5 \to A_1\})$. Здесь зависимость $A_1 \to A_3$ является транзитивной, так как имеем цепочку

$$A_1 \to A_2 \to A_3 \text{ и } A_2 \nrightarrow A_1.$$

В то же время зависимость $A_1 \to A_5$ не является транзитивной, хотя и есть цепочка $A_1 \to A_4 \to A_5$, но здесь $(A_4 \to A_1) \in F^+$, что противоречит определению транзитивной зависимости.

Присутствие в схеме транзитивной зависимости $A \to B \to C$ приводит к избыточности данных о связи атрибутов B и C и, следовательно, избыточности значений атрибутов C, что может привести к нарушению целостности базы данных при изменении значений атрибутов из множеств C и B. Избыточность значений C возрастает, если дополнительно $B \nleftarrow C$. Замена схемы отношения, в которой есть транзитивная зависимость, декомпозицией, которая не содержит зависимостей такого типа, избавляет базу данных от указанных недостатков. Такая декомпозиция при выполнении определенных условий называется третьей нормальной формой отношения.

Третья нормальная форма (ТНФ) схемы отношения – это либо данная схема, если она находится во второй нормальной форме и не содержит транзитивной зависимости непервичных атрибутов от ключей, либо декомпозиция исходной схемы, каждая подсхема которой удовлетворяет этим же требованиям, а сама декомпозиция обладает свойством соединения без потерь.

Третья нормальная форма существует для любой схемы отношения, причем часто не единственная.

Возможность неоднозначного преобразования к ТНФ, а также желание сохранить в подсхемах более «близкие» взаимосвязи между атрибутами приводит к понятию оптимальной ТНФ, содержащей наименьшее число подсхем, и кроме того, в случае зависимости $A \to B \to C$ подсхемы не должны включать одновременно «несмежные» компоненты A и C.

Получить ТНФ (не всегда оптимальную) можно с помощью модификации алгоритма Хита, применявшегося для получения второй нормальной формы.

Суть алгоритма заключается в следующем. Пусть задана схема отношения $R = (U, F)$, $A, B, C \subset U$, $A \to B \to C$ – транзитивная зависимость (C состоит из непервичных атрибутов), тогда переходим к декомпозиции $\rho = (R_1 = (U_1, F_1), R_2 = (U_2, F_2))$, где $U_1 = U \setminus C$, $U_2 = B \cup C$, F_1 и F_2 – проекции F соответственно на U_1 и U_2. Затем, если необходимо, производится декомпозиция R_1 и R_2 и так далее до тех пор, пока все подсхемы не окажутся в ТНФ.

Заметим, что задача проверки, находится ли схема отношения в ТНФ, относится к *NP*-трудным, что связано с задачей выделения всех непервичных атрибутов отношения. В принципе можно выполнять декомпозицию зависимости $A \to B \to C$ не проверяя, находятся первичные атрибуты в C или нет, но это может привести к избытку подсхем, что снижает «качество» декомпозиции.

П р и м е р

Для схемы $R = (\{A_1, A_2, ..., A_6\}, \{A_1, A_2 \to A_3; A_3 \to A_4; A_4 \to A_5; A_4 \to A_6\})$ требуется получить ТНФ. Первичные атрибуты $\{A_1, A_2\}$, непервичные – $\{A_3, A_4, A_5, A_6\}$. Транзитивная зависимость: $A_1, A_2 \to A_3 \to A_4$.

На первом этапе получаем: $R_1 = (\{A_1, A_2, A_3, A_5, A_6\},$

$\{A_1, A_2 \to A_3; A_3 \to A_5; A_3 \to A_6\})$;

$R_2 = (\{A_3, A_4\}, \{A_3 \to A_4\})$. R_2 находится в ТНФ.

В $R1$ нежелательная транзитивная зависимость: $A_1, A_2 \to A_3 \to A_5$. Строим декомпозицию, получаем:

$R_{11} = (\{A_1, A_2, A_3, A_6\}, \{A_1, A_2 \to A_3; A_3 \to A_6\})$,

$R_{12} = (\{A_3, A_5\}, \{A_3 \to A_5\})$. R_{12} находится в ТНФ.

В R_{11} нежелательная транзитивная зависимость: $A_1, A_2 \to A_3 \to A_6$. Строим декомпозицию, получаем:

$R_{111} = (\{A_1, A_2, A_3\}, \{A_1, A_2 \to A_3\})$, $R_{112}=(\{A_3, A_6\}, \{A_3 \to A_6\})$.

R_{111} и R_{112} находятся в ТНФ.

Имеем ТНФ исходной схемы $\rho = (R_2, R_{12}, R_{111}, R_{112})$.

Следует отметить, что простота описанного метода только кажущаяся, основная сложность заключается в выявлении транзитивных зависимостей, которые не обязательно «видны» по системе образующих структуры функциональных зависимостей, например, в случае

$$R = (\{A_1, A_2, ..., A_5\}, \{A_1, A_2 \to A_3; A_1, A_2 \to A_4; A_3, A_4 \to A_5\})$$

транзитивной зависимостью является цепочка $A_1, A_2 \to A_3, A_4 \to A_5$, которая кажется очевидной только из-за малого количества атрибутов. Во многом процесс выявления транзитивных зависимостей облегчается графовым представлением схем отношений. Ниже приводятся алгоритмы приведения схемы отношения к ТНФ, не требующие выделения транзитивных зависимостей в явном виде, но часто дающие декомпозиции, далекие от оптимальных.

Алгоритм **Делобеля – Кейси.**

Пусть задана схема отношений $R = (U, F)$.

1. Перейти от F к элементарному функциональному базису F^*, т. е. к минимальному покрытию, состоящему только из полных зависимостей, причем в правой части каждой зависимости должен находиться только один атрибут.

2. По каждой зависимости $(X_i \to Y_i) \in F^*$ образовать подсхему $R_i = (U_i = X_i \cup Y_i)$; если при этом для некоторых i и j, $i \neq j$, окажется, что $U_j \subseteq U_i$, то R_j удаляется.

3. Если хотя бы одна из полученных подсхем содержит ключ исходного отношения (для этого достаточно проверить, выполняется ли хотя бы для одного U_i соотношение $F_i^+ = U$), то совокупность полученных R_i является ТНФ отношения R, иначе добавить к полученной декомпозиции еще одну схему, состоящую из произвольного ключа отношения R.

Заметим, что полученная декомпозиция сохраняет также зависимости исходной схемы, алгоритм имеет полиномиальную оценку трудоемкости, поскольку здесь не выделяется множество первичных атрибутов и даже не проверяется, находится ли исходное отношение в ТНФ, но такие упрощения естественно приводят к избытку подсхем в декомпозиции.

П р и м е р

Для схемы $R = (\{A_1, A_2, A_3, A_4, A_5\}, \{A_1, A_2 \to A_3; A_3 \to A_4; A_3, A_4 \to A_5\})$ получить ТНФ.

Переходим к элементарному функциональному базису. В зависимости $A_3, A_4 \to A_5$ атрибут $A4$ избыточный, получим $F^* = \{A_1, A_2 \to A_3; A_3 \to A_4; A_3 \to A_5)$. Строим декомпозицию

$$\rho = (\{A_1, A_2, A_3\}, \{A_3, A_4\}, \{A_3, A_5\}).$$

Поскольку $\{A_1, A_2, A_3\}^+ = \{A_1, A_2, A_3, A_4, A_5\} = U$, значит, U_1 содержит ключ исходного отношения и ρ является ТНФ отношения R.

Здесь подсхемы $\{A_3, A_4\}$ и $\{A_3, A_5\}$ можно объединить в одну $\{A_3, A_4, A_5\}$, но в общем случае такие объединения требуют дополнительной проверки, так как подсхема, полученная в результате объединения, может не быть в ТНФ.

Можно модифицировать приведенный алгоритм следующим образом.

1. Шаг 1 предыдущего алгоритма.

2. Найти множество всех первичных атрибутов исходного отношения U_p и непервичных $U \setminus U_p$.

3. Просмотреть все зависимости $X \to Y$ из F^*, если X не является ключом и Y – непервичный атрибут (если таких зависимостей в F нет, то R в ТНФ), строим декомпозицию $U_1 = X \cup Y$, $U_2 = U \setminus Y$, т. е. такие зависимости выделяются в отдельные отношения.

В отличие от предыдущего алгоритма, здесь получается, вообще говоря, меньше подсхем, но увеличивается трудоемкость за счет нахождения всех первичных атрибутов и декомпозиция не обязательно сохраняет зависимости.

4.3.3. Усиленная третья нормальная форма схем отношений (нормальная форма Бойса – Кодда)

Если третья нормальная форма схем отношений не содержит неполных и транзитивных зависимостей любых атрибутов от ключей, то она называется усиленной третьей нормальной формой (УТНФ).

Приведенное определение УТНФ эквивалентно следующему определению, в котором не используются такие понятия, как ключ, первичные и непервичные атрибуты, полная и транзитивная зависимости.

Усиленной третьей нормальной формой схемы отношения $R = (U, F)$ называется либо данная схема, если она нормализована и для любого набора атрибутов $A \subset U$ верно, что если какой-либо атрибут $x \in U \setminus A$ зависит функционально от A, то и все атрибуты отношения функционально зависят от A; либо декомпозиция схемы R, каждая подсхема которой удовлетворяет этим требованиям, обладающая свойством соединения без потерь.

Таким образом, R находится в УТНФ тогда и только тогда, когда для любой зависимости $(X \to Y)$ и $X \not\supseteq Y$ выполняется $X^+ = U$. Очевидно, что если отношение находится в УТНФ, то оно находится и в ТНФ, обратное, вообще говоря, не верно. УТНФ называют также ***нормальной фор-***

мой Бойса – Кодда. Любая схема отношения может быть приведена к УТНФ. Следующий алгоритм получения УТНФ основывается на теореме Хита.

Пусть задана схема отношения $R = (U, F)$. Требуется получить УТНФ схемы R.

1. Перейти от F к элементарному функциональному базису F^*, т. е. к минимальному покрытию, состоящему только из полных зависимостей, причем в правой части каждой зависимости должен находиться только один атрибут.

2. Далее декомпозицию ρ для R строим итеративным методом, при этом ρ всегда будет обладать свойством соединения без потерь. Первоначально ρ состоит только из R. Если S – схема из ρ и в S есть зависимость $X \to Y$, $X \supseteq Y$ и X не содержит ключа S, то заменяем S декомпозицией $S_1 = (U_1, F_1)$, $S_2 = (U_2, F_2)$, где $U_1 = X \cup Y$, $U_2 = U \setminus Y$ (здесь полагается, что $S = (U, F)$). Так продолжается до тех пор, пока все подсхемы ρ не окажутся в УТНФ.

Заметим, что пункт 1 выполнять не обязательно, но тогда сильно возрастет трудоемкость проектирования F на подсхемы.

П р и м е р

Для схемы $R = (U, F) = (\{A_1, A_2, ..., A_6\}, \{A_1 \to A_2; A_2, A_4 \to A_1; A_2 \to A_3; A_3 \to A_4; A_3, A_4 \to A_5; A_5 \to A_6\})$ требуется получить УТНФ. Перейдя к элементарному функциональному базису, получим

$$F^* = \{A_1 \to A_2; A_2 \to A_1; A_2 \to A_3; A_3 \to A_4; A_3 \to A_5; A_5 \to A_6\}.$$

Ш а г 1. Проверяем зависимости:

1) для зависимости $A_1 \to A_2$ выполняется $\{A_1\}^+ = U$, следовательно, зависимость $A_1 \to A_2$ удовлетворяет УТНФ;

2) для зависимостей $A_2 \to A_1$ и $A_2 \to A_3$ выполняется $\{A_2\}^+ = U$, следовательно, эти зависимости удовлетворяют УТНФ;

3) для зависимостей $A_3 \to A_4$ выполняется $\{A_3\}^+ \neq U$. Следовательно, зависимость $A_3 \to A_4$ не удовлетворяет УТНФ. Выделяем зависимость $A_3 \to A_4$ в отдельную схему, получаем:

$R_1 = (\{A_3, A_4\}, \{A_3 \to A_4\})$,

$R_2 = (\{A_1, A_2, A_3, A_5, A_6\}, \{A_1 \to A_2; A_2 \to A_1; A_2 \to A_3; A_3 \to A_5; A_5 \to A_6\})$.

Ш а г 2. R_1 находится в УТНФ. Проверяем зависимость $A_3 \to A_5$; для нее выполняется $\{A_3\}^+ \neq U_2$. Следовательно, она не удовлетворяет УТНФ. Выделяем зависимость $A_3 \to A_5$ в отдельную схему. Получаем:

$R_{21} = (\{A_3, A_5\}, \{A_3 \to A_5\})$,

$R_{22} = (\{A_1, A_2, A_3, A_6\}, \{A_1 \to A_2; A_2 \to A_1; A_2 \to A_3; A_3 \to A_6\})$.

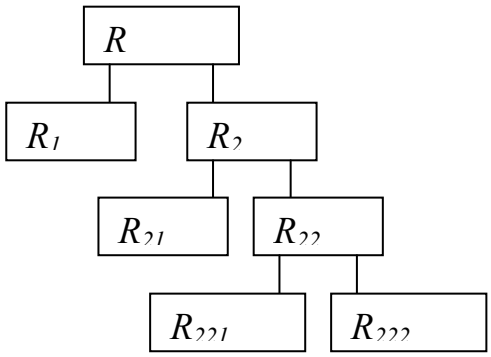
Ш а г 3. R_{21} находится в УТНФ. Рассматриваем зависимость $A_3 \to A_6$; для нее выполняется $\{A_3\}^+ \neq U_{22}$. Следовательно, она не удовлетворяет УТНФ. Выделяем зависимость $A_3 \to A_6$ в отдельную схему. Получаем:

$R_{221} = (\{A_3, A_6\}, \{A_3 \to A_6\})$,

$R_{222} = (\{A_1, A_2, A_3\}, \{A_1 \to A_2; A_2 \to A_1; A_2 \to A_3\})$. R_{221} и R_{222} находятся в УТНФ. Следовательно,

УТНФ $\rho = (R_1, R_{21}, R_{221}, R_{222})$ для схемы R.

Дерево декомпозиции имеет вид:



Однако полученная декомпозиция имеет следующие существенные недостатки.

1. R_1 и R_{21} можно объединить в одну схему, получим $R_1 = (\{A_3, A_4, A_5\}, \{A_3 \to A_4; A_3 \to A_5\})$. Заметим, что так делать можно не всегда. Например, если бы была зависимость $A_4 \to A_5$, то в R_1 получили бы транзитивную зависимость $A_3 \to A_4 \to A_5$.

2. A_3 и A_6 нецелесообразно объединять в одну подсхему, поскольку в исходной схеме есть транзитивная зависимость $A_3 \to A_5 \to A_6$, где A_3 и A_6 не смежны, и поэтому такое объединение приведет к избыточности данных в базе. Лучше вместо $\{A_3, A_6\}$ использовать подсхему $\{A_5, A_6\}$. Легко видеть, что такая замена не нарушает свойства соединения без потерь, попросту функциональные зависимости проверяются в другом порядке. Таким образом, пришли к декомпозиции:

$$\rho = ((\{A_3, A_4, A_5\}, \{A_3 \to A_4; A_3 \to A_5\}), (\{A_5, A_6\}, \{A_5 \to A_6\}), (\{A_1, A_2, A_3\}, \{A_1 \to A_2; A_2 \to A_1; A_2 \to A_3\})).$$

При больших количествах атрибутов такого рода улучшения требуют значительных затрат. Целесообразно поступать следующим образом.

1. Найти произвольный ключ K.

2. Построив K^+, выписать все $K^{(i)}$ (см. алгоритм построения замыканий).

3. Просматривать $K^{(i)}$ в обратном порядке и выделять в отдельные подсхемы зависимости $X \to Y$, нарушающие УТНФ и такие, что X и Y принадлежат $K^{(i)}$ с наибольшими номерами i.

Во многих случаях такой подход исключает включение в подсхемы несмежных транзитивных компонент.

Заметим, что не всегда можно преобразовать схему отношения к УТНФ с сохранением зависимостей.

Например, $R = (\{A_1, A_2, A_3\}, \{A_1, A_2 \to A_3; A_3 \to A_1\})$.

Схема R находится в ТНФ, но не в УТНФ, поскольку для зависимости $A_3 \to A_1$ выполняется $\{A_3\}^+ \neq \{A_1, A_2, A_3\}$. УТНФ дает декомпозиция: $R_1 = (\{A_1, A_3\}, \{A_3 \to A_1\})$, $R_2 = (\{A_2, A_3\})$, но при этом теряется зависимость $A_1, A_2 \to A_3$, и, следовательно, построить УТНФ с сохранением этой зависимости невозможно.

4.3.4. Четвертая нормальная форма схем отношений

На практике проектирование реляционной базы данных завершается после нахождения третьей или усиленной третьей нормальной формы исходной схемы отношения. Однако даже УТНФ не всегда избавляет от избыточности данных. Связано это с существованием так называемых многозначных зависимостей. В простейшем случае многозначные зависимости возникают при приведении исходного отношения к первой нормальной форме.

П р и м е р

ФИО_преподавателя	Группа	Вид_нагрузки
Иванов И.И.	1	Лекции
Иванов И.И.	1	Практика
Иванов И.И.	1	Лабораторные
Иванов И.И.	2	Лекции
Иванов И.И.	2	Практика
Иванов И.И.	2	Лабораторные
Сидоров С.С.	1	Практика
Сидоров С.С.	1	Лабораторные
Сидоров С.С.	2	Практика
Сидоров С.С.	2	Лабораторные

Предположим, что преподаватели некоторой кафедры должны вести занятия в каждой группе студентов, специализирующихся по данной кафедре. Кроме этого существует требование обязательного включения в общую нагрузку преподавателя определенных видов нагрузки (лекции,

практические, лабораторные занятия). Поскольку между номерами групп и видами нагрузки нет никакой прямой связи, то в отношении, содержащем атрибуты ФИО_преподавателя, Группа, Вид_нагрузки, каждый кортеж должен содержать комбинации групп и видов нагрузки. Данное отношение находится в усиленной третьей нормальной форме, так как имеется только один ключ, включающий все три атрибута. Тем не менее имеется явная избыточность данных из-за повторения для каждого преподавателя комбинации группы и вида нагрузки. Связано это с тем, что в отношении между атрибутами существуют две независимые связи типа «один ко многим»: ФИО_преподавателя : Группа, ФИО_преподавателя : Вид_нагрузки. При этом верно следующее ограничение: если кортежи (A, B, C) и (A, D, E) присутствуют одновременно, то кортежи (A, B, E) и (A, D, C) также присутствуют в отношении.

Ситуацию можно улучшить, если выполнить декомпозицию на две подсхемы (ФИО_преподавателя, Группа) и (ФИО_преподавателя, Вид_нагрузки).

ФИО_преподавателя	Группа
Иванов И.И.	1
Иванов И.И.	2
Сидоров С.С.	1
Сидоров С.С.	2

ФИО_преподавателя	Вид_нагрузки
Иванов И.И.	Лекции
Иванов И.И.	Практика
Иванов И.И.	Лабораторные
Сидоров С.С.	Практика
Сидоров С.С.	Лабораторные

Введем понятие многозначной зависимости.

Пусть R – схема отношений с атрибутами $U = \{A_1, \ldots, A_n\}$ и подмножествами $X, Y, Z \subset U$. Подмножество Y многозначно зависит от X, обозначаем $X \text{—>>} Y / Z$ тогда и только тогда, когда в каждом отношении со схемой R множество значений Y, соответствующее заданной паре значений (X, Z), зависит от X, но не зависит от Z.

Нетрудно показать, что для данной схемы $R(A, B, C)$ многозначная зависимость $A \text{—>>} B/C$ выполняется тогда и только тогда, когда выполняется многозначная зависимость $A \text{—>>} C/B$. То есть многозначные зависимости всегда образуют связные пары.

Многозначные зависимости являются обобщениями функциональных зависимостей в том смысле, что всякая функциональная зависимость является многозначной. Точнее говоря, функциональная зависимость $A \to B$ – это многозначная зависимость $A \text{—>>} B/(U \setminus (A \cup B))$, в которой множество зависимых значений B, соответствующее значению A, является одноэлементным множеством.

Теорема Фейгина. Пусть R – схема отношений с множеством атрибутов $U = A \cup B \cup C$. Декомпозиция $\rho = (R_1, R_2)$, где $R_1 = (A, B)$, $R_2 = (A, C)$ обладает свойством соединения без потерь тогда и только тогда, когда выполняется многозначная зависимость $A \longrightarrow\!\!> B/C$.

Теорема Фейгана является более строгой версией теоремы Хита.

Теперь дадим определение четвертой нормальной формы (4НФ).

Четвертая нормальная форма схемы отношения – это либо данная схема, если она находится в УТНФ и все ее многозначные зависимости представляют собой функциональные зависимости от ключей, либо декомпозиция исходной схемы, каждая подсхема которой удовлетворяет этим же требованиям, а сама декомпозиция обладает свойством соединения без потерь.

Четвертая нормальная форма существует для любой схемы отношения.

4.3.5. Пятая нормальная форма схем отношений

Понятие пятой нормальной формы связано с понятием зависимости соединения. Рассмотрим следующий пример. Пусть имеется отношение Поставщик–Товар–Потребитель (ПТП), заданное следующей таблицей:

Поставщик	Товар	Потребитель
Ф 1	Товар 1	П 2
Ф 1	Товар 2	П 1
Ф 2	Товар 1	П 1
Ф 1	Товар 1	П 1

Проекции отношения на два атрибута имеют вид:

ПТ

Поставщик	Товар
Ф 1	Товар 1
Ф 1	Товар 2
Ф 2	Товар 1

ТП

Товар	Потребитель
Товар 1	П 2
Товар 2	П 1
Товар 1	П 1

ПП

Поставщик	Потребитель
Ф 1	П 2
Ф 1	П 1
Ф 2	П 1

Для естественных соединений получим:

ПТ $|><|$ ТП = ПТ $|><|$ ПП = ТП $|><|$ ПП, причем результат будет равен

Поставщик	Товар	Потребитель
Ф 1	Товар 1	П 2
Ф 1	Товар 2	П 1

Ф 2	Товар 1	П 1
Ф 2	Товар 1	П 2
Ф 1	Товар 1	П 1

Излишний кортеж

В то же время ПТ $|><|$ ТП $|><|$ ПП = ПТП.

Таким образом, естественное соединение любых двух проекций не дает исходного соединения. Вместе с тем естественное соединение трех проекций приводит к исходному отношению.

Подобное свойство может быть присуще не только отношению, но и схеме отношения и будет касаться произвольного числа проекций.

Пусть R – схема отношений на множестве атрибутов U, а A, B, …, Z – произвольные подмножества множества U. Схема R удовлетворяет ***зависимости соединения***

$$*\{A, B, \ldots, Z\}$$

тогда и только тогда, когда любое отношение со схемой R эквивалентно естественному соединению его проекций на A, B, …, Z.

Для зависимости соединения теорема Фейгина имеет следующую формулировку.

Схема R с множеством атрибутов $U = A \cup B \cup C$ удовлетворяет зависимости соединения $*\{AB, AC\}$ тогда и только тогда, когда она удовлетворяет многозначной зависимости $A \text{—>>} B/C$.

Формально получим следующее:

$A \text{—>>} B/C = *\{AB, AC\}$.

Из определения зависимости соединения следует, что из всех возможных форм это наиболее общая форма зависимости. То есть не существует более высокой степени зависимости, по отношению к которой зависимость соединения является всего лишь частным случаем.

На основании зависимости соединения дается определение пятой нормальной формы.

Схема отношения R находится в ***пятой нормальной форме*** (5НФ) тогда и только тогда, когда каждая нетривиальная зависимость соединения подразумевается ее ключами.

Зависимость соединения $*\{A, B, \ldots, Z\}$ называется тривиальной, если одна из проекций на A, B, …, Z совпадает с R.

Заданная зависимость соединения $*\{A, B, \ldots, Z\}$ *подразумевается ключами* тогда и только тогда, когда каждое подмножество атрибутов A, B, …, Z фактически является суперключом для схемы R. Таким образом, относительно заданной схемы отношения R можно утверждать, что она находится в 5НФ только при условии, что известны все ее ключи и все зависимости соединения, существующие в ней. Смысловое значение

зависимостей соединения, которые не являются одновременно многозначными и функциональными, далеко не очевидно. Следовательно, процедура определения того, что некоторая схема все еще находится в 4НФ, а не в 5НФ, и, таким образом, существует возможность ее дальнейшей декомпозиции, все еще остается не вполне ясной.

Как следует из определения, 5НФ является окончательной нормальной формой по отношению к операциям проекции и соединения. Таким образом, если схема находится в 5НФ, то гарантируется, что она не содержит аномалий, которые могут быть исключены посредством ее разбиения на проекции.

5. СЕМАНТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

5.1. ЦЕЛИ И СРЕДСТВА СЕМАНТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Первоначально в теории баз данных основное внимание уделялось средствам эффективной организации данных и манипулирования ими. В результате возникли три основные модели данных: иерархическая, сетевая и реляционная. Считалось, что предложенные средства достаточно универсальны для представления информации о любых предметных областях. Однако эти модели не содержат развитых средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в сознании проектировщика. Поэтому в последние годы получило развитие направление, возникшее в конце 1970-х – начале 1980-х гг. – семантическое, или концептуальное, моделирование в БД. Его основная цель – организация интерфейса проектировщика, а также конечного пользователя с информационной системой на уровне представлений о предметной области, а не на уровне структур данных. В результате строилась модель предметной области, не зависящая ни от конкретной СУБД, ни от технических средств. Интерес к этому направлению возрос также в связи с развитием средств автоматизированного проектирования БД на основе CASE-технологий.

В настоящее время определился основной подход к решению задач семантического моделирования в БД. Он заключается в выделении двух уровней моделирования.

1. Концептуального моделирования предметной области.
2. Моделирования собственно БД.

На первом уровне осуществляется переход от неформализованного описания предметной области и информационных потребностей конеч-

ного пользователя к их формальному выражению с помощью специальных языковых средств. На втором уровне происходит преобразование концептуальной модели предметной области в схему БД и нормализация схемы БД.

Первый уровень задачи семантического моделирования характеризуется четырьмя основными этапами.

1. Прежде всего, выявляется некоторое множество семантических концепций (понятий), которые могут быть полезны при неформальном обсуждении реального мира. Например, можно согласиться с тем, что мир построен из сущностей. Развивая данную концепцию, можно допустить, что сущности могут быть классифицированы по разным типам. Преимущество такой классификации заключается в том, что все сущности определенного типа будут обладать некоторыми общими свойствами. Более того, можно пойти еще дальше и согласиться с тем, что каждая сущность обладает неким особым свойством, предназначенным для ее идентификации, т. е. с тем, что каждая сущность обладает собственной идентичностью. Наконец, можно предположить, что каждая сущность соотносится с другими сущностями посредством некоторых связей.

2. Далее определяется набор соответствующих символических (формальных) объектов, которые могут использоваться для представления описанных выше семантических концепций.

3. Затем определяется набор формальных общих правил целостности, предназначенных для работы с такими формальными объектами.

4. Наконец также определяется набор формальных операторов, предназначенных для манипулирования этими формальными объектами.

К настоящему времени разработано много различных концепций для построения таких моделей. Одной из таких концепций является модель «сущность – связь», предложенная П. Ченом ещё в 1976 г., часто ее кратко называют ER-моделью. На различных разновидностях ER-модели основано большинство подходов к проектированию баз данных (главным образом, реляционных). Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных элементов.

5.2. МЕТОД «СУЩНОСТЬ-СВЯЗЬ»

Метод «сущность–связь» основан на использовании графических средств – ER-диаграмм. Поэтому метод называют также методом ER–диаграмм, от аббревиатуры ER – Entity (сущность) и Relationship (связь).

Основными понятиями метода «сущность–связь» являются: сущность, атрибут сущности, ключ сущности, тип сущности, связь между сущностями, степень связи, мощность связи, тип связи, степень участия сущности, диаграммы ER-экземпляров, ER-диаграмма.

Сущность представляет собой множество подобных индивидуальных объектов, информация о которых может храниться в базе данных. Экземпляры сущности индивидуальны, отличаются друг от друга и однозначно идентифицируются. Сущности должны иметь наименование с четким смысловым значением. Название сущности задается существительным в единственном числе, например: КЛИЕНТ, ЗАКАЗ, РАБОТНИК.

Атрибут представляет собой определенное свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так, атрибутами сущности РАБОТНИК могут быть его Табельный номер, Фамилия, Должность, Стаж и т. д.

Ключ сущности – атрибут или набор атрибутов, однозначно идентифицирующий экземпляр сущности. Понятие ключа сущности аналогично понятию ключа отношения.

Связь – это логическое соотношение между сущностями. Связь предполагает зависимость между атрибутами сущностей. Название связи представляется глаголом или глагольной фразой.

В качестве примера можно привести следующие связи: КЛИЕНТ РАЗМЕЩАЕТ ЗАКАЗ (Иванов РАЗМЕЩАЕТ Ремонт компьютера), РАБОТНИК ВЫПОЛНЯЕТ ЗАКАЗ (Петров ВЫПОЛНЯЕТ Ремонт компьютера).

Степень связи указывает количество сущностей, участвующих в связи.

Мощность связи служит для указания отношения соответствия числа экземпляров сущностей, участвующих в связи.

Наиболее распространенными являются бинарные (степень два) связи с мощностями «один к одному», «один ко многим», «многие ко многим».

По типу сущность может быть *независимой* и *зависимой*. В первом случае существование сущности не зависит от существования другой сущности, во втором – зависит. Это означает, что появление экземпляра зависимой (дочерней) сущности возможно только при существовании соответствующего экземпляра основной (родительской) сущности. Например, сущность ЗАКАЗ является зависимой от сущности КЛИЕНТ.

Связи делятся на *идентифицирующие* и *неидентифицирующие*. Идентифицирующая связь устанавливается между независимой сущностью и

зависимой сущностью. Неидентифицирующая связь – между независимыми сущностями.

Степень участия сущности отражает факт обязательности участия экземпляров сущности в связи.

Степень участия является *обязательной*, если все экземпляры сущности участвуют в рассматриваемой связи, в противном случае степень участия сущности является *необязательной*. В частности, в идентифицирующей связи для дочерней сущности степень участия всегда будет обязательной.

Мощность связи и степени участия сущностей при проектировании БД определяются спецификой предметной области.

Для определения мощности связи, степени участия сущностей, полезно использовать диаграммы *ER-экземпляров*. Диаграмма показывает соответствие экземпляров сущностей.

На рис. 1 приведена диаграмма ER-экземпляров для сущностей РАБОТНИК и ЗАКАЗ со связью ВЫПОЛНЯЕТ.

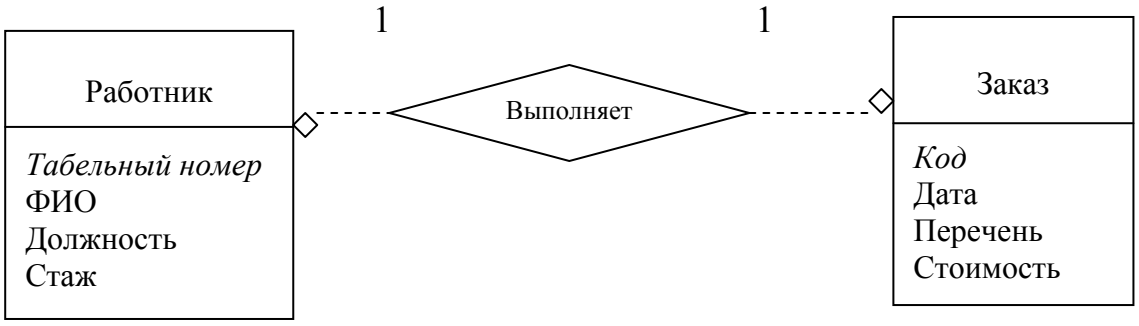
РАБОТНИК	ВЫПОЛНЯЕТ	ЗАКАЗ
Иванов •		• Ремонт монитора
Петрович •		• Замена диска
Кузьмин •		• Установка ОС
Еленин •		• Добавление планок памяти
Козловский •		• Ремонт вентилятора

Рис. 1. Диаграмма ER-экземпляров

Диаграмма показывает, какой конкретно заказ выполняет работник.

На *ER-диаграмме* изображаются сущности со своими именами и при необходимости атрибутами с выделением ключевых атрибутов. Сущность изображается прямоугольником, причем зависимая со скругленными углами. Связь указывается линией с именем в ромбе: идентифицирующая – сплошной линией, неидентифицирующая – пунктирной. Необязательная степень участия обозначается прозрачным ромбом, обязательная – жирной точкой. Мощность связи указывается числами над линией. На рис. 2 представлена ER-диаграмма, соответствующая приведенной диаграмме ER-экземпляров. Рис. 2б. дает диаграмму с указанием атрибутов и выделением курсивом ключевых атрибутов.

а)



б)

Рис. 2. ER-диаграмма

В зависимости от степени участия и типов связи могут возникать различные варианты ER-диаграмм. Так, приведенный выше вариант на рис.1, 2 дает пример неидентифицирующей связи мощности 1:1 с необязательными степенями участия для обеих сущностей.

Если выдвинуть требование обеспечения всех заказов исполнителями, с условием закрепления за каждой заказом только одного из них, то получим неидентифицирующую связь с мощностью 1: М, обязательной степенью участия для сущности ЗАКАЗ и необязательной степенью участия для сущности РАБОТНИК. Диаграмма ER-экземпляров и ER-диаграмма для этого случая приведены на рис. 3.

РАБОТНИК	ВЫПОЛНЯЕТ	ЗАКАЗ
Иванов •		• Ремонт монитора
Петрович •		• Замена диска
Кузьмин •		• Установка ОС
Еленин •		• Добавление планок памяти
Козловский •		• Ремонт вентилятора

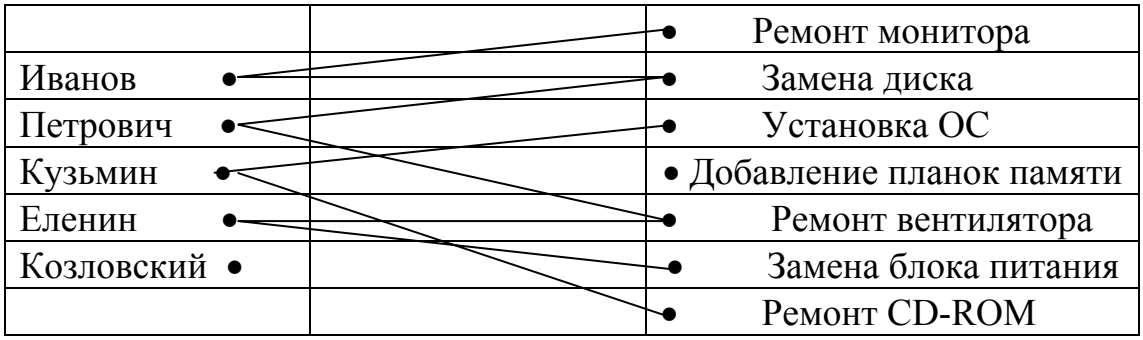
а)

б)

Рис. 3. Диаграммы для связи 1: М и вариантом степеней участия Н-О:

а) ER-экземпляров

РАБОТНИК	ВЫПОЛНЯЕТ	ЗАКАЗ

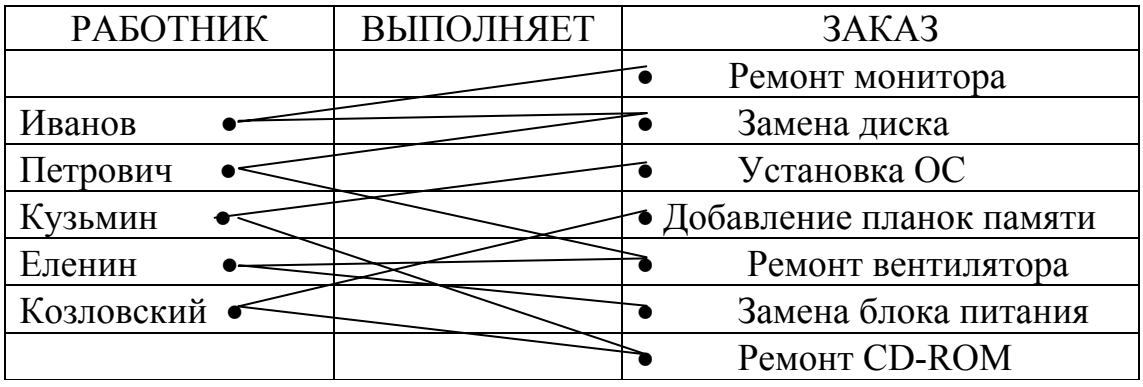


б) ER-диаграмма

Рис. 4. Диаграммы для связи M:M и вариантом степеней участия H-O

Для выполнения сложных заказов может потребоваться привлечение нескольких работников. Тогда при условии закрепления каждого заказа за работниками получим неидентифицирующую связь мощностью M : M с необязательной степенью участия сущности РАБОТНИК и обязательной степенью участия сущности ЗАКАЗ. Этому случаю соответствуют диаграммы на рис. 4.

Допустим, что каждый работник привлекается к выполнению не менее одного заказа, а каждый заказ выполняется не менее чем одним работником. Соответствующие этому случаю диаграммы приведены на рис. 5.



а)

Рис .5. Диаграммы для связи M : M и вариантом степеней участия O-O

Приведем пример идентифицирующей связи. На рис. 6 приведена диаграмма ER-экземпляров для сущностей КЛИЕНТ и ЗАКАЗ со связью РАЗМЕЩАЕТ.

Отметим, что в результате проектирования могут быть получены несколько вариантов ER-диаграмм. Разные проектировщики, рассматривая проблему каждый со своей точки зрения, могут получить различные наборы сущностей и связей. При этом все получаемые варианты могут быть рабочими, и выбор лучшего из них будет результатом дополнительного анализа.

Выявление сущностей и связей между ними, а также формирование на их основе ER-диаграмм выполняется на начальном этапе. Затем, на основе анализа ER-диаграмм формируются отношения проектируемой базы данных.

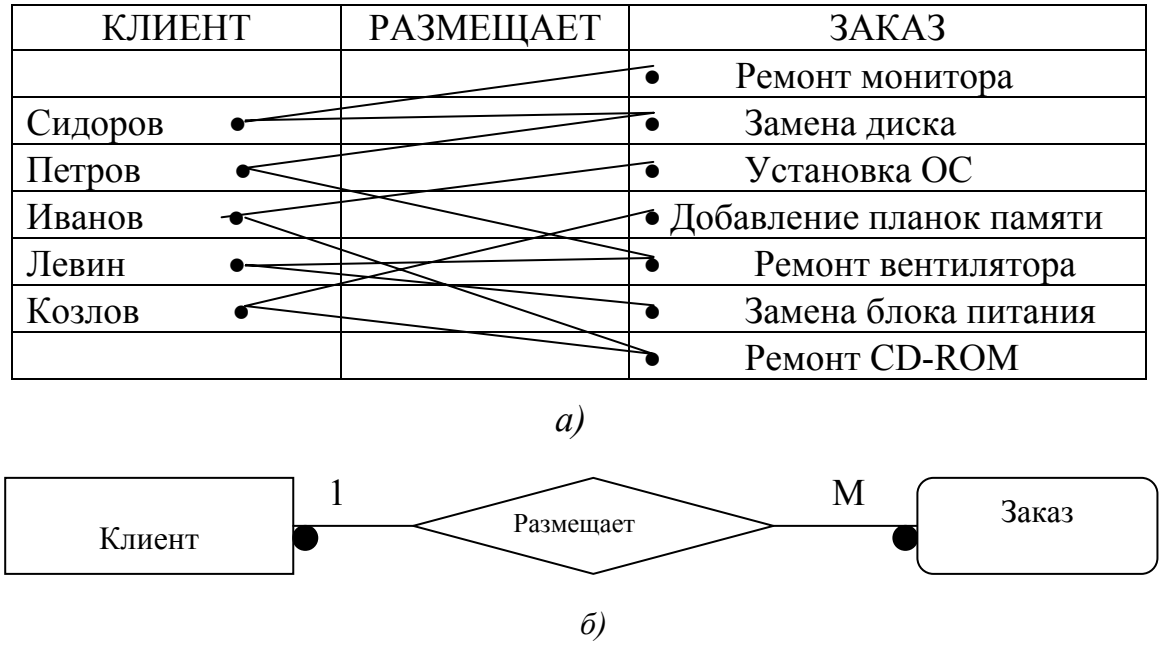


Рис. 6. Диаграммы для идентифицирующей связи

5. 3. ЭТАПЫ МОДЕЛИРОВАНИЯ

Процесс моделирования базы данных является итерационным – допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений. Он включает ряд этапов.

1. Выделение сущностей и связей между ними.
2. Построение диаграмм ER-типа для всех сущностей и их связей.

3. Формирование с использованием диаграмм ER-типа набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения.

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 предварительных отношениях.

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

Одним из узловых этапов моделирования является этап формирования отношений. Рассмотрим процесс формирования предварительных отношений, составляющих первичный вариант схемы базы данных.

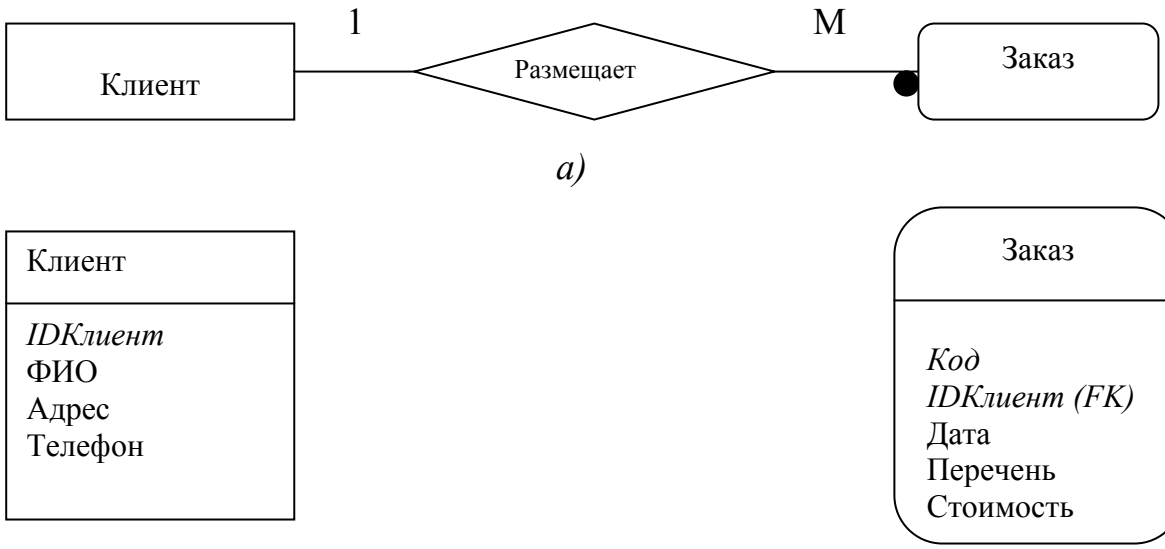
В рассмотренных выше примерах связь РАЗМЕЩАЕТ всегда соединяет две сущности и поэтому является *бинарной*. Рассмотренные ниже правила формирования отношений по диаграммам ER-типа распространяются именно на бинарные связи. Поэтому, когда речь идет о связях, слово «бинарные» далее опускается.

5.4. ПРАВИЛА ФОРМИРОВАНИЯ ОТНОШЕНИЙ.

Правила формирования отношений по ER-диаграммам основываются на учете типа связи и мощности связи.

Формирование отношений для идентифицирующей связи. Если имеется идентифицирующая связь, то для каждой сущности формируется отношение с первичным ключом, являющимся ключом соответствующей сущности. Затем в отношение, сущность которого является дочерней, добавляются в первичный ключ атрибуты первичного ключа родительской сущности. В дочерней сущности новые атрибуты помечаются как *внешний ключ* (*FK*).Эта операция дополнения атрибутов называется *миграцией атрибутов*. Атрибуты первичного ключа в отношении, отвечающего дочерней сущности, получают признак NOT NULL, что означает невозможность появления кортежа без наличия соответствующего кортежа в родительском отношении. Степень участия при этом родительской сущности не имеет значения.

На рис. 7 приведены ER-диаграмма и отношения, сформированные на ее основе для идентифицирующей связи.



а)

б)

Рис. 7. ER-диаграмма и отношения для идентифицирующей связи

Формирование отношений для неидентифицирующей связи. Для неидентифицирующей связи формирование отношений зависит от мощности связи. Возможны два варианта.

Если мощность связи 1:1 или 1: М, то формируются два отношения соответствующие сущностям. Затем из отношения с мощностью связи 1 атрибуты первичного ключа мигрируют в неключевые атрибуты отношения с мощностью связи 1 или М. При этом они получают статус внешнего ключа. Степени участия сущностей не учитываются. На рис. 8 дается пример ER-диаграммы и отношений для неидентифицирующей связи.

Если мощность связи М : М, то формируются три отношения. Два из них соответствуют сущностям связи, третье содержит атрибуты первичных ключей обеих сущностей. Каждый из ключей приобретает статус внешнего ключа к соответствующему отношению. Все атрибуты третьего отношения образуют первичный ключ (рис. 9). При этом кортеж в третьем отношении может появиться только тогда, когда существуют соответствующие кортежи в первом и втором отношениях.

Рис. 8. ER-диаграмма и отношения для неидентифицирующей связи и мощностью связи 1: M

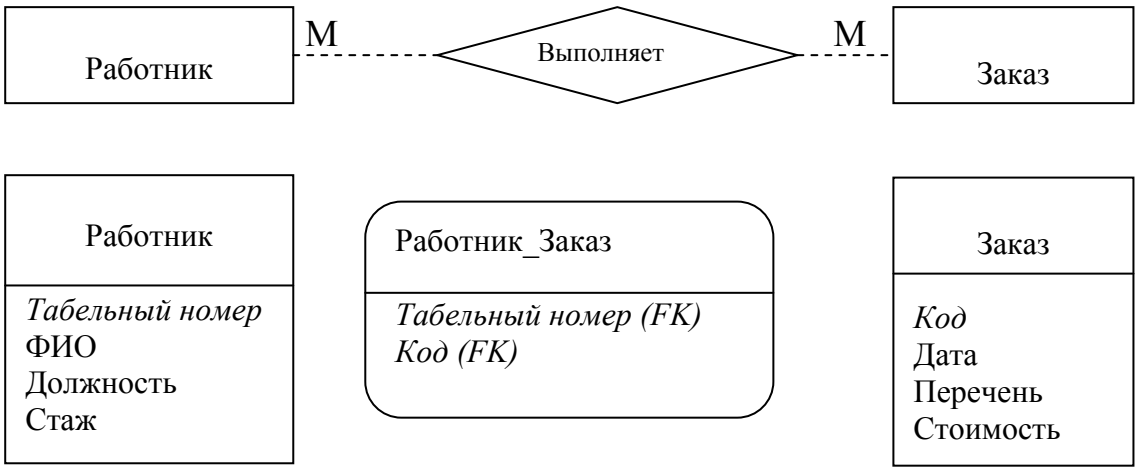


Рис. 9. ER-диаграмма и отношения для неидентифицирующей связи и мощностью связи M : M

6. СТРУКТУРА И ОСНОВНЫЕ ФУНКЦИИ СУБД

6.1. ТИПОВАЯ ОРГАНИЗАЦИЯ СОВРЕМЕННОЙ СУБД

Организация и состав компонентов типичной СУБД соответствуют выполняемому ею набору функций. Выделяют следующие основные функции СУБД:

1) поддержка языков БД;
2) управление данными во внешней памяти;
3) управление буферами оперативной памяти;
4) управление транзакциями;
5) журнализация и восстановление БД после сбоев.

Более подробно эти функции будут рассмотрены далее.

Логически в современной реляционной СУБД можно выделить внутреннюю часть – ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других – нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры «клиент–сервер» ядро является основной составляющей серверной части системы.

Основной функцией *компилятора языка БД* является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т. е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия. Поэтому компилятор должен решить, каким образом выполнять оператор языка, прежде чем произвести программу. Применяются достаточно сложные методы оптимизации операторов. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением *подсистемы поддержки времени выполнения*, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные *утилиты БД* обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т. д. Утилиты программируются с использованием ин-

терфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

Рассмотрим подробнее функции СУБД.

6.2. ФУНКЦИИ СУБД

Поддержка языков БД. Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков, из которых чаще всего использовались два языка – язык определения схемы БД (SDL – Schema Definition Language) и язык манипулирования данными (DML – Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т. е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т. е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language), обеспечивающий доступ к данным, их модификацию, определение структуры и другие операции. Этот язык имеет сравнительно небольшое число операторов и относительно простой синтаксис, приближенный к английскому языку. Формулируя запросы на языке SQL, можно создавать и модифицировать различные объекты базы данных и, оперируя группами строк, вставлять, выбирать, обновлять и удалять данные из таблиц. Кроме того, он позволяет управлять доступом к базе данных и ее объектам и обеспечивать непротиворечивость и целостность данных, хранящихся в базе.

Язык SQL сочетает средства SDL и DML, т. е. дает возможность определять схему реляционной базы данных и манипулировать данными. При этом именование объектов базы данных (для реляционной БД – именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специ-

альных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т. е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Наконец, авторизация доступа к объектам базы данных производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Управление данными во внешней памяти. Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например для ускорения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. В развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

Управление буферами оперативной памяти. СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных производится обмен с внешней памятью, то вся система работает со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной их замены. Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предпо-

ложении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока работа находится в стадии исследований.

Управление транзакциями. ***Транзакция*** – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Таким образом, существование механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

При управлении транзакциями в многопользовательской СУБД необходимо добиться такого порядка их выполнения (*сериализации*), при котором суммарный эффект смеси транзакций был бы эквивалентен эффекту их некоторого последовательного выполнения. Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей. Более подробно этот вопрос будет рассмотрен далее.

Журнализация и восстановление после сбоев. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая исполь-

зуется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал – это особая часть БД, куда поступают записи обо всех изменениях основной части БД. Журнал не доступен пользователям СУБД и поддерживается с особой тщательностью (иногда поддерживаются две копии журнала, расположенные на отдельных физических дисках). Изменения БД заносятся в журнал на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда – минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии «упреждающей» записи в журнал (так называемого протокола Write Ahead Log – WAL). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Для восстановления БД после жесткого сбоя используют не только журнал, но и архивную копию БД. ***Архивная копия*** – это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Восстановление БД состоит в том, что исходя из архивной копии, по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя.

7. УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

7.1. СВОЙСТВА ТРАНЗАКЦИЙ. ПРОБЛЕМЫ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ

Транзакция – это действие или серия действий, которые осуществляют доступ или изменение содержимого базы данных и рассматриваются СУБД как единое целое. Транзакция является логической единицей работы, выполняемой в БД. Она может быть представлена отдельной программой, являться частью алгоритма программы или даже отдельной командой (например, командой INSERT или UPDATE) и включать произвольное количество операций, выполняемых в данных.

Любая транзакция всегда должна переводить БД из одного согласованного состояния в другое, хотя допускается, что согласованность со-

стояния базы будет нарушаться в ходе выполнения транзакции. Любая транзакция завершается одним из двух возможных способов. В случае успешного завершения результаты транзакции *фиксируются* (COMMIT) в БД, и последняя переходит в новое согласованное состояние. Если выполнение транзакции не увенчалось успехом, она *отменяется*. В этом случае в базе данных должно быть восстановлено то согласованное состояние, в котором она находилась до начала данной транзакции. Этот процесс называется *откатом* (ROLLBACK) транзакции. Реализация в СУБД принципа сохранения промежуточных состояний, подтверждения или отката транзакций обеспечивается специальным механизмом, для поддержки которого создается некоторая системная структура, называемая *журналом транзакций*. Зафиксированная транзакция не может быть отменена.

В большинстве языков манипулирования данными для указания границ отдельных транзакций используются операторы BEGIN TRANSACTION, COMMIT и ROLLBACK (или их эквиваленты). Если эти ограничители не были использованы, вся выполняемая программа расценивается как единая транзакция. СУБД автоматически выполнит команду COMMIT при нормальном завершении этой программы. Аналогично, в случае ее аварийного завершения в БД автоматически будет выполнена команда ROLLBACK.

Существуют некоторые свойства, которыми должна обладать любая из транзакций.

1. *Атомарность.* Это свойство типа «все или ничего». Любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вовсе.

2. *Согласованность.* Каждая транзакция должна переводить базу данных из одного согласованного состояния в другое согласованное состояние.

3. *Изолированность.* Все транзакции выполняются независимо одна от другой. Иначе говоря, промежуточные результаты незавершенной транзакции не должны быть доступны другим транзакциям.

4. *Долговечность.* Результаты успешно завершенной (зафиксированной) транзакции должны сохраняться в базе данных постоянно и не должны быть утеряны в результате последующих сбоев.

Важнейшей целью создания баз данных является *организация параллельного доступа* многих пользователей к общим данным, используемым ими совместно. Обеспечить параллельный доступ относительно несложно, если все пользователи только читают данные, помещенные в базу. В этом случае работа каждого из них не оказывает никакого влияния на ра-

боту остальных пользователей. Однако если два или больше пользователя одновременно обращаются к БД и хотя бы один из них имеет целью обновить хранимую в базе информацию, возможно взаимное влияние процессов друг на друга, способное привести к несогласованности данных.

Рассмотрим три потенциальные проблемы, которые могут иметь место при параллельном выполнении транзакций: проблему потерянного обновления, проблему зависимости от нефиксированных результатов и проблему несогласованной обработки. Если обозначить операции чтения, записи новых значений и откат транзакции над БД соответственно как R, W, O, то для первой транзакции это могут быть операции R_1, W_1, O_1, а для параллельно выполняющейся второй транзакции операции R_2, W_2, O_2.

Проблема потерянного обновления возникает, когда результаты вполне успешно завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Эта проблема может возникнуть в результате выполнения последовательности операций R_1, R_2, W_2, W_1.

Проблема зависимости от нефиксированных результатов возникает в том случае, если одна из транзакций получит доступ к промежуточным результатам выполнения другой транзакции до того, как они будут зафиксированы в базе данных. Может возникнуть в результате последовательности операций R_1, W_1, R_2, O_1, W_2.

В обоих приведенных выше примерах речь шла о транзакциях, выполняющих обновление данных в базе, наличие взаимовлияния между которыми и вызывало разрушение базы. Однако транзакции, которые только считывают информацию из БД, также могут давать неверные результаты, если им будут доступны для чтения промежуточные результаты одновременно выполняющихся и еще не завершенных транзакций, обновляющих информацию в базе. Возникает проблема *несогласованной обработки*. В некоторых случаях ее называют *чтением мусора* или *неповторяемостью чтения*. Проблема может возникнуть в результате выполнения последовательности операторов R_1, R_2, W_1, R_2. В данном случае второе R_2 означает считывание следующей порции данных.

С управлением параллельностью или транзакциями в многопользовательской СУБД связаны важные понятия сериализации транзакций и сериального плана выполнения смеси транзакций. Под *сериализацией* параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения.

Сериальный план выполнения смеси транзакций – это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существуют два основных метода управления параллельностью, позволяющих организовать безопасное одновременное выполнение транзакций при соблюдении определенных ограничений: *метод блокировки* и *метод временных меток*.

По сути, и блокировка, и использование временных меток являются *консервативными*, или *пессимистическими*, подходами, поскольку они откладывают выполнение транзакций, способных в будущем в тот или иной момент времени войти в конфликт с другими транзакциями. *Оптимистические* методы, строятся на предположении, что вероятность конфликта невысока, поэтому они допускают асинхронное выполнение транзакций, а проверка на наличие конфликта откладывается на момент их завершения и фиксации в БД.

7.2. МЕТОДЫ УПРАВЛЕНИЯ ТРАНЗАКЦИЯМИ

Консервативные методы. ***Блокировка*** – процедура, используемая для управления параллельным доступом к данным. Когда некоторая транзакция получает доступ к БД, механизм блокировки позволяет (с целью исключения получения некорректных результатов) отклонить попытки получения доступа к этим же данным со стороны других транзакций.

Именно методы блокировки чаще всего используются на практике для обеспечения упорядоченности параллельно выполняемых транзакций. Существует несколько различных вариантов этого механизма, однако все они построены на одном и том же фундаментальном принципе: транзакция должна потребовать выполнить блокировку *для чтения* (разделяемую) или *для записи* (эксклюзивную) некоторого элемента данных перед тем, как она сможет выполнить в БД соответствующую операцию чтения или записи. Установленный *блок* препятствует модификации элемента данных другими транзакциями или даже считыванию его, если этот блок был установлен для записи. Блокировка может быть выполнена для элементов самого различного размера – начиная с БД в целом и заканчивая

отдельным полем конкретной записи. Размер блокируемого элемента задается *уровнем детализации* устанавливаемого блока

Если транзакция установила блокировку элемента данных *для чтения*, она сможет считать его, но не сможет обновить.

Если транзакция установила блокировку элемента данных *для записи*, она может как читать, так и обновлять этот элемент.

Поскольку операция чтения не может служить причиной конфликта, допускается устанавливать блокировку для чтения одного и того же элемента одновременно со стороны сразу нескольких транзакций. В то же время блокировка элемента для записи предоставляет транзакции эксклюзивное право доступа к нему. Следовательно, до тех пор пока транзакция будет удерживать некоторый элемент заблокированным для записи, никакая другая транзакция не сможет ни считать, ни обновить его. Для обеспечения упорядоченности следует использовать дополнительный протокол, определяющий моменты установки и снятия блокировки для каждой из транзакций. Самым известным из таких протоколов является метод двухфазной блокировки.

Двухфазная блокировка – это выполнение транзакции по протоколу, при котором все операции блокирования предшествуют первой операции разблокирования.

В соответствии с основным правилом этого протокола каждая транзакция может быть разделена на две фазы: *фазу нарастания,* в которой выполняются все необходимые блокировки и не освобождается ни один из элементов данных, и *фазу сжатия,* в которой освобождаются все выполненные ранее блокировки и не может быть затребовано ни одной новой. Нет никакой необходимости в том, чтобы все требуемые блокировки были установлены одновременно. Как правило, транзакция устанавливает некоторые блокировки, выполняет определенную обработку, после чего может затребовать установку дополнительных необходимых ей блокировок. Однако она не может освободить ни один из блоков, пока не достигнет той стадии, на которой ей уже не потребуется установка новых блокировок.

Есть одна проблема, связанная с двухфазной блокировкой, которая может иметь место при любых схемах освобождения заблокированных элементов. Эта проблема носит название *взаимной блокировки* и является следствием того факта, что любая транзакция может быть переведена в состояние ожидания освобождения необходимого элемента данных. Если две транзакции будут ожидать освобождения элементов, заблокированных другой транзакцией из этой же пары, то возникнет состояние взаимной блокировки. Для исключения самоблокировок может использоваться

система приоритетов, в которой приоритет транзакции тем выше, чем дольше она находится в состоянии ожидания. Альтернативным вариантом является использование для ожидающих транзакций очереди, построенной по схеме «первым пришел, первым обслуживается». Однако чаще всего прибегают к самому простому варианту: выбирают в качестве жертвы одну из транзакций и просто ее откатывают. Эти действия, несомненно, могут повлиять на эффективность работы системы.

Один из возможных подходов *предупреждения взаимных блокировок* состоит в установлении порядка выполнения транзакций на основе использования временных отметок, о чем будет сказано ниже. Были предложены два возможных алгоритма. Первый алгоритм, получивший название «ожидание – отмена», требует, чтобы более старые транзакции ожидали завершения более новых. В противном случае транзакция отменяется и перезапускается с той же временной отметкой. Однако рано или поздно она станет самой старой из активных транзакций и уже не будет отменена. Второй алгоритм, «отмена – ожидание», использует диаметрально противоположный подход: только более новые могут ожидать завершения более старой транзакции. Если более старая транзакция потребует выполнения блокировки элемента данных, уже заблокированного более новой транзакцией, последняя будет отменена. Иногда для предупреждения тупиков система строит граф выполнения транзакций и пытается заранее определить тупиковые ситуации.

Методы временных отметок *(timestamp)*, применяемые для контроля целостности данных, не требуется использование каких-либо блокировок, и, следовательно, исключается возможность возникновения взаимных блокировок процессов. Методы блокировки обычно устраняют возможные конфликты посредством перевода транзакций в состояние ожидания. Методы с использованием временных отметок не предусматривают какого-либо ожидания: вовлеченные в конфликт транзакции просто отменяются, после чего запускаются заново.

Временная отметка – это уникальный идентификатор, создаваемый СУБД с целью обозначения относительного момента времени запуска транзакции.

Временная отметка может быть создана с использованием системных часов для фиксации момента запуска транзакции либо посредством увеличения значения некоторого логического счетчика при каждом запуске очередной транзакции.

Составляется протокол управления параллельностью, основная цель которого состоит в установлении глобальной очередности выполнения транзакций, при которой более старые транзакции (с меньшим значением

временной отметки) имеют больший приоритет при разрешении возникающих конфликтов.

При использовании протокола временных отметок, когда транзакция предпринимает попытку чтения или записи элемента данных, операция чтения или записи выполняется только в том случае, если последнее обновление требуемого элемента данных было выполнено более старой транзакцией. В противном случае транзакция, запросившая операцию чтения или записи, отменяется и перезапускается с присвоением ей новой временной отметки. Новая временная отметка должна быть присвоена перезапускаемой транзакции для того, чтобы предотвратить ее попадание в цикл постоянной отмены и перезапуска. Без получения новой временной отметки транзакция с более старой временной отметкой не сможет завершить свою работу, поскольку более новая транзакция уже успела зафиксировать свои результаты в БД.

Оптимистические методы управления. В некоторых типах вычислительных систем конфликты между транзакциями происходят очень редко, поэтому дополнительная обработка, вызванная поддержкой протоколов с блокировкой или с использованием временных меток, оказывается совершенно излишней для большей части транзакций. Оптимистические технологии основываются на предположении, что конфликты в системе возможны нечасто, поэтому эффективнее будет организовать выполнение транзакций, исключив все задержки, связанные с достижением гарантированной упорядоченности. Перед завершением работы транзакции выполняется проверка, определяющая, имел ли место конфликт. Если это так, транзакция откатывается и перезапускается. Поскольку исходно утверждается, что конфликты в данной системе – явление нечастое, то и откатов потребуется выполнять немного. Дополнительная нагрузка, связанная с перезапуском некоторых транзакций, может оказаться довольно значительной, поскольку, по сути, она будет связана с повторным выполнением всей транзакции в целом. Поэтому применение данной схемы имеет смысл только в том случае, если откаты будут происходить достаточно редко, а большая часть транзакций в системе будет выполняться без каких-либо дополнительных задержек. Подобная технология потенциально позволяет достичь существенно более высокого уровня параллельности по сравнению с традиционными протоколами, поскольку не требует использования механизма блокировок.

7.3. УРОВЕНЬ ДЕТАЛИЗАЦИИ БЛОКИРУЕМЫХ ЭЛЕМЕНТОВ ДАННЫХ

Уровень детализации – это размер элементов данных, выбранных в качестве ***защищаемой единицы*** для протокола управления параллельностью.

Во всех обсуждавшихся выше протоколах управления параллельностью предполагалось, что БД состоит из некоторого количества «элементов данных», причем дополнительно не уточнялось, что означает этот термин. Как правило, в качестве элемента данных выбирается один из перечисленных ниже объектов, размеры которых варьируются от очень крупных до мельчайших:

- вся база данных;
- отдельный файл;
- отдельная страница данных (иногда называемая областью или блоком – сектор на физическом диске, используемом для хранения таблиц);
- отдельная запись;
- отдельное поле в записи.

Размер, или уровень детализации, элемента данных, который может быть заблокирован отдельной операцией транзакции, оказывает сильнейшее влияние на общую производительность системы и эффективность работы протокола управления параллельностью, что следует учитывать при выборе размера элемента данных в системе. Было предложено несколько методов динамической установки размера блокируемого элемента данных. В соответствии с этими методами размер элемента данных зависит от типа выполняемых в данный момент транзакций и устанавливается из соображений оптимального соответствия их требованиям. В идеале СУБД должна поддерживать смешанный уровень детализации, позволяющий блокировать отдельные записи, страницы и целые файлы. Некоторые системы автоматически повышают уровень детализации от отдельных записей или страниц до уровня файла, если определенная транзакция блокирует больше установленного процента записей или страниц некоторого файла.

8. ВОССТАНОВЛЕНИЕ БД ПОСЛЕ СБОЕВ

8.1. ОСНОВНЫЕ ПРИНЦИПЫ ВОССТАНОВЛЕНИЯ

При возникновении любого аппаратного или программного сбоя СУБД должна восстановить последнее согласованное состояние БД.

Основные принципы восстановления заключаются в следующем:

1) результаты зафиксированных транзакций должны быть сохранены

в восстановленном состоянии БД;

2) результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии БД.

Указанные принципы в типичной СУБД реализуются с помощью:

1) механизма резервного копирования, предназначенного для периодического создания копий БД;

2) средства ведения журнала, в котором фиксируются текущее состояние транзакций и вносимые в БД изменения;

3) функции создания контрольных точек, обеспечивающей перенос выполняемых в БД изменений во вторичную память с целью сделать их постоянными;

4) менеджера восстановления, обеспечивающего восстановление согласованного состояния БД, нарушенного в результате отказа.

8.2. МЕХАНИЗМ РЕЗЕРВНОГО КОПИРОВАНИЯ.

Любая СУБД должна предоставлять механизм, позволяющий создавать резервные копии БД и ее файла журнала через установленные интервалы и без необходимости останавливать систему. Резервная копия БД используется в случае повреждения или разрушения файлов БД во вторичной памяти. Резервное копирование может выполняться для БД в целом или в инкрементном режиме. В последнем случае в копию помещаются сведения только об изменениях, накопившихся с момента создания полной предыдущей или инкрементной копии системы. Как правило, резервные копии создаются на автономных носителях. Для фиксации хода выполнения транзакций в базе данных СУБД использует специальный файл, который называют *журналом.* Он содержит сведения обо всех обновлениях, выполненных в базе данных. В файл журнала может помещаться следующая информация:

1. Записи о транзакциях.
2. Записи контрольных точек.

Записи о транзакциях включают:

1) идентификатор транзакции;

2) тип записи журнала (начало транзакции, операции вставки, обновления или удаления, отмена или фиксация транзакции);

3) идентификатор элемента данных, вовлеченного в операцию обработки БД (операции вставки, удаления и обновления);

4) копию элемента данных до операции, т. е. его значение до изменения (только операции обновления и удаления);

5) копию элемента данных после операции, т. е. его значение после изменения (только для операций обновления и вставки);

6) служебную информацию файла журнала, включающую указатели на предыдущую и следующую записи журнала для этой транзакции (любые операции);

Один из подходов к автономной обработке файла журнала состоит в разделе оперативного файла журнала на две независимые части, организованные в виде записей с произвольным доступом. Записи журнала помещаются в первый файл до тех пор, пока он не оказывается заполненным до установленного уровня (например, на 70 %). Затем открывается второй файл, и все записи журнала для новых транзакций записываются уже в него. Сведения о старых транзакциях помещаются в первый файл до пор, пока обработка всех старых транзакций не будет завершена. В этот момент первый файл закрывается и переводится в автономное состояние. Подобный подход упрощает восстановление отдельных транзакций, поскольку записи о каждой отдельной транзакции всегда содержатся в одном фрагменте файла журнала – либо в оперативном, либо в автономном. Следует отметить, что скорость записи информации в файл журнала может оказаться одним из важнейших факторов, определяющих общую производительность системы с БД.

8.2.1. СОЗДАНИЕ КОНТРОЛЬНЫХ ТОЧЕК

Помещаемая в файл журнала информация предназначена для использования в процессе восстановления системы после отказа. Одно из основных затруднений в этой схеме состоит в том, что когда происходит отказ, может отсутствовать какая-либо информация о том, насколько далеко назад следует «откатиться» в файле журнала, чтобы начать повторный прогон уже завершенных транзакций. В результате может оказаться, что повторный прогон будет выполнен для тех транзакций, которые уже были окончательно зафиксированы в БД. Для ограничения объема поиска в файле журнала используется технология создания контрольных точек.

Контрольная точка – момент синхронизации между базой данных и журналом транзакций. Все буфера системы принудительно записываются во вторичную память системы.

Контрольные точки организуются через установленный интервал времени и включают выполнение следующих действий:

1) запись всех имеющихся в оперативной памяти записей журнала во вторичную память;

2) запись всех модифицированных блоков в буферах БД во вторичную память;

3) помещение в файл журнала записи контрольной точки. Эта запись содержит идентификаторы всех транзакций, которые были активны в момент создания этой контрольной точки.

Если транзакции выполняются последовательно, то после возникновения отказа файл журнала просматривается с целью обнаружения последней из транзакций, начавших свою работу до момента создания последней контрольной точки. Любая более ранняя транзакция будет зафиксирована в БД. Это значит, что ее изменения были перенесены на диск в момент создания последней контрольной точки. Следовательно, прогону подлежит только транзакция, которая была активна в момент создания контрольной точки, а также все прочие транзакции, которые начали свою работу позже и для которых в журнале присутствуют записи как начала, так и завершения. Та транзакция, которая была активна в момент отказа, должна быть отменена. В случае, если транзакции выполняются в системе параллельно, потребуется повторный прогон всех транзакций, которые завершили свою работу с момента создания контрольной точки, и выполнение отката всех транзакций, которые были активны в момент отказа.

8.3. МЕТОДЫ ВОССТАНОВЛЕНИЯ.

Тип процедуры, которая будет использована для восстановления БД, зависит от размера повреждений, которые были получены базой в результате отказа. Рассмотрим два варианта.

Если БД получила обширные повреждения, например разрушилась магнитная головка диска, то потребуется восстановить ее последнюю резервную копию, после чего повторить в ней все выполненные транзакции, сведения о которых присутствуют в журнале регистрации. Безусловно, предполагается, что файл журнала поврежден не был. Рекомендуется во всех случаях, когда это возможно, файл журнала создавать на дисковых носителях, отличных от тех, на которых размещены основные файлы БД. Подобное решение снижает риск одновременной потери, как файлов БД, так и файла ее журнала.

Если БД не получила физических повреждений, но лишь утратила согласованность размещенных в ней данных, например из-за аварийного останова системы в процессе обработки транзакций, то достаточно будет отменить те изменения, которые вызвали переход БД в несогласованное состояние. Кроме того, возможно потребуется повторно прогнать некоторые транзакции, чтобы быть уверенным в том, что внесенные в них изменения действительно зафиксированы во вторичной памяти. В данном случае нет необходимости обращаться к резервной копии базы дан-

ных, поскольку вернуть базу в согласованное состояние можно с помощью информации о содержимом полей до и после модификации, сохраняемой в файле журнала.

Рассмотрим подробно два метода восстановления, которые могут быть применены в последнем из указанных выше случаев – т. е. когда БД не была полностью разрушена, но лишь утратила согласованное состояние. Метод отложенного обновления и метод немедленного обновления отличаются друг от друга способом внесения обновлений во вторичную память. Кроме того, мы познакомимся с альтернативным методом теневых страниц.

Метод отложенного обновления. При использовании этого протокола обновления не заносятся в БД тех пор, пока транзакция не выдаст команду фиксации выполненных изменений. Если выполнение транзакции будет прекращено до достижения этой точки, никаких изменений в БД выполнено не будет, поэтому не потребуется и их отмена. Однако в таком случае может потребоваться повторный прогон уже завершившихся транзакций, поскольку их результаты могли еще не достичь вторичной памяти. При применении данного метода файл журнала используется с целью восстановления следующим образом.

1. При запуске транзакции в журнал помещается запись: *Начало транзакции*.

2. При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные (за исключением значения элементов данных до обновления). Реально запись изменений в буфере СУБД или саму БД не производится.

3. Когда транзакция достигает своей конечной точки, в журнал помещается запись: *Транзакция завершена.* Все записи журнала по данной транзакции выводятся на диск, после чего выполняется фиксация внесенных транзакцией изменений. Для внесения действительных изменений в БД используется информация, помещенная в файл журнала.

4. В случае отмены выполнения транзакции записи журнала по данной транзакции аннулируются и не выводятся на диск.

Обратите внимание на то, что записи журнала по выполненной транзакции выводятся на диск до того, как ее результаты будут зафиксированы. Поэтому, если отказ БД произойдет в процессе действительного выполнения обновлений в БД, помещенные в журнал сведения сохранятся и требуемые обновления можно будет выполнить позже. В случае отказа файл журнала анализируется с целью выявления транзакций, которые находились в процессе выполнения в момент отказа, начиная с послед-

ней строки. Файл журнала просматривается в обратном направлении вплоть до записи о последней выполненной контрольной точке.

5. Любые транзакции, для которых в файле журнала присутствуют записи *Начало транзакции* и *Транзакция завершена,* должны быть выполнены повторно. Процедура повторного прогона транзакций выполняет все операции записи в БД, используя информацию о состоянии элементов данных после обновления, содержащуюся в записях журнала по данной транзакции, причем *в том порядке, в каком они были записаны в файле журнала.* Если операции записи уже были успешно завершены до возникновения отказа, это не окажет никакого влияния на состояние элементов данных, поскольку они не могут быть испорчены, если будут записываться еще раз. Однако этот метод гарантирует, что будут обновлены любые элементы данных, которые не были корректно обновлены до момента отказа.

6. Любая транзакция, для которой в файле журнала присутствуют записи *Начало транзакции* и *Отмена транзакции*, просто игнорируется, поскольку никаких реальных обновлений информации в БД по ней не выполнялось, а значит, не требуется и реального выполнения их отката.

Если в процессе восстановления возникнет другой системный сбой, записи файла журнала могут быть использованы для восстановления БД еще раз. В таком случае не имеет значения, сколько раз каждая из строк журнала была использована для повторного внесения изменений в БД.

Метод немедленного обновления. При использовании этого протокола все изменения вносятся в БД сразу же после их выполнения в транзакции, не дожидаясь ее завершения. Помимо необходимости повторного прогона изменений, выполненных транзакциями, закончившимися до появления сбоя, в данном случае может потребоваться выполнить откат изменений, внесенных транзакциями, которые не были завершены к этому моменту. При применении данного метода файл журнала используется с целью восстановления следующим образом.

1. При запуске транзакции в журнал помещается запись: *Начало транзакции.*

2. При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные.

3. Как только упомянутая выше запись будет помещена в файл журнала, все выполненные обновления вносятся в буфера БД.

4. В собственно файлы БД изменения будут внесены при очередной разгрузке буферов БД во вторичную память.

5. Когда транзакция завершает свое выполнение, в файл журнала заносится запись: *Транзакция завершена.*

Очень важно, чтобы в файл журнала все записи (или хотя бы определенная их часть) помещались *до* внесения соответствующих изменений в БД. Это требование известно как *протокол предварительной записи журнала*. Если изменения вначале будут внесены в БД и сбой в системе возникнет до помещения информации об этом в файл журнала, то менеджер восстановления не будет иметь возможности отменить (или повторить) данную операцию. При использовании протокола предварительной записи журнала менеджер восстановления всегда сможет безопасно предположить, что если для определенной транзакции в файле журнала отсутствует запись *Транзакция завершена*, значит, эта транзакция была активна в момент возникновения отказа и, следовательно, должна быть отменена.

Если выполнение транзакции было прекращено, то для отмены выполненных ею изменений может быть использован файл журнала, так как в нем сохранены сведения об исходных значениях всех измененных элементов данных. Поскольку транзакция может выполнить несколько изменений одного и того же элемента, отмена обновлений выполняется в обратном порядке. Независимо от того, были ли результаты выполнения транзакции внесены в саму БД, наличие в записях журнала исходных значений полей гарантирует, что БД будет приведена в состояние, отвечающее началу отмененной транзакции.

На случай отказа системы процедурой восстановления предусмотрено использование файла журнала для повторного прогона или отката транзакций. Для любой транзакции *T*, для которой в файле журнала присутствуют записи *Начало транзакции* и *Транзакция завершена*, следует выполнить ее повторный прогон, используя для внесения в БД значения после изменения всех обновленных полей, как было описано выше. Отметим, что если новые значения уже были реально внесены в файлы БД, повторная их перезапись хотя и будет излишней, тем не менее, не окажет на БД никакого отрицательного влияния. Те же изменения, которые еще не достигли БД к моменту отказа, будут в нее внесены в процессе восстановления. Для любой транзакции *S*, для которой в файле журнала присутствует запись *Начало транзакции*, но нет записи *Транзакция завершена*, необходимо выполнить откат внесенных ею изменений. На этот раз из записей файла журнала извлекается информация о значении измененных полей до их изменения, что позволяет привести базу данных в стояние, которое она имела до начала данной транзакции. Операции отмены выполняются в порядке, обратном порядку их записи в файл журнала.

Метод теневых страниц. Этот метод является альтернативой описанным выше схемам восстановления, построенным на использовании файла журнала. Он предусматривает организацию на время выполнения транзакции двух таблиц страниц – текущую и теневую. Когда транзакция начинает работу, обе таблицы страниц идентичны. *Теневая таблица страниц* никогда не изменяется и может быть использована для восстановления БД в случае отказа системы. В ходе выполнения транзакции *текущая таблица страниц* используется для записи в БД всех вносимых изменений. После завершения транзакции текущая таблица становится теневой. Метод теневых страниц имеет ряд преимуществ перед методами использования журнала транзакций: исключается нагрузка, связанная с ведением журнала транзакций, процесс восстановления происходит существенно быстрее, поскольку нет необходимости в повторном прогоне или откате операций. Однако ему свойственны и определенные недостатки: фрагментация данных и необходимость периодического выполнения процедуры сборки мусора для возвращения в систему неиспользуемых блоков памяти.

9. ЗАЩИТА БАЗ ДАННЫХ

9.1. ОСНОВНЫЕ ПОНЯТИЯ

Данные являются ценным ресурсом, доступ к которому необходимо строго контролировать и регламентировать.

Любая СУБД должна гарантировать, что созданная в ее среде база данных будет надежно защищена. Под ***защитой базы данных*** понимают обеспечение защищенности базы данных против любых предумышленных или непредумышленных угроз с помощью различных компьютерных и некомпьютерных средств.

Понятие защиты применимо не только к сохраняемым в базе данным. Бреши в системе защиты могут возникать и в других частях системы, что, в свою очередь, подвергает опасности и собственно БД. Следовательно, защита БД должна охватывать используемое оборудование, программное обеспечение, персонал и собственно данные.

Опасность – это любая ситуация или событие, намеренное или непреднамеренное, которое способно неблагоприятно повлиять на систему.

Основными потенциальными опасностями являются:

1) похищение и фальсификация данных;
2) утрата конфиденциальности (нарушение тайны);
3) нарушение неприкосновенности личных данных;

4) утрата целостности данных;
5) потеря доступности данных.

Похищение и фальсификация данных могут происходить не только в среде баз данных – вся организация так или иначе подвержена этому риску. Однако действия по похищению или фальсификации информации всегда совершаются людьми, поэтому основное внимание должно быть сосредоточено на сокращении общего количества удобных ситуаций для выполнения подобных действий.

Понятие *конфиденциальности* означает необходимость сохранения данных в тайне. Как правило, конфиденциальными считаются те данные, которые являются критичными для всей организации. Понятие неприкосновенности данных касается требования защиты информации об отдельных работниках. Следствием нарушения в системе защиты, вызвавшего потерю конфиденциальности данных, может быть утрата позиций в конкурентной борьбе, тогда как следствием нарушения *неприкосновенности личных данных* будут юридические меры, принятые в отношении организации.

Утрата целостности данных приводит к искажению или разрушению данных, что может иметь самые серьезные последствия для дальнейшей работы организации.

Потеря доступности данных означает, что либо данные, либо система, либо и то, и другое одновременно окажутся недоступными пользователям, что может подвергнуть опасности само дальнейшее существование организации. В некоторых случаях те события, которые послужили причиной перехода системы в недоступное состояние, могут одновременно вызвать и разрушение данных в базе.

9.2. КОМПЬЮТЕРНЫЕ СРЕДСТВА ЗАЩИТЫ

Несмотря на широкий диапазон компьютерных средств контроля, общий уровень защищенности СУБД определяется возможностями используемой операционной системы, поскольку эти два компонента работают в тесной связи между собой. Обычно применяются перечисленные ниже типы средств контроля.

1. Авторизация пользователей.
2. Представления.
3. Резервное копирование и восстановление.
4. Поддержка целостности.
5. Шифрование.
6. Вспомогательные процедуры.

Авторизация – это предоставление прав (или привилегий), позволяющих их владельцу иметь законный доступ к системе или к ее объектам.

Средства авторизации пользователей могут быть встроены непосредственно в программное обеспечение и управлять не только предоставленными пользователям правами доступа к системе или объектам, но и набором операций, которые пользователи могут выполнять с каждым доступным ему объектом. Термин «владелец» в определении может представлять пользователя-человека или программу. Термин «объект» – таблицу данных, представление, приложение, процедуру или любой другой объект, который может быть создан в рамках системы. В некоторых системах все предоставляемые субъекту права должны быть явно указаны для каждого из объектов, к которым этот субъект должен обращаться. Процесс авторизации включает аутентификацию субъектов, требующих получения доступа к объектам.

Аутентификация – механизм определения того, является ли пользователь тем, за кого себя выдает.

За предоставление пользователям доступа к компьютерной системе обычно отвечает системный администратор, в обязанности которого входит создание учетных записей пользователей. Каждому пользователю присваивается уникальный идентификатор, который используется операционной системой для того, чтобы определить, кто есть кто. С каждым идентификатором связывается пароль, выбираемый пользователем и известный операционной системе. При регистрации пользователь должен предоставлять системе свой пароль для аутентификации.

Подобная процедура позволяет организовать контролируемый доступ к компьютерной системе, но не обязательно предоставляет право доступа к СУБД или иной прикладной программе. Для получения пользователем права доступа к СУБД может использоваться отдельная подобная процедура. Ответственность за предоставление прав доступа к СУБД обычно несет администратор базы данных (АБД), в обязанности которого входит создание индивидуальных идентификаторов пользователей, на этот раз уже в среде самой СУБД. Каждый из идентификаторов пользователей СУБД также связывается с паролем, который должен быть известен только данному пользователю. Этот пароль будет использоваться подпрограммами СУБД для идентификации данного пользователя.

Некоторые СУБД поддерживают списки разрешенных идентификаторов пользователей и паролей, отличающиеся от аналогичного списка, поддерживаемого ОС. Другие типы СУБД поддерживают списки, элементы которых приведены в соответствие существующим спискам поль-

зователей операционной системы и выполняют регистрацию, исходя из текущего идентификатора пользователя, указанного им при регистрации в системе. Это предотвращает попытки пользователей зарегистрироваться в среде СУБД под идентификатором, отличным от того, который они использовали при регистрации в системе.

Использование паролей является наиболее распространенным методом аутентификации пользователей. Однако этот подход не дает абсолютной гарантии, что данный пользователь является именно тем, за кого себя выдает.

Как только пользователь получит право доступа к СУБД, ему могут автоматически предоставляться различные другие *привилегии*, связанные с его идентификатором. В частности, эти привилегии могут включать разрешение на доступ к определенным БД, таблицам, представлениям и индексам или же право запуска различных утилит СУБД. Некоторые типы СУБД функционируют как закрытые системы, поэтому пользователям помимо разрешения на доступ к самой СУБД потребуется иметь отдельные разрешения и на доступ к конкретным ее объектам. Эти разрешения выдаются либо АБД, либо владельцами определенных объектов системы. В противоположность этому открытые системы по умолчанию предоставляют авторизированным пользователям полный доступ ко всем объектам БД. В этом случае привилегии устанавливаются посредством явной отмены тех или иных прав конкретных пользователей. Типы привилегий, которые могут быть предоставлены авторизированным субъектам, включают, например, право доступа к указанным БД, право выборки данных, право создания таблиц и других объектов.

Некоторыми объектами в среде СУБД владеет сама СУБД. Обычно это владение организуется посредством использования специального идентификатора особого *суперпользователя* – например, с именем Database Administrator. Как правило, владение некоторым объектом предоставляет его владельцу весь возможный набор привилегий в отношении этого объекта. Это правило применяется ко всем авторизированным пользователям, получающим права владения определенными объектами. Любой вновь созданный объект автоматически передается во владение его создателю, который и получает весь возможный набор привилегий для данного объекта. Тем не менее, хотя пользователь может быть владельцем некоторого представления, единственной привилегией, которая будет предоставлена ему в отношении этого объекта, может оказаться право выборки данных из этого представления. Принадлежащие владельцу привилегии могут быть переданы им другим авторизированным пользователям. В некоторых типах СУБД всякий раз, когда пользовате-

лю предоставляется определенная привилегия, дополнительно может указываться, передается ли ему право предоставлять эту привилегию другим пользователям (уже от имени этого пользователя). Естественно, что в этом случае СУБД должна контролировать всю цепочку предоставления привилегий пользователям с указанием того, кто именно ее предоставил, что позволит поддерживать корректность всего набора установленных в системе привилегий. В частности, эта информация будет необходима в случае отмены предоставленных ранее привилегий для организации каскадного распространения вносимых изменений среди цепочки пользователей.

Если СУБД поддерживает несколько различных типов идентификаторов авторизации, с каждым из существующих типов могут быть связаны различные приоритеты. В частности, если СУБД поддерживает использование идентификаторов как отдельных пользователей, так и их групп, то, как правило, идентификатор пользователя будет иметь более высокий приоритет, чем идентификатор группы.

Очень важно освоить все механизмы авторизации и другие средства защиты, предоставляемые целевой СУБД. Это особенно важно для тех систем, в которых существуют различные типы идентификаторов и допускается передача права присвоения привилегий. Это позволит корректно выбирать типы привилегий, предоставляемых отдельным пользователям, исходя из исполняемых ими обязанностей и набора используемых прикладных программ.

Представление – это динамический результат одной или нескольких реляционных операций с базовыми отношениями с целью создания некоторого иного отношения. Представление является виртуальным отношением, которого реально в БД не существует, но которое создается по требованию отдельного пользователя в момент поступления этого требования.

Механизм представления является мощным и гибким инструментом организации защиты данных, позволяющим скрыть от определенных пользователей некоторые части БД. В результате пользователи не будут иметь никаких сведений о существовании любых атрибутов или строк данных, которые недоступны через представления, находящиеся в их распоряжении. Представление может быть определено на базе нескольких таблиц, после чего пользователю будут предоставлены необходимые привилегии доступа к этому представлению, но не к базовым таблицам. В данном случае использование представления являет собой более жесткий механизм контроля доступа, чем обычное предоставление пользователю тех или иных прав доступа к базовым таблицам.

Резервное копирование – периодически выполняемая процедура получения копии БД и ее файла журнала (а также, возможно, программ) на носителе, сохраняемом отдельно от системы.

Средства резервного копирования позволяют восстанавливать БД в случае ее разрушения. Рекомендуется создавать резервные копии БД и ее файла журнала с некоторой установленной периодичностью и организовывать хранение созданных копий в местах, обеспеченных необходимой защитой. В случае отказа, в результате которого БД становится непригодной для дальнейшей эксплуатации, резервная копия и зафиксированная в файле журнала оперативная информация используются для восстановления БД до последнего согласованного состояния.

Ведение журнала – это процедура создания и обслуживания файла журнала, содержащего сведения обо всех изменениях, внесенных в БД с момента создания последней резервной копии, и предназначенного для обеспечения эффективного восстановления системы в случае ее отказа.

Если в отказавшей системе функция ведения системного журнала не использовалась, базу данных можно будет восстановить только до того состояния, которое было зафиксировано в последней созданной резервной копии. Все изменения, которые были внесены в БД после создания последней резервной копии, окажутся потерянными.

Контрольная точка – это момент синхронизации между состоянием БД и состоянием журнала выполнения транзакций. В этот момент все буфера принудительно выгружаются на устройства вторичной памяти.

Механизм создания контрольных точек используется совместно с ведением системного журнала, что повышает эффективность процесса восстановления. В момент создания контрольной точки СУБД выполняет действия, обеспечивающие запись на диск всех данных, хранившихся в основной памяти машины, а также помещение в файл журнала специальной записи контрольной точки.

Средства ***поддержки целостности*** данных также вносят определенный вклад в общую защищенность БД, поскольку их назначением является предотвращение перехода данных в несогласованное состояние, а значит, и предотвращение угрозы получения ошибочных или некорректных результатов расчетов.

Шифрование – кодирование данных с использованием специального алгоритма, в результате чего данные становятся недоступными для чтения любой программой, не имеющей ключа дешифрования.

Если в системе с БД содержится весьма важная конфиденциальная информация, то имеет смысл закодировать ее с целью предупреждения возможной угрозы несанкционированного доступа с внешней стороны

(по отношению к СУБД). Некоторые СУБД включают средства шифрования, предназначенные для использования в подобных целях. Подпрограммы таких СУБД обеспечивают санкционированный доступ к данным (после их декодирования), хотя это связано с некоторым снижением производительности, вызванным необходимостью перекодировки. Шифрование также может использоваться для защиты данных при их передаче по линиям связи. Существует множество различных технологий кодирования данных с целью сокрытия передаваемой информации. *Необратимые* технологии, как и следует из их названия, не позволяют установить исходные данные, хотя последние могут использоваться для сбора достоверной статистической информации. *Обратимые* технологии используются чаще. Для организации защищенной передачи данных по незащищенным сетям должны использоваться системы шифрования, включающие следующие компоненты:

1) ключ шифрования, предназначенный для шифрования исходных данных (обычного текста);

2) алгоритм шифрования, который описывает, как с помощью ключа шифрования преобразовать обычный текст в шифротекст;

3) ключ дешифрования, предназначенный для дешифрования шифротекста;

4) алгоритм дешифрования, который описывает, как с помощью ключа дешифрования преобразовать шифротекст в исходный обычный текст.

Некоторые системы шифрования, называемые *симметричными*, используют один и тот же ключ как для шифрования, так и для дешифрования, при этом предполагается наличие защищенных линий связи, предназначенных для обмена ключами. Однако большинство пользователей не имеют доступа к защищенным линиям связи, поэтому для получения надежной защиты длина ключа должна быть не меньше длины самого сообщения. Тем не менее большинство эксплуатируемых систем построено на использовании ключей, которые короче самих сообщений. Одна из распространенных систем шифрования называется DES (Data Encryption Standard) – в ней используется стандартный алгоритм шифрования, разработанный фирмой IBM. В этой схеме для шифрования и дешифрования используется один и тот же ключ, который должен храниться в секрете, хотя сам алгоритм шифрования не является секретным. Этот алгоритм предусматривает преобразование каждого 64-битового блока обычного текста с использованием 56-битового ключа шифрования. В системе шифрования POP (Pretty Good Privacy) используется 128-битовый симметричный алгоритм, применяемый для шифрования блоков отсылаемых данных.

Другой тип систем шифрования предусматривает использование для шифровки и дешифровки сообщений различных ключей – подобные системы принято называть *несимметричными*. Примером является система с открытым ключом, предусматривающая использование двух ключей, один из которых является открытым, а другой хранится в секрете. Алгоритм шифрования также может быть открытым, поэтому любой пользователь, желающий направить владельцу ключей зашифрованное сообщение, может использовать его открытый ключ и соответствующий алгоритм шифрования. Однако дешифровать данное сообщение сможет только тот, кто знает парный закрытый ключ шифрования. Системы шифрования с открытым ключом могут также использоваться для отправки вместе с сообщением «цифровой подписи», подтверждающей, что данное сообщение было действительно отправлено владельцем открытого ключа. Наиболее популярной несимметричной системой шифрования является RSA (это инициалы трех разработчиков данного алгоритма). Как правило, симметричные алгоритмы являются более быстродействующими, чем несимметричные, однако на практике обе схемы часто применяются совместно, когда алгоритм с открытым ключом используется для шифрования случайным образом сгенерированного ключа шифрования, а уже этот случайный ключ – для шифровки самого сообщения с применением некоторого симметричного алгоритма.

Описанные выше различные механизмы, которые могут использоваться для защиты данных в среде СУБД, сами по себе не гарантируют необходимого уровня защищенности и могут оказаться неэффективными в случае неправильного применения или управления. По этой причине должны использоваться также различные ***вспомогательные процедуры***.

Одно из назначений процедуры *аудита* – проверка того, все ли предусмотренные средства управления задействованы и соответствует ли уровень защищенности установленным требованиям. В ходе выполнения инспекции аудиторы могут ознакомиться с используемыми ручными процедурами, обследовать компьютерные системы и проверить состояние всей имеющейся документации на данную систему.

Для определения активности использования БД анализируются ее файлы журнала. Эти же источники могут использоваться для выявления любых необычных действий в системе. Регулярное проведение аудиторских проверок, дополненное постоянным контролем содержимого файлов журнала с целью выявления ненормальной активности в системе, очень часто позволяет своевременно обнаружить и пресечь любые попытки нарушения защиты.

В процессе *установки нового прикладного программного обеспечения* его обязательно следует тщательно протестировать, прежде чем принимать решение об их развертывании и передаче в эксплуатацию. Если уровень тестирования будет недостаточным, существенно возрастает риск разрушения БД. Следует считать хорошей практикой выполнение резервного копирования БД непосредственно перед сдачей нового программного обеспечения в эксплуатацию. Кроме того, в первый период эксплуатации нового приложения обязательно следует организовать тщательное наблюдение за функционированием системы. Отдельным вопросом, который должен быть согласован со сторонними разработчиками программ, является право собственности на разработанное ими программное обеспечение. Данная проблема должна быть решена еще до начала разработки, причем это особенно важно в тех случаях, когда существует вероятность, что впоследствии организации обязательно потребуется вносить изменения в создаваемые приложения. Риск, связанный с подобной ситуацией, состоит в том, что организация не будет иметь юридического права использовать данное программное обеспечение или модернизировать его. Потенциально подобная ситуация угрожает организации серьезными потерями.

В обязанности АБД входит выполнение *модернизации программного обеспечения* СУБД при поступлении от разработчика очередных пакетов изменений. В некоторых случаях вносимые изменения оказываются совсем незначительными и касаются небольшой части модулей системы. Однако возможны ситуации, когда требуется полная ревизия всей установленной системы. Никакие изменения и модернизации ни в коем случае не должны вноситься в систему без предварительной оценки их возможного влияния на имеющиеся данные и программное обеспечение.

Результатом ознакомления с сопроводительной документацией пакета должен быть план действий по его установке. Главная задача АБД – обеспечить полный и безболезненный переход от старой версии системы к новой.

9.3. НЕКОМПЬЮТЕРНЫЕ СРЕДСТВА ЗАЩИТЫ.

Некомпьютерные средства защиты включают выработку ограничений, соглашений и других административных мер, не связанных с компьютерной поддержкой. К ним относятся:

1) меры обеспечения безопасности и планирование защиты от непредвиденных обстоятельств;
2) контроль за персоналом;
3) защита помещений и хранилищ;

4) гарантийные соглашения;

5) договора о сопровождении;

6) контроль за физическим доступом.

В документе по *мерам обеспечения безопасности* должно быть определено следующее:

1) область деловых процессов организации, для которой они устанавливаются;

2) ответственность и обязанности отдельных работников;

3) дисциплинарные меры, принимаемые в случае обнаружения нарушения установленных ограничений;

4) процедуры, которые должны обязательно выполняться.

План защиты от непредвиденных обстоятельств разрабатывается с целью подробного определения последовательности действий, необходимых для выхода из разных необычных ситуаций, не предусмотренных процедурами нормального функционирования системы.

Контроль за персоналом. Создатели коммерческих СУБД возлагают всю ответственность за эффективное управления системой на ее пользователей. Поэтому с точки зрения защиты системы исключительно важную роль играют отношение к делу и действия людей, непосредственно вовлеченных в эти процессы.

Защита помещений и хранилищ. Основное оборудование системы, включая принтеры, если они используются для печати конфиденциальной информации, должно размещаться в запираемом помещении с ограниченным доступом, который должен быть разрешен только основным специалистам. Все остальное оборудование, особенно переносное, должно быть закреплено в месте размещения и снабжено сигнализацией.

Гарантийные соглашения представляют собой юридические соглашения в отношении программного обеспечения, заключаемые между разработчиками программ и их клиентами. На основании этих соглашений некоторая третья фирма обеспечивает хранение исходного текста программ приложения, разработанного для клиента. Это одна из форм страховки клиента на случай, если компания-разработчик отойдет от дел. В этом случае клиент получит право забрать исходные тексты программ у третьей фирмы, вместо того чтобы остаться с приложением, лишенным всякого сопровождения.

Для всего используемого организацией оборудования и программного обеспечения сторонней разработки или изготовления обязательно должны быть заключены соответствующие ***договора о сопровождении***.

Методы ***контроля за физическим доступом*** к оборудованию могут быть разделены на внешние и внутренние. Внутренний контроль исполь-

зуется внутри отдельных зданий и предназначен для управления теми лицами, кто будет иметь доступ в определенные помещения. Внешний метод контроля применяется вне строений и предназначен для организации доступа на площадку или в отдельные здания.

10. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

10.1. ОСНОВНЫЕ КОНЦЕПЦИИ

Распределенная база данных – это набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети.

Программный комплекс, предназначенный для управления распределенными БД и позволяющий сделать распределенность информации прозрачной для конечного пользователя, является ***распределенной СУБД***.

Система управления распределенными базами данных (СУРБД) состоит из единой логической БД, разделенной на некоторое количество *фрагментов*. Каждый фрагмент БД сохраняется на одном или нескольких компьютерах, которые соединены между собой линиями связи и каждый из которых работает под управлением отдельной СУБД. Любой из сайтов способен независимо обрабатывать запросы пользователей, требующие доступа к локально сохраненным данным, а также способен обрабатывать данные, сохраненные на других компьютерах сети.

Пользователи взаимодействуют с распределенной БД через приложения. *Локальные приложения* не требуют доступа к данным на других сайтах, *глобальные приложения* требуют подобного доступа. В распределенных СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая СУРБД должна обладать следующими свойствами:

1. Иметь набор логически связанных разделяемых данных.
2. Сохраняемые данные должны быть разбиты на некоторое количество фрагментов.
3. Между фрагментами может быть организована репликация данных.
4. Фрагменты и их реплики должны быть распределены по различным сайтам.
5. Сайты связаны между собой сетевыми соединениями.
6. Работа с данными на каждом сайте управляется СУБД.
7. СУБД на каждом сайте способна поддерживать автономную работу локальных приложений.

8. СУБД каждого сайта поддерживает хотя бы одно глобальное приложение.

Нет необходимости в том, чтобы на каждом сайте системы существовала своя собственная локальная БД (рис. 10).

От пользователей должен быть полностью скрыт тот факт, что распределенная БД состоит из нескольких фрагментов, т. е. для конечного пользователя распределенность системы должна быть полностью *прозрачна* (*невидима*). Назначение обеспечения прозрачности состоит в том, чтобы распределенная система внешне вела себя точно так, как и централизованная. Это требование называют *основным принципом* построения распределенных СУБД.

Очень важно понимать различия, существующие между распределенными СУБД и распределенной обработкой данных.

Распределенная обработка является обработкой с использованием централизованной БД, доступ к которой может осуществляться с различных компьютеров сети.

Ключевым моментом в определении распределенной БД является утверждение, что система работает с данными, физически расположенными в сети.

Кроме того, следует четко понимать различия, существующие между распределенными и параллельными СУБД. *Параллельная СУБД* функционирует с использованием нескольких процессоров и устройств жестких дисков, что позволяет ей (если это возможно) распараллеливать выполнение некоторых операций с целью повышения общей производительности обработки.

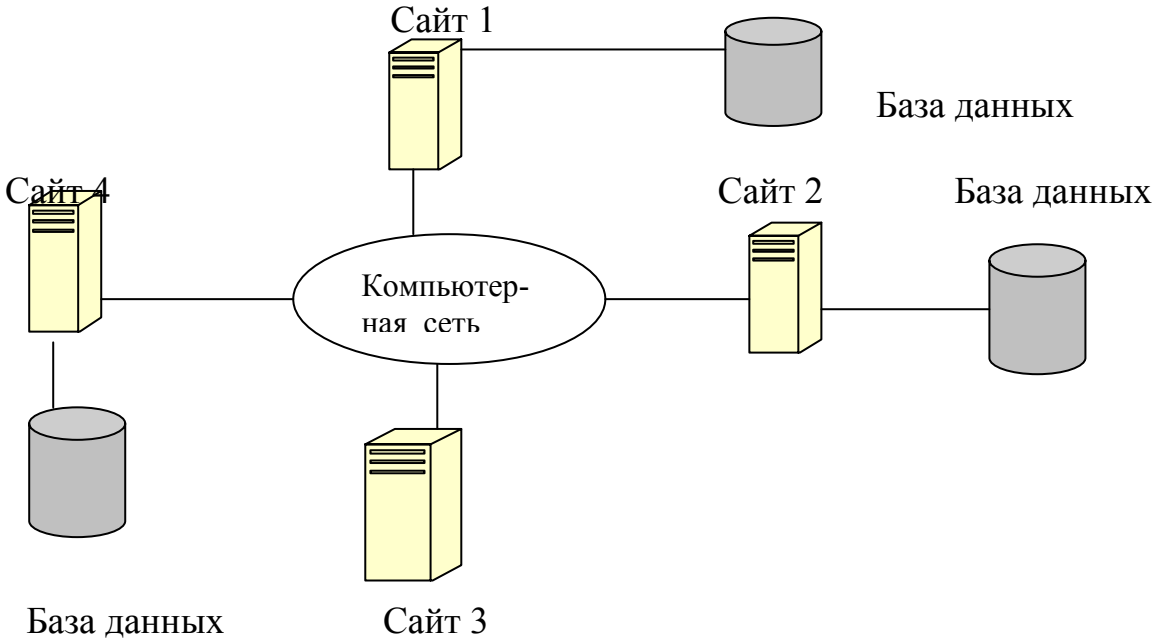


Рис. 10. Топология СУРБД

К преимуществам СУРБД можно отнести:
1) отражение структуры организации;
2) разделяемость и локальная автономность;
3) повышение доступности данных;
4) повышение надежности;
5) повышение производительности;
6) экономические выгоды;
7) модульность системы.

Недостатками СУРБД являются:
1) повышение сложности;
2) увеличение стоимости;
3) проблемы защиты;
4) усложнение контроля за целостностью данных;
5) отсутствие стандартов;
6) недостаток опыта;
7) усложнение процедуры разработки базы данных.

Распределенные СУБД разделяются на гомогенные и гетерогенные. В *гомогенных* системах все сайты используют один и тот же тип СУБД. В *гетерогенных* системах на сайтах могут функционировать различные типы СУБД, использующие разные модели данных.

Гомогенные системы значительно проще проектировать и сопровождать. Кроме того, подобный подход позволяет поэтапно наращивать размеры системы, последовательно добавляя новые сайты к уже существующей распределенной системе. Дополнительно появляется возможность повышать производительность системы за счет организации на различных сайтах параллельной обработки данных.

Гетерогенные системы возникают в тех случаях, когда независимые сайты, уже эксплуатирующие свои собственные системы с БД, интегрируются во вновь создаваемую распределенную систему. Здесь для организации взаимодействия между различными типами СУБД потребуется организовать трансляцию передаваемых сообщений. Для обеспечения прозрачности в отношении типа используемой СУБД пользователи каждого из сайтов должны иметь возможность вводить интересующий их запрос на языке той СУБД, которая используется на данном сайте.

Одной из разновидностей распределенных СУБД являются мультибазовые системы. *Мультибазовая система* – это распределенная СУБД, в которой управление каждым сайтом осуществляется совершенно авто-

номно. Мультибазовая СУБД прозрачна: она располагается поверх существующих баз данных и файловых систем, предоставляя их своим пользователям как некую единую БД.

Существуют *нефедеральные* (не имеющие локальных пользователей) и *федеральные* мультибазовые системы. Федеральная система представляет собой некоторый гибрид распределенной и централизованной системы, поскольку она выглядит как распределенная для удаленных пользователей и как централизованная – для локальных.

10.2. ФУНКЦИИ РАСПРЕДЕЛЕННЫХ СУБД

Типичная СУРБД должна обеспечивать, по крайней мере, тот же набор функциональных возможностей, который был определен для централизованных СУБД. Кроме того, СУРБД должна предоставлять следующий набор функциональных возможностей.

1. Расширенные службы установки соединений должны обеспечивать доступ к удаленным сайтам и позволять передавать запросы и данные между сайтами, входящими в сеть.
2. Расширенные средства ведения каталога должны сохранять сведения о распределении данных в сети.
3. Средства обработки распределенных запросов должны обеспечивать оптимизацию запросов и организацию удаленного доступа.
4. Расширенные функции управления параллельностью должны поддерживать целостность реплицируемых данных.
5. Расширенные функции восстановления должны учитывать возможность отказов в работе сайтов и отказов линий связи.

10.3. РАЗРАБОТКА РАСПРЕДЕЛЕННЫХ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ.

При разработке распределенных реляционных БД возникают следующие аспекты проектирования:

1. *Фрагментация*. Любое отношение может быть разделено на некоторое количество частей, называемых фрагментами, которые затем распределяются по различным сайтам. Существуют два основных типа фрагментов: горизонтальные и вертикальные. Горизонтальные фрагменты представляют собой подмножества кортежей, а вертикальные – подмножества атрибутов.
2. *Распределение.* Каждый фрагмент сохраняется на сайте, выбранном с учетом «оптимальной» схемы их размещения.
3. *Репликация.* СУРБД может поддерживать актуальную копию некоторого фрагмента на нескольких различных сайтах.

Определение и размещение фрагментов должно проводиться с учетом особенностей использования БД. В частности, это подразумевает выполнение анализа приложений.

Проектирование должно выполняться как на основе количественных, так и качественных показателей. Количественная информация используется как основа для распределения, тогда как качественная служит базой при создании схемы фрагментации. Количественная информация включает такие показатели:

1) частота запуска приложения на выполнение;
2) сайт, на котором запускается приложение;
3) требования к производительности транзакций и приложений.

Качественная информация может включать перечень выполняемых в приложении транзакций, используемые отношения, атрибуты и кортежи, к которым осуществляется доступ, тип доступа (чтение или запись), предикаты, используемые в операциях чтения.

Определение и размещение фрагментов по сайтам выполняется для достижения следующих стратегических целей.

1. *Локальность ссылок.* Везде, где только это возможно, данные должны храниться как можно ближе к местам их использования. Если фрагмент используется на нескольких сайтах, может оказаться целесообразным разместить на этих сайтах его копии.

2. *Повышение надежности и доступности.* Надежность и доступность данных повышаются за счет использования механизма репликации. В случае отказа одного из сайтов всегда будет существовать копия фрагмента, сохраняемая на другом сайте.

3. *Приемлемый уровень производительности.* Неверное распределение данных будет иметь следствием возникновение в системе узких мест. В этом случае некоторый сайт оказывается просто завален запросами со стороны других сайтов, что может вызвать существенное снижение производительности всей системы. В то же время неправильное распределение может иметь следствием неэффективное использование ресурсов системы.

4. *Баланс между емкостью и стоимостью внешней памяти.* Обязательно следует учитывать доступность и стоимость устройств хранения данных, имеющихся на каждом из сайтов системы. Везде, где только это возможно, рекомендуется использовать более дешевые устройства массовой памяти. Это требование должно быть сбалансировано с требованием поддержки *локальности ссылок.*

5. *Минимизация расходов на передачу данных.* Следует тщательно учитывать стоимость выполнения в системе удаленных запросов. Затра-

ты на выборку будут минимальны при обеспечении максимальной *локальности ссылок*, т. е. когда каждый сайт будет иметь собственную копию данных. Однако при обновлении реплицируемых данных внесенные изменения потребуется распространить на все сайты, имеющие копию обновленного отношения, что вызовет увеличение затрат на передачу данных.

10.4. РАСПРЕДЕЛЕНИЕ ДАННЫХ

Существуют четыре альтернативные стратегии размещения данных в системе: централизованное, раздельное (фрагментированное), размещение с полной репликацией и с выборочной репликацией.

Централизованное размещение. Данная стратегия предусматривает создание на одном из сайтов единственной базы данных под управлением СУБД, доступ к которой будут иметь все пользователи сети. В этом случае локальность ссылок минимальна для всех сайтов, за исключением центрального, поскольку для получения любого доступа к данным требуется установка сетевого соединения. Соответственно уровень затрат на передачу данных будет высок. Уровень надежности и доступности в системе низок, поскольку отказ на центральном сайте вызовет паралич работы всей системы.

Раздельное (фрагментированное) размещение. В этом случае БД разбивается на непересекающиеся фрагменты, каждый из которых размещается на одном из сайтов системы. Если элемент данных будет размещен на том сайте, на котором он чаще всего используется, полученный уровень локальности ссылок будет высок. При отсутствии репликации стоимость хранения данных будет минимальна, но при этом будет невысок также уровень надежности и доступности данных в системе. Однако он будет выше, чем в предыдущем варианте, поскольку отказ на любом из сайтов вызовет утрату доступа только к той части данных, которая на нем хранилась. При правильно выбранном способе распределения данных уровень производительности в системе будет относительно высоким, а уровень затрат на передачу данных – низким.

Размещение с полной репликацией. Эта стратегия предусматривает размещение полной копии всей БД на каждом из сайтов системы. Следовательно, локальность ссылок, надежность и доступность данных, а также уровень производительности системы будут максимальны. Однако стоимость устройств хранения данных и уровень затрат на передачу данных в этом случае также будут самыми высокими. Для преодоления части этих проблем в некоторых случаях используется технология моментальных снимков. *Моментальный снимок* представляет собой копию БД

в определенный момент времени. Эти копии обновляются через некоторый установленный интервал времени, например один раз в час или в неделю, а потому они не всегда будут актуальными в текущий момент. Иногда в распределенных системах моментальные снимки используются для реализации представлений, что позволяет улучшить время выполнения в БД операций с представлениями.

Размещение с выборочной репликацией. Данная стратегия представляет собой комбинацию методов фрагментации, репликации и централизации. Одни массивы данных разделяются на фрагменты, что позволяет добиться для них высокой локальности ссылок, тогда как другие, используемые на многих сайтах, но не подверженные частым обновлениям, подвергаются репликации. Все остальные данные хранятся централизованно. Целью применения стратегии является объединение всех преимуществ, существующих в остальных моделях, с одновременным исключением свойственных им недостатков. Благодаря своей гибкости именно эта стратегия используется чаще всего.

10.5. ФРАГМЕНТАЦИЯ

Корректность фрагментации. Фрагментация данных не должна выполняться непродуманно, наугад. Существуют три правила, которых следует обязательно придерживаться при проведении фрагментации.

1. *Полнота.* Если экземпляр отношения R разбивается на фрагменты, например R_1, R_2, ..., R_n, то каждый элемент данных, присутствующий в отношении R, должен присутствовать, по крайней мере, в одном из созданных фрагментов. Выполнение этого правила гарантирует, что какие-либо данные не будут утрачены в результате выполнения фрагментации.

2. *Восстановимость.* Должна существовать операция реляционной алгебры, позволяющая восстановить отношение R из его фрагментов. Это правило гарантирует сохранение функциональных зависимостей.

3. *Непересекаемость.* Если элемент данных d_i присутствует во фрагменте R_i, то он не должен одновременно присутствовать в каком-либо ином фрагменте. Исключением из этого правила является операция вертикальной фрагментации, поскольку в этом случае в каждом фрагменте должны присутствовать атрибуты первичного ключа, необходимые для восстановления исходного отношения. Это правило гарантирует минимальную избыточность данных во фрагментах.

Типы фрагментации. Существуют два основных типа фрагментации: *горизонтальная* и *вертикальная*. В случае горизонтальной фрагментации элементом данных является кортеж, а в случае вертикальной фрагментации – атрибут. Горизонтальные фрагменты представляют собой подмно-

жества кортежей отношения, а вертикальные – подмножества атрибутов отношения, как показано на рис. 11.

Кроме того, существуют еще два типа фрагментации: *смешанная* (рис. 12) и *производная* (представляющая собой вариант горизонтальной фрагментации).

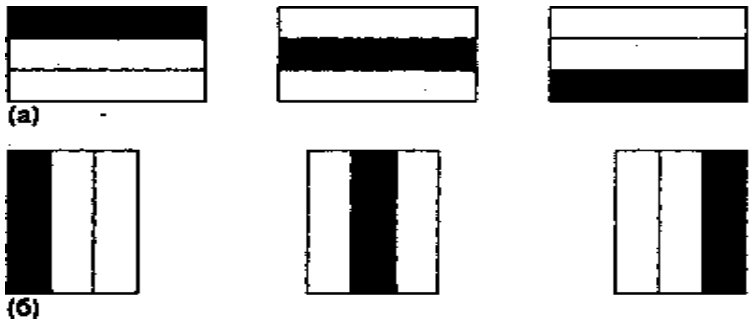


Рис. 11. Различные типы фрагментации: а) – горизонтальная; б) – вертикальная

Рис. 12. Смешанная фрагментация:
а) – горизонтально разделенные вертикальные фрагменты;
б) – вертикально разделенные горизонтальные фрагменты

Горизонтальная фрагментация. *Горизонтальный фрагмент* – это выделенный по горизонтали фрагмент отношения, состоящий из некоторого подмножества кортежей этого отношения

Горизонтальный фрагмент создается посредством определения предиката, с помощью которого выполняется отбор кортежей из исходного отношения. Данный тип фрагмента определяется с помощью операции выборки реляционной алгебры. Операция выборки позволяет выделить группу кортежей, обладающих некоторым общим для них свойством, например все кортежи, используемые одним из приложений, или все кортежи, применяемые на одном из сайтов. Если задано отношение R, то его горизонтальный фрагмент может быть определен с помощью следующей формулы: $\sigma_p(R)$. Здесь p является предикатом, построенным с использованием одного или больше атрибутов отношения.

В одних случаях целесообразность использования горизонтальной фрагментации вполне очевидна. Однако в других случаях потребуется выполнение детального анализа приложений. Этот анализ должен включать проверку предикатов (или условий) поиска, используемых в транзакциях или запросах, выполняемых в приложении. Предикаты могут быть *простыми*, включающими только по одному атрибуту, или *сложными*, включающими несколько атрибутов. Для каждого из используемых атрибутов предикат может содержать единственное значение или несколько значений. В последнем случае значения могут быть дискретными или задавать диапазон значений.

Стратегия определения типа фрагментации предполагает поиск набора *минимальных* (т. е. полных и релевантных) предикатов, которые можно будет использовать как основу для построения схемы фрагментации. Набор предикатов является *полным* тогда и только тогда, когда вероятность обращения к любым двум кортежам одного и того же фрагмента со стороны любого приложения будет одинакова. Предикат является релевантным, если существует по крайней мере одно приложение, которое по-разному обращается к выделенным с помощью этого предиката фрагментам.

Вертикальная фрагментация. *Вертикальный фрагмент* – выделенный по вертикали фрагмент отношения, состоящий из подмножества атрибутов этого отношения.

При вертикальной фрагментации в различные фрагменты объединяются атрибуты, используемые отдельными приложениями. Определение фрагментов в этом случае выполняется с помощью операции *проекции* реляционной алгебры. Для заданного отношения R вертикальный фрагмент может быть вычислен с помощью формулы: $\pi_{a_1,\ldots,a_n}(R)$. Здесь a_1, …, a_n представляют собой атрибуты отношения R.

Вертикальные фрагменты определяются путем установки родственности одного атрибута по отношению к другому. Один из способов решить эту задачу состоит в создании матрицы, содержащей количество обращений с выборкой каждой из пар атрибутов. Например, транзакция, которая осуществляет доступ к атрибутам a_1, a_2 и a_4 отношения R, состоящего из набора атрибутов (a_1, a_2, a_3, a_4), может быть представлена следующей матрицей:

	a_1	a_2	a_3	a_4
a_1	1	0	1	
a_2		0	1	
a_3			0	

a_4⌊ ⌋

Эта матрица является треугольной, поскольку диагональ ее не заполняется, а нижняя часть является зеркальным отражением верхней части. Единицы в матрице означают наличие доступа с обращением к соответствующей паре атрибутов и, в конечном счете, должны быть заменены числами, отражающими частоту выполнения транзакции. Подобная матрица составляется для каждой транзакции, после чего строится общая матрица, содержащая суммы всех показателей доступа к каждой из пар атрибутов. Пары атрибутов с высоким показателем родственности должны присутствовать в одном и том же вертикальном фрагменте. Пары с невысоким показателем родственности могут быть разнесены в разные вертикальные фрагменты. Очевидно, что обработка сведений об отдельных атрибутах для всех важнейших транзакций может потребовать немало времени и вычислений. Следовательно, если заранее известно о родственности определенных атрибутов, может оказаться целесообразным обработать сведения сразу о группах атрибутов.

Смешанная фрагментация. В некоторых случаях применения только лишь горизонтальной и вертикальной фрагментации элементов схемы БД оказывается недостаточно для адекватного распределения данных между приложениями. Тогда приходится прибегать к смешанной (или гибридной) фрагментации.

Смешанный фрагмент образуется либо посредством дополнительной вертикальной фрагментации созданных ранее горизонтальных фрагментов, либо за счет вторичной горизонтальной фрагментации предварительно определенных вертикальных фрагментов.

Смешанная фрагментация определяется с помощью операций выборки и проекции реляционной алгебры. Если имеется некоторое отношение R, то смешанный фрагмент может быть определен по формуле $\sigma_P(\pi a_1, a_2, \ldots, a_n (R))$. Здесь p является предикатом, построенным на использовании одного или больше атрибутов отношения R, обозначенных в формулах символами $a_1, a_2, \ldots, a_n$.

Производная горизонтальная фрагментация. Некоторые приложения включают операции соединения двух или больше отношений. Если отношения сохраняются в различных местах, то выполнение их соединения создаст очень большую дополнительную нагрузку на систему. В подобных случаях более приемлемым решением будет размещение соединяемых отношений или их фрагментов в одном и том же месте. Цель может быть достигнута за счет применения производной горизонтальной фрагментации.

Производный фрагмент – горизонтальный фрагмент отношения, созданный на основе горизонтального фрагмента родительского отношения.

Термин «*дочернее»* мы будем использовать для ссылок на отношение, содержащее внешний ключ, а термин «*родительское»* – для ссылок на отношение с соответствующим первичным ключом. Определение производных фрагментов осуществляется с помощью операции *полусоединения* реляционной алгебры. Если заданы дочернее отношение R и родительское отношение S, то производный фрагмент отношения R может быть определен следующим образом: $Ri = R \mid >< \mid_F Si$, $1 \le i \le w$.

Здесь значение w – это количество горизонтальных фрагментов, определенных для отношения S, а параметр F задает атрибут, по которому выполняется соединение.

Отказ от фрагментации. Последний вариант возможной стратегии состоит в отказе от фрагментации отношения. Например, если отношение содержит небольшое количество кортежей, которые относительно редко обновляются. Вместо того чтобы попытаться выполнить горизонтальную фрагментацию этого отношения (например, по номеру отделения компании), имеет смысл оставить это отношение не фрагментированным и просто разместить на каждом из сайтов его реплицируемые копии. Это первый этап типовой процедуры определения схемы фрагментации (поиск отношений, которые не нуждаются в фрагментации). Затем следует проанализировать отношения, расположенные на единичной стороне связей типа «один ко многим», и подобрать для них оптимальные схемы фрагментации. На последнем этапе анализируются отношения, расположенные на множественной стороне тех же связей. Именно они чаще всего используются для применения производной фрагментации.

10. 6. ОБЕСПЕЧЕНИЕ ПРОЗРАЧНОСТИ В СУРБД.

В определении СУРБД утверждается, что система должна обеспечить прозрачность распределенного хранения данных для конечного пользователя. Под прозрачностью понимается сокрытие от пользователей деталей реализации системы. Например, в централизованной СУБД обеспечение независимости программ от данных также можно рассматривать как одну из форм прозрачности – в данном случае от пользователя скрываются изменения, происходящие в определении и организации хранения данных. Распределенные СУБД могут обеспечивать различные уровни прозрачности. Однако в любом случае преследуется одна и та же цель: сделать работу с распределенной базой данных совершенно аналогичной

работе с обычной централизованной СУБД. Все виды прозрачности, которые мы будем обсуждать ниже, очень редко можно найти в какой-либо одной СУРБД. Мы выделим четыре основных типа прозрачности, которые могут иметь место в системе с распределенной базой данных.

1) прозрачность распределенности;
2) прозрачность транзакций;
3) прозрачность выполнения;
4) прозрачность использования СУБД.

Прежде чем приступить к рассмотрению каждого из этих типов, следует отметить, что полная прозрачность не всегда принимается как одна из целевых установок.

Прозрачность распределенности. Прозрачность распределенности БД позволяет конечным пользователям воспринимать базу данных как единое логическое целое. Если СУРБД обеспечивает прозрачность распределенности, то пользователю не требуется каких-либо знаний о фрагментации данных (прозрачность фрагментации) или их размещении (прозрачность расположения).

Если пользователю необходимо иметь сведения о фрагментации данных и расположении фрагментов, то этот тип прозрачности мы будем называть прозрачностью локального отображения.

Прозрачность фрагментации является самым высоким уровнем прозрачности распределенности. Если СУРБД обеспечивает прозрачность фрагментации, то пользователю не требуется знать, как именно фрагментированы данные. В этом случае доступ к данным осуществляется на основе глобальной схемы и пользователю нет необходимости указывать имена фрагментов или расположение данных.

Прозрачность расположения представляет собой средний уровень прозрачности распределенности. В этом случае пользователь должен иметь сведения о способах фрагментации данных в системе, но не нуждается в сведениях о расположении данных.

Прозрачность расположения очень тесно связана с еще одним типом прозрачности – *прозрачность репликации*. Он означает, что пользователю не требуется иметь сведения о существующей репликации фрагментов. Под прозрачностью репликации подразумевается прозрачность расположения реплик. Однако могут существовать системы, которые не обеспечивают прозрачности расположения, но поддерживают прозрачность репликации.

Прозрачность локального отображения – самый низкий уровень прозрачности распределенности. При наличии в системе прозрачности локального отображения пользователю необходимо указывать как имена

используемых фрагментов, так и расположение соответствующих элементов данных.

11. ВВЕДЕНИЕ В СУБД ORACLE

11.1. ХАРАКТЕРИСТИКА СУБД ORACLE

СУБД Oracle – это современная система управления реляционными базами данных, поддерживающая работу в различных операционных средах. Система Oracle реализует самые современные технологии и поддерживает многие возможности, что позволяет характеризовать ее как достаточно мощную систему. Архитектура СУБД Oracle включает две важных части – ядро, которое является программным обеспечением, и словарь данных, который состоит из структур данных системного уровня, используемых ядром, управляющим базой данных. СУБД можно рассматривать как операционную систему, разработанную специально для управления доступом к данным; ее основные функции – хранение, выборка и обеспечение безопасности данных. Подобно операционной системе СУБД Oracle управляет доступом одновременно работающих пользователей БД к некоторому набору ресурсов. Подсистемы СУБД очень схожи с соответствующими подсистемами операционной системы и сильно интегрированы с представляемыми базовой ОС сервисными функциями доступа на машинном уровне к таким ресурсам, как память, центральный процессор, устройства ввода-вывода и файловые структуры. Подсистемы СУБД поддерживают собственный список авторизованных пользователей и их привилегий; управляют кэшем памяти и страничным обменом; блокировкой разделяемых ресурсов; использованием табличного пространства; принимают и планируют выполнение запросов пользователя. К основным функциям ядра СУБД Oracle, управляющего БД, относятся:

1) ввод-вывод;
2) управление памятью;
3) управление блокировками;
4) управление транзакциями;
5) контроль распределенных операций;
6) ведение журналов транзакций и восстановление базы данных;
7) управление хранением данных;
8) управление процессом;
9) поддержка языка управления данными;
10) защита информации.

СУБД Oracle является сложным программным продуктом. Поэтому для обеспечения работы пользователей с базой данных Oracle требуются специалисты, отвечающие за работу всей системы – *администраторы базы данных*. Например, в функции администратора базы данных входит:

1) создание экземпляра Oracle и его запуск;
2) создание начального варианта БД и планирование ее дальнейшего расширения;
3) регистрация пользователей в системе, назначение им привилегий, ролей и профилей;
4) отслеживание работы БД и принятие мер по оптимизации ее функционирования;
5) создание резервных копий БД и восстановление БД после сбоев.

11.2. ОБЪЕКТЫ БАЗЫ ДАННЫХ ORACLE

В базе данных Oracle содержатся различные типы объектов. Их можно разделить на две категории: объекты схемы и объекты, не принадлежащие схеме.

Схема – это набор объектов различной логической структуры данных. Каждая схема принадлежит пользователю БД и имеет одинаковое с ним имя. Схема может содержать следующие объекты:

1) таблицы (tables);
2) представления, или виды (views);
3) синонимы (synonyms);
4) последовательности (sequence);
5) индексы (indexes);
6) кластеры (clusters);
7) связи с БД (database links);
8) снимки (snapshots);
9) триггеры (triggers);
10) хранимые процедуры и функции (stored procedures and functions);
11) пакеты (packages).

К объектам, не принадлежащим схеме, но хранимым в БД, относятся профили, роли, пользователи, табличные пространства, сегменты отката, временные сегменты.

Дадим краткое описание объектов, принадлежащих схеме.

Таблицы представляют собой сегменты БД, в которых хранятся собственно данные. Каждая таблица состоит из строк (записей). Каждый столбец таблицы имеет имя и содержит данные одного типа. Информация о таблицах хранится в представлении DBA_TABLES словаря данных.

Представления – виртуальные таблицы, которые строятся на основе других таблиц и представлений, называемых в этом случае базовыми таблицами данного представления, в результате выполнения запроса. Так как представление не содержит никаких данных, то для него не выделяется физическая память на диске. При обращении к представлению этот запрос выполняется заново. Информация о представлениях хранится в представлении DBA_VIEWS словаря данных.

Синонимы – это псевдонимы или альтернативные имена объектов БД, которыми может быть таблица, представление, последовательность, процедура, функция, пакет или снимок. При создании синонима задается его имя и имя объекта, на который указывает синоним. Когда сервер Oracle встретит синоним в запросе, он автоматически заменит его названием объекта ссылки. Синонимы часто используются для удобства работы с данными, а также могут использоваться в целях обеспечения безопасности. Информация о синонимах хранится в представлении DBA_SYNONYMS словаря данных.

Последовательности – это объекты БД, которые используются для формирования уникальных числовых величин для столбца таблицы, который будет играть роль первичного ключа. Информация обо всех последовательностях хранится в представлении DBA_SEQUENCES словаря данных.

Индексы – это сегменты БД, созданные для ускорения поиска данных в определенной таблице. Индексы могут быть связаны с каждой таблицей или кластером. В индексах хранятся значения из одного или нескольких столбцов таблицы и значение ROWID – физического адреса строки, для каждого из хранимых значений столбца (столбцов). Для одной таблицы данных может быть создано несколько индексов, которые отличаются друг от друга набором или упорядоченностью столбцов этой таблицы. Существует несколько типов индексов: двоичный древовидный индекс, кластерный индекс, масочный двоичный индекс, который строится для небольшого диапазона значений.

Кластеры таблиц – это объекты БД, которые физически группируют совместно используемые таблицы в пределах одного блока данных. Кластеризация таблиц дает значительный эффект в том случае, если в системе приходится оперировать запросами, которые требуют совместной обработки данных из нескольких таблиц. В кластере таблицы хранятся ключ кластера (столбец, используемый для объединения таблиц) и значения из столбцов в кластеризованных таблицах. Поскольку кластеризованные таблицы хранятся в одном блоке БД, время на выполнение операций ввода-вывода заметно сокращается.

Связи с БД – это хранимые определения подключений к удаленным БД. Они используются при запросах к удаленным таблицам в распределенных БД.

Снимки представляют собой копии таблиц данных, полученные с удаленных БД в распределенных БД.

Триггеры – хранимые процедуры, написанные на языке PL/SQL, которые активизируются и выполняются в следующих случаях:

1) при модификации некоторой таблицы БД;
2) при создании, изменении или удалении объектов схемы БД.

Триггеры представляют собой удобное средство для обеспечения целостности и безопасности данных. Информацию о триггерах можно получить через представление DBA_TRIGGERS словаря данных.

Хранимые процедуры и функции представляют собой программы на языке PL/SQL, создаваемые пользователем и хранящиеся в БД. Могут запускаться как с помощью интерактивного редактора, так и с помощью других хранимых процедур и функций. Информация о хранимых процедурах и функциях содержится в представлениях DBA_OBJECTS и DBA_SOURCE словаря данных. Там же сохраняется и их исходный код.

Пакеты представляют собой совокупность процедур, переменных и функций, объединенных для выполнения некоторой задачи. Пакеты имеют заголовочную часть и тело. В заголовочной части описываются курсоры, исключительные ситуации, заголовки функций, процедур, переменные. В теле непосредственно реализуются функции и процедуры. Информация о пакетах хранится в представлениях DBA_OBJECTS и DBA_SOURCE словаря данных.

11.3. СЛОВАРЬ ДАННЫХ ORACLE

Словарь данных (Data Dictionary) представляет собой совокупность таблиц и представлений, содержащих всю справочную информацию (он хранит метаданные – данные о данных) обо всех объектах БД: таблицах, индексах, представлениях, триггерах, пакетах, процедурах и функциях. Иногда его называют каталогом системы. В словаре данных содержатся определения объектов БД, размеры выделенной памяти для каждого объекта схемы, описания фактического физического расположения объектов в памяти, список пользователей с указанием привилегий и ролей, ограничения целостности, значения столбцов по умолчанию и т. д. Все запросы к БД обрабатываются с использованием словаря данных. Он создается системой автоматически одновременно с БД, хранится в табличном пространстве SYSTEM и ведется ядром Oracle. Словарь данных доступен только в режиме чтения, причем для разных категорий пользо-

вателей доступны разные уровни. Все входящие в словарь таблицы и представления делятся на четыре вида:

1) внутренние таблицы СУБД (X$-таблицы);
2) таблицы словаря данных;
3) представления текущей активности (V$-представления);
4) представления словаря данных.

Рассмотрим подробнее их назначение.

Внутренние (*базовые*) *таблицы СУБД* – это таблицы, которые используются только самой системой Oracle. Они являются ключевым компонентом всей информационной структуры БД. Именно к ним обращается СУБД за всей внутренней информацией о текущем состоянии и процессах, происходящих в системе. Таблицы хранят информацию о БД в закодированном виде, что затрудняет их использование.

Таблицы словаря данных содержат информацию обо всех типах объектов, хранящихся в БД. Таблицы словаря данных имеют в конце имени знак доллара. Большую часть информации из них можно найти в представлениях словаря данных.

Представления текущей активности формируются и динамически изменяются в процессе работы СУБД Oracle. В них содержится огромное количество разнообразной информации о процессах, происходящих в БД, ее конфигурации и параметрах настройки большинства функций. Информация представлена в легкодоступной форме и может быть использована администратором БД для диагностики и настройки системы.

Представления словаря данных предназначены для просмотра информации из словаря данных пользователями. Они формируются на базе X$-таблиц и таблиц словаря данных. Пользователи получают доступ к представлениям словаря данных посредством операторов языка SQL. Большинство представлений словаря данных имеют префиксы USER_, ALL_, DBA_.

Представления с префиксом USER_ содержат информацию обо всех объектах, принадлежащих пользователю.

Представления с префиксом ALL_ содержат информацию обо всех объектах БД, к которым может получить доступ пользователь при выполнении запроса.

Представления с префиксом DBA_ содержат информацию обо всех объектах в БД и, следовательно, эти представления доступны только пользователю, который имеет доступ ко всем таблицам БД, в частности ими являются администраторы БД.

11.4. АРХИТЕКТУРА БАЗЫ ДАННЫХ ORACLE

Термин ***база данных Oracle*** используется для обозначения физической и логической структур данных совместно со всей служебной информацией. Чтобы разделить описание логической организации данных от способов их хранения и доступа к ним, в Oracle используется двухуровневая организация БД. Объекты верхнего (логического) уровня называются *логическими структурами*, а объекты нижнего (физического) уровня – *физическими структурами* БД.

Опишем ***физическую организацию*** БД Oracle. Физически БД Oracle организована как совокупность файлов, создаваемых обычными средствами операционной системы. Таким образом, основой физического уровня является файл.

Все компоненты физического уровня БД можно разделить на две большие группы – системные объекты, используемые внутри системы и необходимые СУБД для выполнения ее функций, и объекты пользователя. Системные файлы создаются и настраиваются администратором БД и не могут быть доступны пользователю в явном виде. В число системных объектов входят:

1) файл параметров инициализации;
2) управляющий файл;
3) файлы журнала регистрации транзакций;
4) файлы трассировки.

К пользовательским объектам физического уровня БД относятся файлы данных.

Файл параметров инициализации Init.ora содержит список параметров и ключей настройки, каждый из которых связан с определенной функцией системы или компонентом БД и представляет собой обычный текстовый файл. Этот файл считывается перед запуском БД, формировании экземпляра Oracle и считывании управляющих файлов. Значения параметров, которые заданы в Init.ora, определяют характеристики формируемого экземпляра Oracle и создаваемой БД. В частности, здесь заданы параметры распределения памяти для разделяемого пула, кэш-буфера данных, буфера журнала транзакций, характеристики автоматически запускаемых фоновых процессов, считываемых управляющих файлов, сегментов отката и т. д.

Управляющий файл содержит информацию о файлах данных и файлах журналов регистрации транзакций, о наборе символов, используемом для хранения данных, о статусе и датах обновления всех файлов данных, а также прочую аналогичную информацию. Большинство параметров, хранящихся в управляющем файле, формируются при создании БД. При отсутствии корректного управляющего файла БД не открывается, и ее

информация становится недоступной. Поэтому внутри СУБД предусмотрен поддерживаемый сервером Oracle механизм тиражирования управляющего файла. Рекомендуется иметь до трех копий управляющего файла.

Файлы журнала регистрации транзакций делятся на оперативные и архивные. *Оперативные* файлы журнала регистрации транзакций используются для записи содержимого буфера журнала транзакций. Эти журналы хранят сведения обо всех транзакциях, которые так или иначе связаны с изменениями содержимого БД и при необходимости могут быть использованы для восстановления БД. Минимальное количество оперативных файлов журнала регистрации транзакций равняется двум. Запись в них производится поочередно. Файл журнала регистрации транзакций, в который производится запись, называется активным. После заполнения одного файла журнала регистрации транзакций система переключается на следующий файл, а информация первого файла начинает перезаписываться в архивные файлы журнала регистрации транзакций. *Архивные* файлы журнала регистрации транзакций совместно с создаваемой регулярно копией БД позволяют полностью восстановить содержание базы, если произойдет искажение или потеря информации БД. В целях обеспечения бесперебойной работы системы создаются группы (как минимум две) оперативных файлов журнала регистрации транзакций, где каждая группа состоит из множества одинаковых элементов. Как только группа становится активной, очередная запись в журнале производится параллельно во все элементы группы.

В *файлах трассировки* регистрируются системные сообщения, ошибки и сведения обо всех главных событиях системы. Именно в этих файлах следует искать информацию при анализе причин возникновения той или иной нестандартной ситуации. Здесь фиксируются все критические сбои системы, а также сообщения о запуске или прекращении работы с БД, переключении файлов журнала транзакций и других событиях. Пользовательские и фоновые процессы могут сформировать свои собственные файлы трассировки.

Файлы данных предназначены для хранения как служебной, так и пользовательской информации. К служебной информации относятся словарь данных и другие служебные таблицы и представления, к пользовательской – объекты, созданные пользователем. Файл данных является обычным двоичным файлом операционной системы. Файлы данных формируются при создании или модификации табличных пространств, являющихся объектами логической структуры БД Oracle. При первоначальном создании файлы данных разбиваются сервером на *блоки Oracle*.

Информация обо всех файлах данных, составляющих физическое пространство БД, хранится в представлении словаря данных DBA_DATA_FILES.

Блоки Oracle являются элементами низшего в системе уровня иерархии хранения данных, представляя собой наименьшие адресуемые сервером Oracle единицы хранения (наименьшие единицы внешней памяти). Объемы объектов в БД и размеры блоков в кэш-буфере данных устанавливаются в блоках Oracle. Блоки Oracle нумеруются последовательно, начиная с единицы, для каждого файла данных. В словаре данных Oracle ведет список свободных блоков для каждого файла данных. Блоки Oracle формируются из блоков операционной системы. Размер блока Oracle должен быть кратен размеру блока данных, который используется операционной системой для ввода и вывода данных на диск. Размер блока Oracle устанавливается при создании базы данных и в дальнейшем не может быть изменен.

В каждом блоке Oracle предусмотрено место для заголовка, будущих обновлений данных блока и реально существующих строк данных. Заголовок содержит информацию о том, какому сегменту данных (какой таблице, индексу и т. д.) принадлежат строки, хранящиеся в блоке, о максимальном числе транзакций, которые могут быть одновременно адресованы к данным блока. Строки данных, помещенные в блок, нумеруются. Некоторая часть блока остается свободной. Размер свободной части блока задается параметром PCTFREE при создании таблицы и указывается в процентах. Это свободное пространство резервируется для возможного увеличения объема строк данных, хранящихся в блоке, в результате модификации.

Для более гибкой политики использования пространства блока используется параметр PCTUSED, также задаваемый при создании таблицы. Этот параметр указывает, какая часть объема блока должна освободиться, прежде чем блок будет вновь включен в список доступных для ввода новых строк. Освобождение пространства блока возможно в результате удаления строк и сокращения объема хранящейся в них информации после обновления данных.

Информационная часть блока содержит строки данных. При помещении очередной строки в блок для нее формируется внутренняя информационная структура Oracle – ROWID, представляющая собой физический адрес размещения строки. Этот адрес остается неизменным до удаления строки или реорганизации сегмента и имеет формат: BBBBBBBB.RRRR.FFFF, где: BBBBBBBB – шестнадцатеричный номер блока в файле данных, в котором находится строка; RRRR – шестнадца-

теричный номер строки в блоке; FFFF – шестнадцатеричный номер файла, содержащего блок.

Логическая структура БД Oracle состоит из пользовательских компонентов. Форма и назначение объектов логической структуры имеют смысл только в контексте сервера Oracle. К ним относятся табличные пространства (tablespace), сегменты (segments) и экстенты (extents).

Табличное пространство – это логический объект, который используется для группировки данных с целью организации более четкой структуры БД. В одно табличное пространство пользователь БД может поместить логические объекты, близко связанные между собой. Например, все таблицы, используемые одним и тем же приложением, могут быть объединены в одно табличное пространство. Физически табличное пространство реализовано совокупностью одного или нескольких файлов данных. Каждый файл данных размещен в одном табличном пространстве и хранит текущие данные этого пространства. Файлы данных табличного пространства создаются одновременно с созданием табличного пространства. При создании БД по умолчанию создается табличное пространство SYSTEM, содержащее файлы словаря данных и других служебных таблиц и представлений. Имена табличных пространств, за исключением SYSTEM, могут выбираться произвольно. Табличные пространства могут быть добавлены, удалены из БД. Кроме этого, добавляемые табличные пространства могут переводиться в автономное состояние (состояние блокировки) или в оперативное состояние (доступны для работы). Табличное пространство SYSTEM не может быть ни удалено, ни переведено в автономное состояние. При формировании объекта БД нужно указывать табличное пространство, которому оно будет принадлежать. После этого данные (которые, собственно, и образуют объект) будут храниться в файлах данных, принадлежащих указанному табличному пространству. Поэтому строки одной таблицы, например, могут храниться в нескольких файлах данных. Информация обо всех табличных пространствах БД и их статусе хранится в представлении словаря БД DBA_TABLESPACES.

Сегмент – это общее название для объектов информационной структуры, которые хранятся в БД, т. е. занимают место в файлах БД. Сегмент использует определенное число блоков Oracle, которые находятся в одном табличном пространстве, но могут принадлежать различным файлам. Объединение сегментов образует табличное пространство. Существует четыре типа таких объектов:

1) сегменты данных (data segments);
2) сегменты индексов (index segments);

3) сегменты отката (rollback segments);
4) временные сегменты (temporary segments).

Сегменты отката и временные сегменты создаются, как правило, системой. Сегменты данных и сегменты индексов – это объекты пользователя. Следует отметить, что многие объекты схемы имеют сегменты, располагаемые в табличных пространствах. Однако в БД есть ряд объектов пользователя, которые нельзя квалифицировать как сегменты. Это представления, последовательности, синонимы, связи, триггеры, хранимые функции и процедуры и пакеты. Информация о них хранится в словаре данных, но в БД они места не занимают.

При создании объекта ему выделяется некоторое пространство в указанном табличном пространстве – сегмент. Задавая имя табличного пространства, следует учесть объем имеющегося свободного места в табличном пространстве и возможность автоматического расширения файлов данных, составляющих табличное пространство. При увеличении объекта размер сегмента может увеличиться на заданный размер расширения сегмента, называемый экстентом.

Экстенты – это объекты информационной структуры хранения данных. Каждый сегмент БД состоит из одного или нескольких экстентов. Внешняя память для объекта БД выделяется порциями из определенного числа блоков. В файлах БД эти блоки должны быть смежными. Группа смежных блоков называется экстентом. Следует отметить, что память для объектов выделяется экстентами, которые могут находиться на разных дисках. После выделения экстента объекту эти блоки не могут быть использованы другими объектами БД. При создании объекта система Oracle автоматически распределяет в соответствующий сегмент начальный экстент для данного объекта. Если начальный сегмент полностью заполняется данными во время работы с объектом, то Oracle автоматически выделяет дополнительные экстенты для этого объекта. При удалении информации экстенты освобождаются, но остаются связанными с объектом (за исключением экстентов сегмента отката и временного сегмента) до тех пор, пока не будет выполнена операция удаления объекта. Например, если удалить все строки таблицы, то распределенные ей блоки все равно остаются закрепленными за этой таблицей. Таблицу нужно удалить (оператор DROP) или усечь (оператор TRUNCATE), чтобы освободить внешнюю память, выделенную таблице. Удаление объектов может привести к фрагментации. В системе Oracle устранением фрагментации занимается системный монитор SMON.

Кратко опишем назначение перечисленных типов сегментов.

Сегменты данных предназначены для хранения обычных таблиц и кластеризованных таблиц, следовательно, содержат строки таблиц данных. Экстенты для таблицы могут быть выделены из различных файлов, но эти файлы должны обязательно принадлежать одному табличному пространству. Одной таблице соответствует один сегмент.

Сегменты индексов служат для хранения индексов – это специальные таблицы, которые содержат информацию из ключевого столбца таблицы и идентификатор номера строки – ROWID.

Сегменты отката – это объекты информационной структуры БД. Они строятся системой и используются при выполнении транзакций. При модификации данных транзакцией их предыдущее состояние копируется в сегмент отката, а изменения выполняются в блоках, сохраняемых в кэш-буфере данных. Если другой запрос пользователя затребует эти данные, то они извлекаются из сегмента отката. Когда же результаты модификации считаются окончательно принятыми, соответствующий сегмент отката помечается как недействительный. Если транзакция завершается неуспешно, то информация из сегмента отката помещается назад в БД, и исходное состояние БД восстанавливается. Сегменту отката необходимо как минимум два экстента. Первый сегмент отката создается автоматически при создании БД, имеет имя SYSTEM и размещается в табличном пространстве SYSTEM. Сегменты отката также используются для восстановления БД после сбоев оборудования или отмены действий операторов модификации данных.

Временные сегменты создаются системой и используют пространство в файлах БД, чтобы создать временную рабочую область для промежуточных стадий обработки запроса, записанного на языке SQL, и для больших операций сортировки. Следующие операции могут приводить к созданию временных сегментов:

1) создание индекса;
2) использование фраз ORDER BY, DISTINCT или GROUP BY в операторе SELECT;
3) использование операторов работы с множествами UNION, INTERSECT, MINUS;
4) создание соединений таблиц;
5) использование некоторых типов подзапросов.

В целях более эффективной работы рекомендуется сегменты данных размещать в одном табличном пространстве, сегменты индексов – в другом, временные сегменты – в третьем и т. д.

11.5. АРХИТЕКТУРА ЭКЗЕМПЛЯРА БАЗЫ ДАННЫХ ORACLE

Экземпляр базы данных (Database Instance) Oracle представляет собой совокупность сложного комплекса взаимодействующих *процессов* и определенных *структур оперативной памяти*. Каждая БД Oracle имеет связанный с ней экземпляр, причем до тех пор, пока не будет использоваться опция Oracle Parallel Server, для БД формируется только один экземпляр. Организация экземпляра позволяет СУБД обслуживать множество типов транзакций, инициируемых одновременно большим количеством пользователей, и в то же время обеспечивать высокую производительность, целостность и безопасность данных.

Процессы экземпляра. Все процессы, работающие с БД Oracle, можно разделить на системные и пользовательские. ***Системные процессы*** Oracle делятся на две категории: серверные процессы и фоновые процессы. ***Пользовательские процессы*** запускаются пользователями БД, которые для осуществления доступа к БД используют прикладные средства Oracle, такие как, например, интерактивную среду SQL*Plus, генератор отчетов Oracle Reports, генератор форм Oracle Forms, или различные прикладные программы. Каждый процесс пользователя подключается к процессу сервера, который либо может быть жестко связан с одним процессом пользователя, либо разделяться между многими пользовательскими процессами. *Серверный процесс* анализирует и выполняет переданные ему операторы SQL и возвращает результаты пользовательскому процессу. Кроме того, серверный процесс считывает блоки данных из файлов данных и размещает их в кэш-буфере данных. *Фоновые процессы* Oracle работают в фоновом режиме. Они выполняют различные функции, необходимые для поддержания работы БД, а также осуществляют асинхронный ввод-вывод данных. Фоновые процессы делятся на обязательные и необязательные.

Обязательные фоновые процессы присутствуют в любой конфигурации экземпляра Oracle и к ним относятся четыре процесса.

1. *Процесс DBWR* (Database Writer), осуществляющий запись модифицированных блоков данных из кэш-буфера данных обратно в БД.
2. *Процесс LGWR* (Log Writer), осуществляющий запись информации из буфера журнала транзакций, расположенного в оперативной памяти, в оперативные файлы журнала транзакций.
3. *Процесс SMON* (System Monitor) – системный монитор, осуществляющий мониторинг экземпляра БД.
4. *Процесс PMON* (Process Monitor) – монитор процессов, контролирующий процессы экземпляра.

Опишем *необязательные фоновые* процессы.

1. *CKPT (Checkpoint Process)* – процесс контрольной точки. Он обрабатывает событие «контрольная точка», возникающее в системе при определенных условиях. Этот процесс может присутствовать в любой конфигурации экземпляра Oracle.

2. *ARCH (Archiver)* – процесс записи журнала архива. Он обеспечивает копирование оперативных файлов журналов транзакции в архивные файлы при их заполнении. Этот процесс может присутствовать в любой конфигурации экземпляра Oracle.

3. *Dnnn (Dispatcher), n* = 0, 1, …, 9, – процессы-диспетчеры. При обслуживании пользовательских процессов разделяемыми серверными процессами, одним или несколькими, они выполняют синхронизацию взаимодействия серверных и пользовательских процессов.

4. *RECO (Recoverer)* – процесс восстановления. Он отвечает за восстановление незавершенных транзакций в распределенной БД в конфигурация экземпляра Oracle Distributed.

5. *SNPn (Snapshots Process), n* = 1, 2, …, 9, A, …, Z, – эти процессы используются для получения снимков удаленных БД в случае распределенной БД в конфигурация экземпляра Oracle Distributed.

6. *LCKn (Parallel Server Lock Process), n* = 0, 1, …, 9, – процессы блокировки. Эти процессы в конфигурации экземпляра Oracle Parallel Server отвечают за координацию блокировок БД, устанавливаемых различными экземплярами Oracle.

7. *Pnnn, n* = 0, 1, …, 9, – процессы параллельных запросов. Эти процессы используются в конфигурации экземпляра Oracle Parallel Query для обслуживания параллельно выполняемых частей запросов.

Структуры памяти экземпляра. Каждому пользовательскому процессу выделяется область памяти, которая называется ***глобальной областью процесса*** (process global area) и сокращенно обозначается PGA. Содержимое PGA зависит от режима подключения пользовательского процесса к процессу сервера. Если пользовательский процесс взаимодействует с выделенным серверным процессом, то в PGA размещается информация о текущем сеансе работы пользователя, стек и информация о состоянии курсора. Информация о текущем сеансе, в свою очередь, включает данные, необходимые для системы обеспечения безопасности, и данные об используемых ресурсах. В стеке содержатся локальные переменные, а в области состояния курсора – текущая информация о положении курсора, возвращаемые строки и возвращаемый код курсора. Если же пользовательский процесс связан с разделяемым серверным процессом, то информация о текущем сеансе и текущем состоянии курсора хранится в глобальной системной области.

Кроме того, для всех процессов выделяется общая область памяти, которая называется ***глобальной областью системы*** (system global area – SGA). В SGA хранятся структуры памяти, необходимые для манипулирования данными, анализа предложений SQL и кэшированием транзакций. Эта область разделяемая, т. е. к ней одновременно имеет доступ множество процессов, которые могут считывать и модифицировать содержащиеся в ней данные.

Глобальная область системы состоит из следующих компонентов:

1) разделяемый пул (shared pool);
2) кэш-буфер базы данных (database buffer cache);
3) буфер журнала транзакций (redo log buffer).

Кратко опишем назначение каждого из этих разделов глобальной области системы.

Разделяемый пул кэширует информацию, используемую при разборе и выполнении операторов SQL. Разделяемый пул содержит два основных раздела: кэш библиотек и кэш словаря данных.

Кэш библиотек хранит текст SQL-выражений, форматы лексического анализатора и план выполнения предложений SQL. Кроме того, здесь же содержатся заголовки пакетов PL/SQL и процедур, которые могут совместно использоваться пользовательскими процессами. Сервер Oracle использует кэш библиотек для повышения скорости выполнения операторов SQL. Когда передается очередное SQL-выражение, сервер в первую очередь просматривает кэш в поисках такого же выражения, переданного ранее. Если оно найдено, то используется соответствующее ему дерево лексического анализа и план выполнения запроса, что избавляет от необходимости формировать их повторно. Кэш библиотек содержит и разделяемые области SQL, и локальные области SQL. Разделяемая область SQL включает дерево лексического анализа и план выполнения SQL-выражения, а локальная область – информацию, зависящую от текущего сеанса работы. Это могут быть присоединенные переменные, параметры окружения, стеки и буферы, необходимые при выполнении. Локальная область формируется для каждой инициируемой транзакции и освобождается после того, как закрывается соответствующий курсор. Используя эти структуры памяти, сервер Oracle может повторно использовать информацию, общую для всех выражений SQL, а информация, специфичная для данного сеанса, может быть выбрана из локальной области. Локальная область SQL делится в свою очередь на переходящую (persistent) область и область времени выполнения (runtime). При этом переходящая область содержит информацию, которая сохраняет свое значение и может быть использована несколькими выражениями SQL, а

область времени выполнения – только информацию для выражения, выполняемого в текущий момент.

Кэш словаря – хранит строки словаря данных, которые были использованы для лексического анализа SQL-выражений. В этой области находятся данные, касающиеся сегментирования, привилегий доступа и размеров свободной памяти. При запуске экземпляра он загружается некоторым начальным набором элементов и в процессе работы пополняется необходимыми данными из словаря.

Кэш-буфер данных состоит из буферов БД и хранит информацию, загружаемую из БД серверными процессами. Все модификации над данными реализуются в кэш-буфере. Имеется список, отслеживающий частоту обращений к хранящимся в кэш-буфере блокам данных. Сервер переносит данные на диск в соответствии с порядком их размещения в списке LRU (Least Recently Used, дословно – «наиболее давно использовавшиеся»). Этот список отслеживает обращение к блокам данных и учитывает частоту обращений. Когда выполняется очередное обращение к блоку данных, хранящемуся в кэш-буфере, он помещается в тот конец списка, который называется MRU (Most Recently Used – «только что использованные»). Если серверу требуется место в кэш-буфере для загрузки нового блока с диска, он обращается к списку LRU и решает, какой из блоков перенести на диск, чтобы освободить место для нового блока. Блоки, наиболее удаленные в списке от MRU, самые вероятные кандидаты на удаление из кэш-буфера. Таким образом, дольше всего остаются в кэш-буфере те блоки, к которым обращение выполняется чаще всего. Модифицированные блоки называются «грязными» (dirty) и помещаются в соответствующий dirty-список. В этом списке отслеживаются все модификации блоков данных, выполненные за время их нахождения в кэш-буфере и не зафиксированные на диске. Когда Oracle получает запрос на изменение данных, соответствующие изменения выполняются в области кэш-буфера, а сведения об изменениях в блоках заносятся в dirty-список; одновременно данные о выполненных операциях вносятся в журнал транзакций. В дальнейшем при обращении к блокам данных, попавшим в dirty-список, будут считываться уже модифицированные значения, хотя сами данные могут к этому времени еще не быть записаны на диск. Сервер использует отложенную многоблочную процедуру записи на диск с целью повышения производительности. Отложенная процедура означает, что обновление данных, выполненное Oracle, не фиксируется немедленно в дисковой памяти. Перенос этих dirty-блоков назад в БД осуществляется процессом DBWR при наступлении одного из определенных событий, таких как «контрольная точка», «выгрузка файла» и др.

Буфер журнала транзакций хранит данные о транзакциях до тех пор, пока они не будут переписаны в оперативный файл журнала транзакций. Это типичный циклический буфер – он заполняется от начала до конца, и затем новая информация снова записывается в начало буфера. После заполнения содержимое буфера процессом LGWR переносится в оперативный файл журнала транзакций. Для того чтобы гарантировать последовательный характер записи в буфер журнала, сервер Oracle управляет доступом к нему при помощи защелок (latch). Такая защелка представляет собой не что иное, как блокировку процессом Oracle некоторой структуры в памяти – аналогично блокировке файла или строки таблицы. Процесс блокирует посторонние обращения к памяти, выделенной для буфера журнала транзакций. Таким образом, если один процесс наложил защелку на буфер, другие не могут обратиться к нему до тех пор, пока защелка не будет снята. Сервер Oracle ограничивает количество транзакций, данные о которых заносятся в журнал одновременно.

11.6. ФОРМИРОВАНИЕ БАЗЫ ДАННЫХ И ЭКЗЕМПЛЯРА ORACLE

База данных Oracle открывается в три этапа.

1. Формирование экземпляра Oracle (предустановочная стадия).
2. Установка БД экземпляром (установочная стадия).
3. Открытие БД (стадия открытия).

Экземпляр Oracle формируется на предустановочной стадии запуска системы. На данной стадии считывается файл параметров Init.ora, запускаются фоновые процессы и инициализируется SGA. Этот файл определяет параметры конфигурации экземпляра, в частности размер структур памяти, количество и тип фоновых процессов. Имя экземпляра устанавливается в соответствии со значением переменной окружения Oracle_SID и необязательно должно совпадать с именем БД (но, как правило, совпадает). Следующая стадия установочная. Значения параметров управляющего файла из Init.ora определяют параметры БД, устанавливаемой экземпляром. Экземпляр Oracle создает табличное пространство SYSTEM, словарь данных, один сегмент отката и два оперативных файла журнала транзакций. На этой стадии доступ к управляющему файлу открыт и возможна модификация хранящихся в нем данных. На последней стадии открывается БД. Экземпляр получает исключительный доступ к файлам БД, имена которых хранятся в управляющем файле, и через него они становятся доступны пользователю. Если на втором или третьем этапах произойдет сбой, то формируется так называемый «пустой» экземпляр Oracle, который сам будет функционировать, но доступ к БД будет отсутствовать.

В процессе формирования могут быть получены экземпляры Oracle различной конфигурации:

1) экземпляр с типовой структурой;
2) конфигурация Parallel Server;
3) конфигурация Distributed;
4) конфигурация Parallel Query.

Рассмотрим подробнее различные конфигурации экземпляров.

Экземпляр с типовой структурой включает основные фоновые процессы и при необходимости к ним добавляются процесс архивации – ARCH и/или процесс контрольной точки – CKPT.

В обычной конфигурации одной БД соответствует один экземпляр Oracle и, наоборот, одному экземпляру Oracle должна соответствовать лишь одна БД. В ***конфигурации Parallel Server*** к одной и той же физической БД может быть подключено несколько экземпляров, что позволяет нескольким пользователями, расположенным на разных компьютерах, совместно использовать одну БД. В такой среде с параллельным обслуживанием дополнительно к основным фоновым процессам используется процесс блокировки LCKn, который отвечает за координацию блокировок БД, устанавливаемых различными экземплярами.

Конфигурация Distributed используется в случае распределенной БД. Она представляет механизмы развитого симметричного тиражирования, которые обеспечивают возможность распространять данные по отдельным экземплярам (по физическим БД) посредством снимков и отсроченных транзакций. Операции тиражирования планируются в очереди заданий и выполняются в фоновом режиме. В дополнение к процессам блокировки эта опция требует наличия по меньшей мере одного фонового процесса обновления снимка (SNP – snapshot) для выполнения работ, поставленных в очередь. Эта опция требует, чтобы фоновый процесс восстановления RECO завершал распределенные транзакции, которые закончились аварийно из-за отказов сети или экземпляра.

Конфигурация Parallel Query используется в том случае, если компьютер, на котором установлен экземпляр Oracle, имеет в наличии несколько процессоров. В этом случае дополнительно запускаются процессы Pnnn – процессы параллельных запросов. Процессы Pnnn используются для реализации параллельного выполнения отдельных частей запроса и принимают участие в формировании индексов и таблиц.

11.7. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ В ТИПОВОЙ КОНФИГУРАЦИИ ЭКЗЕМПЛЯРА ORACLE

Рассмотрим, что происходит при выполнении транзакции в типовой структуре экземпляра Oracle. Начинается транзакция в тот момент, когда пользователь подключается к серверу через драйвер SQL*Net. Это подключение может быть выделенным (dedicated) или разделяемым (shared). В первом случае происходит подключение к собственному процессу сервера, а во втором – к разделяемому серверному процессу через процесс-диспетчер. Фоновые процессы-диспетчеры, количество которых может доходить до десяти, выполняют синхронизацию взаимодействия серверных и пользовательских процессов. В простейшем случае все пользовательские процессы могут обслуживаться одним серверным процессом и одним процессом-диспетчером. Сервер хэширует поступившее от пользователя выражение SQL и сравнивает сформированный хэш-код с хэш-кодами, сохраненными в разделяемой области SQL ранее. Если в разделяемом пуле найдено точное совпадение выражений SQL, используются сформированные ранее дерево лексического разбора и план выполнения выражения. Если же совпадения не обнаружено, сервер выполняет разбор выражения.

Далее сервер проверяет, не находятся ли блоки данных, необходимые для выполнения транзакции соответственно поступившему выражению SQL, в кэш-буфере данных. Если блоки в буфере не обнаружены, сервер считывает их из файла данных и копирует в кэш. Если транзакция представляет собой запрос, сервер возвращает результат процессу пользователя. При этом чтение и копирование блоков данных выполняется столько раз, сколько потребуется для выборки всех затребованных данных. Если транзакция предусматривает модификацию данных, процесс несколько усложняется. В этом случае, после того как затребованные блоки данных будут считаны в кэш-буфер данных, модификация данных будет выполняться именно в кэш-буфере. Модифицируемые блоки кэша помечаются как «грязные» (dirty) и помещаются в dirty-список. Если возникает необходимость очередной модификации этих блоков данных, то они модифицируются прямо в кэш-буфере.

Если транзакция сравнительно коротка (например обновление одной строки с данными), она заканчивается и формируется сообщение пользователя, которое одновременно сигнализирует процессу LGWR о необходимости переписать информацию из буфера журнала транзакций в оперативный файл журнала. Если же транзакция довольно продолжительна, может произойти одно из следующих событий.

1. Информация о транзакции, которая записывается в кэш журнала, заполнила этот буфер на одну треть, что вынуждает процесс LGWR переписать содержимое буфера в файл.

2. Количество блоков, отмеченных в dirty-списке, достигло заданного значения. Это вынуждает процесс DBWR выполнить перезапись модифицированных блоков из кэш-буфера данных в файл данных, что в свою очередь заставляет процесс LGWR переписать содержимое буфера транзакций в файл.

3. Появилась контрольная точка БД. Это событие запускает уже описанную процедуру перезаписи модифицированных блоков данных в файлы и перезаписи буфера журнала транзакций в соответствующий файл.

4. Количество свободных блоков в кэш-буфере уменьшилось и достигло критической величины. Это также приводит к перезаписи данных из кэш-буфера в файлы.

5. Потребовалось освободить место в кэш-буфере данных для размещения считываемых новых блоков из БД. Это событие запускает уже описанную процедуру перезаписи модифицированных блоков данных в файлы и перезаписи буфера журнала транзакций в соответствующий файл.

6. Истекло время ожидания, заданное процессу DBWR, в течение которого он бездействовал. Это событие запускает уже описанную процедуру перезаписи модифицированных блоков данных в файлы и перезаписи буфера журнала транзакций в соответствующий файл.

7. Возникла невосстановимая ошибка БД. Это заставляет прекратить выполнение транзакций и запустить механизм отката. Соответственно формируется сообщение об ошибке, которое направляется серверу.

В процессе обработки транзакции формируется информация для журнала регистрации транзакций, которая время от времени перезаписывается из буфера журнала в оперативные файлы журнала. Со временем эти файлы заполняются информацией. Когда текущий файл заполнится, процесс LGWR начнет перезаписывать данные из буфера в другой файл, в то время как процесс ARCH копирует заполненный файл в архив на ленте или на диске. Поскольку транзакция считается завершенной только после того, как вся информация о регистрации транзакции будет переписана из буфера журнала в файл, для успешного завершения транзакции оба процесса, LGWR и ARCH, должны сработать без сбоев.

Перейдем к более подробному описанию этих процессов.

Процесс DBWR отвечает за перенос обновленных блоков, занесенных в dirty-список, из кэш-буфера данных в файлы данных. Вместо того что-

бы записывать каждый блок на диск сразу же после внесения изменений, процесс DBWR ждет, пока будет выполнено одно из определенных условий, а затем просматривает dirty-список и все отмеченные в нем блоки переписывает на диск. Такой подход позволяет уменьшить влияние скоростных характеристик дисковой памяти на производительность системы. Процесс DBWR выполняет перезапись информации из кэш-буфера данных на диск в следующих случаях:

1) обнаружена контрольная точка;
2) количество элементов в dirty-списке достигло заданной величины;
3) количество использованных буферов достигло величины, заданной параметром DB_BLOCK_MAX_SCAN из файла Init.ora;
4) истек заданный для процесса DBWR интервал времени в течение которого процесс DBWR бездействовал;
5) потребовалось место в кэше для размещения новых блоков данных.

Процесс LGWR отвечает за перезапись информации из буфера журнала транзакций, который находится в глобальной системной области, в оперативные файлы журнала транзакций. Эта запись выполняется при следующих условиях:

1) транзакция принимается;
2) процесс DBWR завершает перезапись данных из кэш-буфера после обнаружения контрольной точки;
3) буфер журнала транзакций заполняется на определенную величину (на одну треть);
4) истекает время ожидания, заданный временной интервал, в течение которого процесс LGWR бездействовал.

Важно отметить, что Oracle не считает транзакцию выполненной до тех пор, пока процесс LGWR не перезапишет данные о ней из буфера журнала транзакций в файл. Сообщение о выполнении транзакции передается процессу сервера не после изменения данных в файле данных, а после успешного завершения записи в журнал транзакций.

Процесс CKPT отвечает за обработку контрольных точек. Если этот процесс не запущен, то все операции по обработке контрольных точек выполняются процессом LGWR. Запуск специального процесса, обрабатывающего контрольные точки, повышает производительность системы. В контрольной точке модифицированные блоки буферов БД записываются в файлы БД на диск фоновым процессом DBWR. В контрольной точке обновляются все заголовки файлов данных и заголовки журналов обновлений, чтобы зафиксировать сам факт выполнения контрольной точки. В этом случае СУБД будет ожидать завершения процесса обработки контрольной точки, прежде чем в журналы транзакций смогут за-

писываться дальнейшие изменения БД. Событие «контрольная точка» возникает в следующих случаях:

1) происходит смена текущего журнала транзакций;

2) очередной том оперативного журнала транзакций заполнен до определенного объема;

3) истек заданный интервал времени, измеряемый в секундах, после последнего запуска процесса CKPT.

Процесс ARCH – процесс-архиватор, является вспомогательным фоновым процессом. Он отвечает за копирование полностью заполненного оперативного файла журнала транзакций в архивные файлы журнала транзакций, освобождая эти файлы для дальнейшего использования. Процесс автоматически активизируется после заполнения тома журнала транзакций. Необходимость в такой процедуре возникает только при работе в режиме архивирования журнала транзакций. Во время перезаписи в архив никакой другой процесс не может получить доступ к журналу. Таким образом, на время копирования в архив система должна находиться в состоянии ожидания.

Процесс PMON – монитор процессов выполняет автоматическую «уборку» после внезапно прекратившихся или завершившихся аварийно пользовательских процессов. Эта «уборка» предусматривает удаление сеанса, закрепленного за прекратившимся процессом, блокировок, которые были им установлены, а также освобождение ресурсов глобальной системной области, выделенных этому процессу. При наличии незавершенных транзакций производится откат к началу транзакций. Кроме того, PMON следит за процессами сервера и диспетчера и автоматически перезапускает их в случае останова.

Процесс SMON – системный монитор, осуществляющий мониторинг экземпляра Oracle. Основные задачи этого процесса следующие.

1. Если две транзакции ждут, пока одна из них снимет блокировки и ни одна не может продолжаться (это называется взаимоблокировкой), SMON распознает ситуацию и один из процессов получает сообщение о том, что возникла взаимоблокировка.

2. Системный монитор освобождает временные сегменты, которые более не используются создавшими их пользовательскими процессами.

3. Следит за свободным пространством в файлах БД, ведет учет освобождающихся блоков, автоматически выделяя их под нужды системы.

4. Если во время работы экземпляра происходит сбой, и экземпляр оказывается не в состоянии продолжать работу, то системный монитор производит перезапуск экземпляра. Автоматически восстанавливает при

запуске ненормально остановленный экземпляр (если нет потерянных файлов).

5. При простое СУБД SMON дефрагментирует свободное пространство в файлах БД, подготавливая распределение внешней памяти под новые объекты или для расширения существующих объектов БД.

6. После фиксации транзакции изменения заносятся в оперативные файлы журнала транзакций, даже если измененные блоки БД все еще находятся в буфере SGA. Процесс SMON может всегда повторно выполнить зафиксированные в журнале транзакций изменения в файлах БД при внезапной остановке компьютера и потере содержимого оперативной памяти.

12. ОСНОВЫ ЯЗЫКА SQL

12.1. АЛФАВИТ И ЛЕКСЕМЫ ЯЗЫКА SQL

Язык SQL (Structured Query Language – язык структурированных запросов) является непроцедурным языком и ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Формулируя запросы на языке SQL, можно создавать и модифицировать различные объекты БД и, оперируя группами строк, вставлять, выбирать, обновлять и удалять данные из таблиц. Кроме того, он позволяет управлять доступом к объектам БД и обеспечивать непротиворечивость и целостность данных, хранящихся в базе.

Алфавит языка включает следующие символы:

1) буквы: A..Z, a..z;
2) цифры: 0..9;
3) символы: + – * / ! @ $ # = < > ^ ' “ ( ) | _ ; , .

Идентификаторы и комментарии. Идентификаторы, длина которых может достигать 30 символов, обычно начинаются с буквы и могут включать в себя также цифры, символы $ и #, символ подчеркивания. Исключение составляют имена базы данных (до восьми символов) и удаленные имена. В некоторых версиях системы Oracle допускается использование русских букв. Имя любого объекта может состоять из нескольких частей: [схема.]объект[@dblink]. Схема представляет собой набор объектов разной структуры, принадлежащих конкретному пользователю, и идентифицируется его именем. Среди объектов схемы могут быть таблицы, представления (виртуальные таблицы), индексы, последовательности, триггеры, процедуры и функции. @dblink – это удаленное имя таблицы или представления БД.

Допускается использование однострочных и многострочных комментариев. Однострочные комментарии представляют собой следующую конструкцию:

-- текст комментария

Многострочные комментарии имеют следующий вид:

/* текст комментария */

Литералы. Символьные литералы определяются как тип CHAR и записываются в одинарных кавычках: 'test'. При необходимости присутствия одинарной кавычки внутри символьного литерала она удваивается.

Числовые литералы определяются как тип NUMBER и представляют собой целые или действительные значения со знаком или без знака, при этом действительные значения могут быть записаны в формате с десятичной точкой или в экспоненциальной форме.

Пустые значения. В языке SQL имеется специальное предопределенное значение NULL, которое расценивается как неопределенное значение. Оно не эквивалентно понятию «пустая строка» для символьных типов и не эквивалентно нулевому значению для числовых типов. Если в некотором столбце таблицы данные отсутствуют, говорят, что его значение NULL. Столбец с данными любого типа может содержать значение NULL, если только он специально не описан как NOT NULL.

Псевдостолбцы. Это формируемые системой столбцы, имеющие стандартные имена. Их значения можно только просматривать и использовать, но корректировать (добавлять, удалять, изменять) нельзя.

К ним относятся: ROWID, ROWNUM, LEVEL, CURRVAL, NEXTVAL.

Псевдостолбец ROWID содержит уникальные для всей БД физические адреса строк таблицы. Значение псевдостолбца ROWID определяется при вставке строки в таблицу и не изменяется, пока строка присутствует в таблице.

Псевдостолбец ROWNUM определяет порядковый номер строки, выбранной при выполнении запроса. Он обычно используется для ограничения числа строк, выбираемых из таблицы.

Псевдостолбец LEVEL возвращает уровень вложенности данных, позволяя тем самым строить запросы для получения информации об иерархии данных.

Для работы с последовательностями генерируемых значений, используемых в качестве уникальных ключей, имеются псевдостолбцы:

1) имя_последовательности.CURRVAL – возвращает текущее значение из указанной последовательности генерируемых значений;

2) имя_последовательности.NEXTVAL – возвращает следующее значение из указанной последовательности генерируемых значений.

Предварительно последовательность с именем имя_последовательности должна быть создана с помощью оператора CREATE SEQUENCE.

12.2. ТИПЫ ДАННЫХ ЯЗЫКА SQL

Наиболее часто используются следующие типы данных: символьные, числовые, тип DATE, двоичные и большие объекты.

Символьные типы данных представлены типами CHAR (длина), VARCHAR2 (длина) и LONG.

Тип данных CHAR представляет собой символьные строки фиксированной длины. Минимальная длина равна 1, максимальная – 2000 байт. Если значение, помещаемое в столбец данного типа, превосходит указанный размер, то выводится сообщение об ошибке; если длина помещаемого значения меньше указанной длины, то значение дополняется пробелами справа.

Тип данных VARCHAR2 представляет собой символьные строки переменной длины. Максимальный размер строки 4000 байт, минимальный – 1 байт. При помещении текста в столбец большего размера дополнение пробелами не производится.

Строки этих двух типов сравниваются по-разному. Строки типа CHAR – посимвольно с дополнением пробелами строки с меньшей длиной до размера строки с большей длиной. Строки VARCHAR2 – без дополнения пробелами до большей длины. Поэтому для двух в принципе одинаковых строк могут быть получены различные результаты при их сравнении.

П р и м е р. Для двух строк 'AB' и 'AB ' (вторая строка содержит пробел, а первая нет) типа CHAR получим, что 'AB' = 'AB '. Для этих же строк типа VARCHAR2 результатом сравнения будет 'AB' < 'AB '.

Тип данных LONG представляет собой символьные данные переменной длины, величина которой может достигать 2 Гб. На использование переменных этого типа накладывается ряд ограничений: столбец такого типа должен быть единственным в таблице, его нельзя индексировать, использовать в качестве ключа упорядочения и в операциях группирования, а также для построения условий.

Числовые типы данных представлены типом NUMBER, который позволяет определить три различных типа данных:

NUMBER, NUMBER(p), NUMBER(p, s).

В первом случае определяются действительные числа, диапазон которых от $1.0*10^{-130}$ до $1.0*10^{126}-1$, мантисса содержит 38 знаков. Во втором случае считается, что определяется диапазон целых чисел, где p – количество цифр в числе (от 1 до 38). В третьем случае описываются числа с фиксированной точкой; p – общее количество цифр (от 1 до 38), s – масштаб (от –84 до 127) – определяет количество цифр после запятой. Если $s > 0$, то число округляется до указанного числа знаков справа от десятичной точки, если $s < 0$, то число округляется до указанного числа знаков слева от десятичной точки.

П р и м е р**.** Значение 123.89, помещенное в переменную с типом NUMBER(5,1) будет округлено до значения 123.9, а при помещении в переменную с типом NUMBER(5, –1) будет округлено до 120.

Следует отметить, что язык SQL поддерживает типы данных стандарта ANSI SQL. Если такой тип данных (INTEGER, SMALLINT, DECIMAL, FLOAT и REAL и др.) встречается при определении типа столбца таблицы, то имя типа сохраняется, но сами данные хранятся в виде, определяемом одним из типов БД Oracle.

Тип данных DATE представляет собой специальным образом организованный тип. Он хранит столетие, год, месяц, день, часы, минуты, секунды. Для выборки текущего дня и времени в стандартном формате используется функция SYSDATE. Стандартный формат даты, автоматически преобразуемый во внутреннее представление, записывается символьной строкой следующего вида: 'DD–MM–YY', где DD – день месяца, MM – значение месяца, а YY – значение года.

П р и м е р. Строка символов '11–01–04' будет представлять дату 11 января 2004 г.

Над переменными типа DATE можно выполнить арифметическое действие – вычитание. Результат операции определяет количество дней между этими двумя датами. К дате можно прибавить или отнять числовую константу, рассматриваемую как количество дней или часть дня.

П р и м е р. Для добавления месяца к текущей дате необходимо записать SYSDATE+30, а для добавления часа – SYSDATE+1/24.

Двоичные типы данных используются для хранения двоичных неструктурированных данных (звуковые файлы, файлы изображений и др.), обработка которых не поддерживается языком. К ним относятся типы RAW(длина) и LONGRAW. Максимальный размер строки типа RAW 2000 байт, минимальный – 1 байт. Длина переменных типа LONGRAW может достигать 2 Гб.

Большие объекты (LOB-объекты) представлены типами CLOB, BLOB и BFILE и предназначены для хранения неструктурированных

данных большого объема – до 4 Гб. Тип данных CLOB хранит данные символьного типа, типы данных BLOB и BFILE используют для хранения двоичных данных. Столбцы типа LOB содержат не сами данные, а указатели на их местоположение. При этом типы CLOB и BLOB хранятся в специальных сегментах БД, а тип BFILE, являющийся внешним двоичным файлом, хранится как обычный файл. На их использование наложен ряд ограничений.

12.3. ОПЕРАТОРЫ ЯЗЫКА SQL

Все операторы SQL делятся на группы:

1) операторы языка описания данных – DDL;
2) операторы языка манипулирования данными – DML;
3) операторы управления транзакциями;
4) операторы управления сеансом;
5) операторы управления системой.

К ***операторам DDL*** относятся следующие:

CREATE, ALTER, DROP, GRANT, REVOKE.

Оператор CREATE используется для создания объектов БД. К ним относятся:

1) TABLE – таблица базы данных;
2) VIEW – представление или вид; представляет собой виртуальную таблицу, которая строится на основе выбранной в результате выполнения запроса информации из одной или нескольких таблиц;
3) SEQUENCE – последовательность, последовательно выбираемые значения которой можно использовать для задания уникальных значений ключа;
4) INDEX – индекс, используемый для обеспечения более быстрого доступа к данным таблицы;
5) TRIGGER – хранимая в БД программная единица, запускаемая автоматически при наступлении определенных событий;
6) FUNCTION – хранимая в БД программная единица, вызываемая пользователем или другими программными единицами для выполнения;
7) PROCEDURE – хранимая в БД программная единица, вызываемая пользователем или другими программными единицами для выполнения;
8) USER – имя пользователя, имеющего доступ к информации БД;
9) ROLE – совокупность определенных привилегий, обеспечивающих возможность создания, удаления и модификации объектов БД.

Оператор ALTER используется для изменения объектов БД. Применяется по отношению ко всем перечисленным выше объектам.

Оператор DROP применяется для удаления всех вышеперечисленных объектов из БД.

Оператор GRANT позволяет наделить роль или пользователя различного вида привилегиями или ролями.

Оператор REVOKE отменяет предоставленные пользователям или ролям привилегии и роли.

К группе ***операторов DML*** относятся:

INSERT, DELETE, UPDATE, SELECT.

Первые три оператора позволяют осуществить соответственно вставку, удаление и модификацию строк таблиц. Оператор SELECT предназначен для построения запросов, в результате выполнения которых из таблиц выбирается вся необходимая информация. Запрос на выборку информации, включенный в запись некоторого другого оператора, образует подзапрос.

В группу ***операторов управления транзакциями*** входят следующие операторы:

1) COMMIT – фиксация текущей транзакции;
2) ROLLBACK – откат текущей транзакции.

Транзакция представляет собой неделимую с точки зрения системы единицу работы, выполняемую системой. В случае успешного выполнения всех входящих в транзакцию действий ее результаты фиксируются (COMMIT). В противном случае состояние БД можно вернуть в исходное состояние, отменив транзакцию (ROLLBACK).

Операторы управления сеансом работы изменяют установки, задаваемые для сеанса работы в БД (ALTER SESSION, SET ROLE).

Операторы управления системой изменяют установки всей БД (ALTER SYSTEM).

12.4. ОПЕРАЦИИ ЯЗЫКА SQL

Операции языка SQL делятся на ряд групп.

Арифметические операции представлены в свою очередь двумя группами операций:

1) унарные +, –
2) бинарные +, –, *, / .

Арифметические операции используются в выражениях для изменения знака операнда, сложения, вычитания, умножения и деления числовых величин. Унарные операции оперируют только с одним операндом, бинарные требуют при своем использовании два операнда.

В группе ***операций над строками*** имеется только одна операция – операция сцепления строк. Для ее обозначения используется комбинация двух символов вертикальная черта (||).

Операции сравнения применяются в основном в операторах DML при построении простых условий проверки для сравнения значения одного выражения со значением другого выражения. Результатом сравнения может быть либо TRUE, либо FALSE, либо UNKNOWN. Значение UNKNOWN может появиться в результате сравнения значений двух выражений, если одно из них или оба имели значение NULL. Над значениями двух выражений X и Y могут быть выполнены следующие операции сравнения:

1) X = Y – проверка значений выражений X и Y на равенство; результат равен TRUE, если указанное соотношение выполняется;

2) X != Y, X<>Y, X ^= Y – проверка значений выражений X и Y на неравенство; результат равен TRUE, если указанное соотношение выполняется;

3) X < Y, X > Y, X >= Y, X <= Y – проверка значений выражений X и Y на соотношение «меньше, чем», «больше, чем», «больше или равно», «меньше или равно»; результат равен TRUE, если указанное соотношение выполняется;

4) X [NOT] BETWEEN A AND B – проверка, (не) находится ли значение выражения X в указанном диапазоне, определяемом значениями выражений A и B; результат равен TRUE, если указанное соотношение выполняется;

5) X IN (список выражений | подзапрос) – проверка значения выражения X на равенство некоторому элементу из списка значений выражений или множества значений, возвращенных подзапросом; результат равен TRUE, если указанное соотношение выполняется хотя бы для одного элемента списка выражений или множества значений, возвращенных подзапросом;

6) X NOT IN (список выражений | подзапрос) – проверка значения выражения X на неравенство ни одному элементу из списка значений выражений или множества значений, возвращенных подзапросом; результат равен TRUE, если указанное соотношение выполняется для всех элементов списка выражений или множества значений, возвращенных подзапросом;

7) X LIKE Z – проверка значения выражения X на подобие; результат проверки равен TRUE, если X совпадает с шаблоном Z. Шаблон представляет собой символьную строку, внутри которой символ '%' используется для сопоставления с любой строкой из нуля или более символов,

кроме NULL – строки, а символ подчеркивания (_) сопоставляется с любым одиночным символом;

8) X IS [NOT] NULL – проверка значения выражения X на (не) пустое значение NULL; результат равен TRUE, если указанное соотношение выполняется;

9) операция сравнения с квантором ANY позволяет сравнивать проверяемое значение со всеми элементами из заданного списка значений выражений или множества значений, возвращенных подзапросом; результат проверки равен TRUE, если указанная операция сравнения (=, !=, >, <, >=, <=) выполняется хотя бы для одного элемента списка выражений или множества значений, возвращенных подзапросом;

10) операция сравнения с квантором ALL позволяет сравнивать проверяемое значение со всеми элементами из заданного списка значений выражений или множества значений, возвращенных подзапросом; результат проверки равен TRUE, если указанная операция сравнения (=, !=, >, <, >=, <=) выполняется для всех элементов списка выражений или множества значений, возвращенных подзапросом;

11) операция сравнения EXISTS проверяет результат выполнения подзапроса; результат проверки равен TRUE, если подзапрос возвращает не пустое множество значений.

Логические операции представлены стандартными логическими операциями: NOT, AND, OR, используемыми при построении сложных условий проверки, в которых простые условия объединяются в более сложное условие с помощью логических операций.

Логические операции выполняются в трехзначной логике, которая задается следующими таблицами истинности:

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

NOT	True	False	Unknown

	False	True	Unknown

Операции над множествами позволяют выполнить определенные действия над выбираемыми в результате выполнения одного или нескольких запросов группами строк. Естественно, что структуры этих строк должны совпадать по количеству, порядку расположения и типу данных входящих в них элементов. К ним относятся следующие операции:

1) UNION ALL – объединяет все строки, извлеченные одним или несколькими запросами, включая повторяющиеся;

2) UNION – объединяет все строки, извлеченные одним или несколькими запросами, с устранением дублирующих строк;

3) INTERSECT – объединяет только те строки, которые присутствуют в результатах выполнения каждого из запросов, с устранением дублирующих строк;

4) MINUS – объединяет все неповторяющиеся строки, извлеченные первым запросом, но не извлеченные вторым.

Класс ***других операций*** содержит две операции: операцию внешнего соединения (+) и специальную операцию PRIOR.

Операция внешнего соединения используется при выборе информации из нескольких таблиц в том случае, если из одной таблицы необходимо выбрать все строки, а из остальных таблиц только те строки, для которых выполняются определенные условия.

Операция PRIOR устанавливает взаимосвязь между родительскими и дочерними строками при построении иерархических запросов.

12.5. ФУНКЦИИ ЯЗЫКА SQL

Рассмотрим наиболее часто используемые группы функций языка SQL.

Числовые функции предназначены для вычисления степени числа, абсолютного значения, округления и усечения числа с заданной точностью, вычисления тригонометрических значений. Опишем некоторые числовые функции.

1. Функция ABS(n) возвращает абсолютное значение аргумента *n*, имеющего числовой тип.

2. Функция ROUND(n, [r]) осуществляет округление значения аргумента *n*, имеющего числовой тип, с точностью до количества указанных знаков *r*. При этом если значение *r* положительно, то округление производится до указанного количества знаков после запятой, если значение *r*

отрицательно, то округление производится до указанного количества знаков до запятой. При $r = 0$ функция возвращает округленную целую часть аргумента n.

3. Функция MOD(m, n) возвращает остаток от деления целочисленного аргумента *m* на целочисленный аргумент *n*.

4. Функция POWER(m, n) возвращает аргумент *m*, имеющий числовой тип, возведенный в степень, заданную аргументом *n*, имеющим также числовой тип.

5. Функция SQRT(m) возвращает квадратный корень из аргумента *n*, имеющего числовой тип.

Символьные функции предназначены для работы со строками. Они могут возвращать либо строку, либо целое значение. Ниже приводится описание некоторых символьных функций.

1. Функция UPPER(str) возвращает строку str, все символы которой преобразованы в верхний регистр.

2. Функция LENGTH(str) возвращает длину строки str в символах.

3. Функция SUBSTR(str, n, m) выделяет из строки str подстроку длины *n*, начиная с символа в позиции *m*.

4. Функция LPAD(str, n, chr) возвращает строку str, дополненную слева указанным символом chr до указанной длины *n*.

5. Функция RPAD(str, n, chr) возвращает строку str, дополненную справа указанным символом chr до указанной длины *n*.

Функции работы с датами предназначены для работы с данными типа DATE. Ниже описываются некоторые функции этой группы.

1. Функция ADD_MONTHS(data, n) добавляет к указанной дате (аргумент data) или вычитает из нее *n* месяцев.

2. Функция MONTHS_BETWEEN(data1, data2) возвращает количество месяцев, находящихся между указанными датами data1 и data2.

3. Функция LAST_DAY(data) возвращает последний день месяца, указанного датой data.

Функции преобразования типа в основном используются для преобразования данных символьного типа в числовой или в тип DATE и, наоборот, для преобразования данных числового типа или типа DATE в символьный тип. Преобразование осуществляется в соответствии с задаваемым форматом. Формат преобразования имеет вид символьной строки, где каждый символ или группа символов имеет определенное назначение.

1. Функция TO_CHAR(d1, [fmt]) преобразует значение d1 типа DATE в значение типа VARCHAR2 по формату fmt.

2. Функция TO_CHAR(n1, [fmt]) преобразует значение n1 типа NUMBER в значение типа VARCHAR2 по формату fmt.

3. Функция TO_DATE(char, [fmt]) преобразует значение char типа CHAR или VARCHAR2 в значение типа DATE по формату fmt.

4. Функция TO_NUMBER(char, [fmt]) преобразует значение char типа CHAR или VARCHAR2 в значение типа NUMBER по формату fmt.

В форматах для даты используются следующие группы символов:

1) DD – задает номер дня месяца в диапазоне от 1 до 31;
2) DAY – задает полное название дня недели;
3) MON – задает краткое название месяца;
4) MONTH – задает полное название месяца;
5) YY – задает две последние цифры номера календарного года;
6) YYYY – задает полный номер календарного года.

В форматах для чисел используются следующие символы:

1) цифра 9 задает цифру;
2) символ точка (.) задает десятичную точку;
3) цифра 0 задает обязательный нуль;
4) буква s задает обязательное наличие знака {+; –};
5) символ $ задает знак доллара, проставляемый в начале числа.

Групповые функции выполняют операции над группами строк.

1. Функция COUNT({*}) – возвращает количество строк в группе.

2. Функция COUNT([DISTINCT] выражение) – возвращает количество строк в группе, игнорируя значение NULL.

3. Функция SUM([DISTINCT] выражение) – возвращает сумму значений указанного выражения для группы строк или списка значений, игнорируя значение NULL.

4. Функция AVG([DISTINCT] выражение) – возвращает среднее значение указанного выражения для группы строк или списка значений, игнорируя значение NULL.

5. Функция MIN([DISTINCT] выражение) – возвращает минимальное из значений указанного выражения для группы строк или списка значений, игнорируя значение NULL.

6. Функция MAX([DISTINCT] выражение) – возвращает максимальное из значений указанного выражения для группы строк или списка значений, игнорируя значение NULL.

7. Фраза DISTINCT предписывает групповым функциям рассматривать только различные значения выражения.

Выполняются также ***другие функции.*** Относящаяся к этой группе функция NVL(выражение1, выражение2) обрабатывает пустое значение NULL. Если значение выражения1 равно NULL, то функция возвращает

значение выражения2; если же значение выражения1 не равно NULL, то функция возвращает значение выражения1.

12.6. СОЗДАНИЕ, МОДИФИКАЦИЯ И УДАЛЕНИЕ ТАБЛИЦ

Рассмотрим основные функции, выполняемые над таблицами.

Создание таблицы выполняется с помощью оператора CREATE, упрощенный синтаксис которого имеет следующий вид:

```
CREATE TABLE имя_таблицы
{({ имя_столбца тип_данных [DEFAULT значение]
[ограничения_столбца] | ограничение_таблицы}
[,{ имя_столбца тип_данных [DEFAULT значение]
[ограничения_столбца] | ограничение_таблицы}]...) |
AS подзапрос};
```

Таблица может быть создана либо стандартным образом через описание ее компонентов, либо в результате выполнения некоторого подзапроса. Подзапрос – это обычный запрос на выборку информации, реализуемый оператором SELECT. При создании таблицы задаваемые имена таблиц и имена столбцов должны удовлетворять правилам, предписываемым идентификаторам. При этом естественно, что имена, присваиваемые таблицам, должны быть уникальными в схеме пользователя, а имена столбцов – в рамках одной таблицы. Для каждого столбца указывается тип данных и, если необходимо, значение, вставляемое в столбец по умолчанию (DEFAULT значение). Важным элементом при создании таблиц является задание ***ограничений целостности*** данных, которые позволяют отслеживать правильность модификации имеющихся данных или вставляемых в таблицу новых данных. Ограничения целостности данных делятся на два типа: ограничения столбца и ограничения таблицы. Ограничения столбца позволяют определить условия, которым должны удовлетворять значения соответствующего столбца; ограничения целостности данных, накладываемые на таблицу, позволяют проверить правильность всех добавляемых или модифицируемых строк таблицы. Ограничение может быть именованным или безымянным. Удобнее использовать именованные ограничения, поскольку при выдаче информации, связанной с возникшим нарушением одного из ограничений, выдается и имя этого ограничения, что весьма полезно для исправления ошибок в дальнейшем. Задание ограничений на столбец или ограничений на таблицу осуществляется по следующему синтаксису:

[CONSTRAINT имя_ограничения] тип_ограничения

Имеются следующие типы ограничений, накладываемых на столбец:

1) PRIMARY KEY – это ограничение требует, чтобы вводимые в столбец значения были уникальными и отличными от пустых, поскольку они будут использоваться в качестве первичного ключа, однозначно идентифицирующего запись; первичный ключ определяется для таблицы только единожды;

2) UNIQUE – это ограничение требует, чтобы вводимые в столбец значения в рамках одной таблицы были уникальными;

3) NOT NULL – это ограничение требует обязательного присутствия в столбце некоторого значения;

4) CHECK(выражение) – это ограничение позволяет подвергнуть определенной проверке вставляемое в столбец значение; если условия, наложенные на вставляемые значения, не выполняются, то значения в столбец не помещаются;

5) REFERENCES – это ограничение позволяет установить взаимосвязь значений данного столбца со значениями столбца другой таблицы. Взаимосвязь обеспечивается использованием следующей конструкции:

REFERENCES имя_таблицы[(имя_столбца)] [ON DELETE CASCADE]

При внесении значения в столбец создаваемой таблицы система будет автоматически проверять наличие аналогичного значения в указанном столбце указанной таблицы. При этом естественно, что для обеспечения однозначности устанавливаемой взаимосвязи все значения, находящиеся в указанном столбце, на которые производится ссылка, должны иметь ограничение UNIQUE или PRIMARY KEY. Если в качестве имени столбца указанной таблицы используется первичный ключ, то имя столбца можно не указывать. Таблица, на чей столбец ссылается другая таблица, называется главной, а таблица, ссылающаяся на нее, – подчиненной. Конструкция ON DELETE CASCADE указывает, что при удалении строк в главной таблице автоматически осуществляется удаление соответствующих строк и в подчиненной таблице.

Ограничения на таблицу во многом напоминают ограничения столбца, но при этом обычно задействуют, как правило, несколько столбцов. Например, можно задать ограничение PRIMARY KEY, указав список имен столбцов, тем самым определив ***составной первичный ключ***. При этом для столбцов, указываемых в списке, должны быть заданы ограничения UNIQUE и NOT NULL.

Используя следующую форму записи:

FOREIGN KEY (список_имен_столбцов)
REFERENCES имя_таблицы(список_имен_столбцов)

[ON DELETE CASCADE]

можно определить ***составной внешний ключ*** для таблицы. Естественно, что в случае составного внешнего ключа перечень столбцов в подчиненной таблице и перечень столбцов в главной таблице должны совпадать по количеству, типу данных и порядку следования. Конструкция ON DELETE CASCADE позволяет обеспечить целостность и непротиворечивость данных при изменениях, которые затрагивают значения столбцов главной таблицы, являющихся внешним ключом по отношению к подчиненной таблице.

Если ограничение CHECK затрагивает значения нескольких столбцов, увязывая их в некоторое достаточно сложное условие, то такое ограничение также удобно определить как ограничение на таблицу.

Для генерации и вставки в столбец таблицы уникальных значений можно создать специально предназначенный для этих целей объект – последовательность. Создание последовательности выполняется с помощью оператора CREATE по следующему упрощенному синтаксису:

CREATE SEQUENCE имя_последовательности
[START WITH начальное_значение] [INCREMENT BY шаг];

В самом простейшем случае генерируется последовательность целых чисел от 1 до 10^{27} с шагом единица. Конструкция INCREMENT BY позволяет указать шаг изменения значений последовательности. Конструкция START WITH позволяет задать начальное значение генерируемой последовательности, которое при ее отсутствии устанавливается равным единице. Для вставки в столбец текущего значения последовательности нужно указать имя_последовательности.CURRVAL, а для вставки в столбец измененного по правилам формирования последовательности следующего значения используется имя_последовательности.NEXTVAL. Последовательности являются самостоятельными объектами БД, одна и та же последовательность может быть использована для задания уникальных значений столбцов нескольких таблиц; при удалении или модификации последовательности значения, созданные ею, сохраняются в таблицах БД.

Вставка строк в таблицу осуществляется с помощью оператора INSERT, который имеет следующий синтаксис:

INSERT INTO имя_таблицы [(список_столбцов)]
{VALUES (значение1 [, значение2] ...) | подзапрос};

Если список столбцов не указывается, то список значений в конструкции VALUES должен содержать значения для всех столбцов табли-

цы, причем порядок их следования должен однозначно соответствовать порядку их следования в строке. Использование подзапроса позволяет перенести строки из некоторой таблицы в создаваемую таблицу.

Удаление строк из таблицы осуществляется с помощью оператора DELETE, который имеет следующий синтаксис:

DELETE [FROM] имя_таблицы [WHERE условие];

При отсутствии ключевого слова WHERE из таблицы удаляются все строки, но сама таблица остается.

Модификация строк таблицы реализуется с помощью оператора UPDATE, форма записи которого приведена ниже. При отсутствии условия модифицируются все строки таблицы для указанного списка столбцов.

```
UPDATE имя_таблицы
SET {(имя_столбца1 [, имя_столбца2] ...) = (подзапрос) |
имя_столбца1=значение1, имя_столбца2={значение2 | (подзапрос)}}
[WHERE условие];
```

Изменение структуры таблицы выполняется оператором ALTER TABLE, с помощью которого можно осуществить добавление одного или нескольких новых столбцов в таблицу, изменение характеристик у одного или нескольких столбцов, добавление ограничения столбца или таблицы или удаление ограничений столбца или таблицы. Примеры его использования приведены ниже.

Удаление таблицы можно выполнить с помощью следующего оператора:

DROP TABLE имя_таблицы [CASCADE CONSTRAINTS];

При наличии конструкции CASCADE CONSTRAINTS вместе с удалением таблицы уничтожаются ограничения внешнего ключа в других таблицах.

П р и м е р ы

Постоянно работающий магазин напрямую контактирует с издательствами и имеет постоянный штат продавцов. Директор магазина должен иметь сведения:

1) *о поступивших книгах*: название книги, фамилия автора, цена, издательство, жанр;

2) *о распределении книг среди продавцов*: название книги, фамилия продавца, количество экземпляров, дата поступления.

1. Необходимо создать таблицы: BOOKS, BOOKS_DELIVERY, указав все необходимые ограничения целостности данных.

Перечень, названия и тип данных столбцов таблицы BOOKS:

Код книги	CODE_BOOK	Number(5)
Название книги	TITLE	Varchar2(25)
ФИО автора	AUTHOR	Varchar2(20)
Цена книги	PRICE	Number(7)
Издательство	PUBLISH_HOUSE	Varchar2(15)
Жанр	GENRE	Varchar2(15)

Информация таблицы BOOKS:

Код кн.	Назв. книги	ФИО автора	Цена	Изд-во	Жанр
1	Гибель Богов	Перумов Н.	345	Аст	Фантастика
2	Казаки	Толстой Л.	5568	Нова	Роман
3	Ярость	Перумов Н.	1385	Аст	Детектив
4	Дюна	Герберт Ф.	2668	Нова	Фантастика
5	Гибель Титана	Кристи А.	2345	Аст	Роман
6	Дети Дюны	Герберт Ф.	2500	Аст	Фантастика

Перечень, названия и тип данных столбцов таблицы BOOKS_DELIVERY:

Код операции	CODE_OPERATION	Number(10)
Код книги	CODE_BOOK	Number(5)
ФИО продавца	SALESMAN	Varchar2(20)
Количество единиц	QUANTITY	Number(4)
Дата поставки	DATE_DELIVERY	Date

Информация таблицы BOOKS_DELIVERY:

Код опер.	Код книги	ФИО Продавца	Кол-во ед.	Дата поставки
1	1	Иванов И. И.	5	20-01-2006
2	3	Иванов И. И.	5	10-02-2006
3	2	Петров П. П.	4	25-01-2006
4	4	Петров П. П.	4	20-02-2006

5 4 Иванов И. И. 4 20-02-2006

Для создания таблицы BOOKS воспользуемся оператором CREATE TABLE. Столбец CODE_BOOK, содержащий уникальный код книги, является ключевым, и по отношению к его значениям устанавливаем ограничение PRIMARY KEY. За значениями для этого столбца пользователь должен следить сам. Ограничение именуется как PK_BOOKS. Поскольку столбец TITLE не может иметь пустые значения, на него накладываем ограничение NOT NULL. На значения столбца PRICE накладываем ограничение, связанное со стоимостью книги, она должна быть не менее 100 руб. Ограничение получает имя PRICE_BOOKS. Если таблица BOOKS была уже создана ранее, то перед повторным созданием ее следует удалить следующим оператором:

```
DROP TABLE BOOKS;
```

Следующий оператор CREATE языка SQL создает таблицу BOOKS с необходимыми ограничениями целостности данных:

```
CREATE TABLE BOOKS (CODE_BOOK NUMBER(5)
CONSTRAINT PK_BOOKS PRIMARY KEY,
TITLE VARCHAR2(25) CONSTRAINT TITLE_BOOKS NOT NULL,
AUTHOR VARCHAR2(20),
PRICE NUMBER(7) CONSTRAINT PRICE_BOOKS
CHECK(PRICE >100),
PUBLISH_HOUSE VARCHAR2(15), GENRE VARCHAR2(15));
```

Вставка строк в таблицу BOOKS осуществляется следующей совокупностью операторов:

```
INSERT INTO BOOKS VALUES(1,'Гибель Богов','Перумов Н.', 345,
'Аст', 'Фантастика');
INSERT INTO BOOKS VALUES(2,'Казаки','Толстой Л.', 5568, 'Нова',
'Роман');
INSERT INTO BOOKS VALUES(3,'Ярость','Перумов Н.',1385,'Аст',
'Детектив');
INSERT INTO BOOKS VALUES(4,'Дюна', 'Герберт Ф.', 2668, 'Нова',
'Фантастика');
INSERT INTO BOOKS VALUES(5,'Гибель Титана', 'Кристи А.', 2345,
'Аст', 'Роман');
INSERT INTO BOOKS VALUES(6,'Дети Дюны', 'Герберт Ф.', 2500,
'Аст', 'Фантастика');
```

Просмотр введенных значений можно выполнить с помощью следующего оператора SQL:

```
SELECT * FROM BOOKS;
```

Для создания таблицы BOOKS_DELIVERY также воспользуемся оператором CREATE TABLE. Столбец CODE_OPERATION, содержащий уникальный код операции, является ключевым, и по отношению к его значениям устанавливаем ограничение PRIMARY KEY. Это ограничение получает имя DELIVERY_PR. Для генерации уникальных значений этого столбца используем предварительно созданную с помощью оператора CREATE SEQUENCE последовательность CODE_OP:

```
CREATE SEQUENCE CODE_OP;
```

Поскольку столбец CODE_BOOK должен содержать только те значения, которые присутствуют в соответствующем столбце таблицы BOOKS, необходимо задать соответствующее ограничение на значения столбца CODE_BOOK. Это ограничение можно сделать как ограничением столбца, так и ограничением таблицы, задав ему имя DELIVERY_FK. Конструкция ON DELETE CASCADE обеспечит при удалении из таблицы BOOKS строк, содержащих значения внешнего ключа, автоматическое удаление строк и из таблицы BOOKS_DELIVERY. Для столбца DATE_DELIVERY значением по умолчанию устанавливаем текущую дату (SYSDATE). Если таблица BOOKS_DELIVERY была уже создана ранее, то перед повторным созданием ее следует удалить следующим оператором:

```
DROP TABLE BOOKS_DELIVERY;
```

Следующий оператор создает таблицу BOOKS_DELIVERY с необходимыми ограничениями целостности данных:

```
CREATE TABLE BOOKS_DELIVERY (
CODE_OPERATION NUMBER(10)
CONSTRAINT DELIVERY_PR PRIMARY KEY,
CODE_BOOK NUMBER(5), SALESMAN VARCHAR2(20),
QUANTITY NUMBER(4),
DATE_DELIVERY DATE DEFAULT SYSDATE,
CONSTRAINT DELIVERY_FK FOREIGN KEY(CODE_BOOK)
REFERENCES BOOKS(CODE_BOOK) ON DELETE CASCADE);
```

Вставка строк в таблицу BOOKS_DELIVERY осуществляется использованием следующей совокупности операторов INSERT:

```
INSERT INTO BOOKS_DELIVERY VALUES(CODE_OP.NEXTVAL,
1, 'Иванов И. И.', 5, '20-01-2006');
INSERT INTO BOOKS_DELIVERY VALUES(CODE_OP.NEXTVAL,
3, 'Иванов И. И.', 5, '10-02-2006');
INSERT INTO BOOKS_DELIVERY VALUES(CODE_OP.NEXTVAL,
2, 'Петров П. П.', 4, '25-01-06');
INSERT INTO BOOKS_DELIVERY VALUES(CODE_OP.NEXTVAL,
4, 'Петров П. П.', 4, '20-02-06');
INSERT INTO BOOKS_DELIVERY VALUES(CODE_OP.NEXTVAL,
4, 'Иванов И. И.', 4, '20-02-06');
```

Просмотр введенных значений можно выполнить с помощью следующего оператора SQL:

```
SELECT * FROM  BOOKS_DELIVERY;
```

2. Создать таблицу BOOKS1, структура которой идентична структуре таблицы BOOKS, и перенести в нее строки с информацией о книгах жанра «Фантастика» и «Роман» из таблицы BOOKS.

Следующий набор операторов SQL создает таблицу BOOKS1 и переносит в нее строки из таблицы BOOKS:

```
CREATE  TABLE  BOOKS1 (CODE_BOOK  NUMBER(5)
CONSTRAINT  PK_BOOKS1  PRIMARY  KEY,
TITLE  VARCHAR2(25)  CONSTRAINT  TITLE_BOOKS1
NOT  NULL, AUTHOR  VARCHAR2(15),
PRICE  NUMBER(7)  CONSTRAINT PRICE_BOOKS1
CHECK(PRICE >100), PUBLISH_HOUSE  VARCHAR2(15),
GENRE  VARCHAR2(15));

INSERT  INTO  BOOKS1  SELECT  CODE_BOOK, TITLE, AUTHOR,
PRICE, PUBLISH_HOUSE, GENRE  FROM  BOOKS  WHERE
GENRE = 'Фантастика'  OR  GENRE = 'Роман';
```

Аналогичные действия выполняются следующим оператором:

```
CREATE  TABLE  BOOKS1 AS
SELECT  CODE_BOOK, TITLE, AUTHOR, PRICE,
PUBLISH_HOUSE, GENRE  FROM  BOOKS  WHERE
GENRE = 'Фантастика'  OR  GENRE = 'Роман';
```

Просмотр введенных значений можно выполнить с помощью следующего оператора SQL:

```
SELECT * FROM  BOOKS1;
```

3. Выполнить с помощью оператора UPDATE следующие изменения в таблице BOOKS:

а) заменить в таблице BOOKS в строке с фамилией автора «Кристи А.» в столбце AUTHOR фамилию «Кристи А.» на «Агата Кристи» и в столбце GENRE жанр «Роман» на жанр «Детектив»;

б) заменить в таблице BOOKS цену книги «Гибель Титана» на цену книги «Казаки».

Следующий набор операторов выполнит указанные действия:

```
UPDATE BOOKS SET GENRE='Детектив', AUTHOR='Агата Кристи'
WHERE AUTHOR='Кристи А.';

UPDATE BOOKS SET PRICE=
(SELECT PRICE FROM BOOKS WHERE TITLE='Казаки')
WHERE TITLE='Гибель Титана';
```

4. Удалить из таблицы BOOKS все строки, содержащие информацию о книгах в жанре «Роман». Используем следующий оператор:

```
DELETE BOOKS WHERE GENRE = 'Роман';
```

5. Добавить в таблицу BOOKS новый столбец PRICE_У_E с типом данных NUMBER(7,2) и пересчитать цену книг в условных единицах. Следующие операторы выполнят сначала добавление нового столбца в таблицу, а затем заполнят его соответствующими значениями:

```
ALTER TABLE BOOKS ADD PRICE_У_E NUMBER(7,2);
UPDATE BOOKS SET PRICE_У_E=PRICE/2155;
```

Отметим, что при добавлении нескольких новых столбцов в таблицу они заключаются в скобки:

```
ALTER TABLE BOOKS ADD (YEAR_PUBLISH CHAR(4),
TOWN_PUBLISH CHAR(15));
```

Для изменения характеристик столбца указывается ключевое слово MODIFY, имя столбца и новые характеристики:

```
ALTER TABLE BOOKS MODIFY (YEAR_PUBLISH NUMBER(4),
TOWN_PUBLISH CHAR(25));
```

Для добавления ограничения целостности данных необходимо указать ключевое слово CONSTRAINT, имя ограничения и само ограничение. При этом необходимо помнить, что добавляемое ограничение целостности данных не должно противоречить данным, находящимся в таблице.

ALTER TABLE BOOKS ADD CONSTRAINT ZZ CHECK(PRICE>200);

Для временного отключения проверки ограничения целостности с именем ZZ необходимо использовать команду

ALTER TABLE BOOKS DISABLE CONSTRAINT ZZ;

Для включения проверки этого ограничения целостности можно использовать команду

ALTER TABLE BOOKS ENABLE CONSTRAINT ZZ;

Для удаления ограничения целостности с именем ZZ необходимо указать ключевые слова DROP и CONSTRAINT и имя удаляемого ограничения

ALTER TABLE BOOKS DROP CONSTRAINT ZZ;

12.7. ВЫБОР ИНФОРМАЦИИ ИЗ БАЗЫ ДАННЫХ

Выбор информации из одной или нескольких таблиц БД осуществляется при помощи оператора SELECT, упрощенный синтаксис которого имеет следующий вид:

```
SELECT [DISTINCT] {* | {[имя_таблицы.] имя_столбца | выражение }
[псевдоним] [,{[имя_таблицы.] имя_столбца | выражение} [псевдоним]]...}
FROM {имя_таблицы | имя_представления | подзапрос} [псевдоним]
[,{имя_таблицы | имя_представления | подзапрос} [псевдоним]…]
[WHERE   условие]
[GROUP BY выражение1 [, выражение2... ]  [HAVING условие] ]
[{UNION | UNION ALL | INTERSECT | MINUS} SELECT оператор ]
[ORDER BY выражение1 [ASC | DESC]
[, выражение2 [ASC | DESC]] ...];
```

Построение списка выбора. В операторе SELECT при формировании списка выбора, состоящего из имен столбцов и выражений, для построения выражений могут использоваться имена столбцов таблиц и представлений, литералы, функции, соединяемые знаками арифметических действий. Если необходимо указать имена столбцов, которые имеют одинаковые идентификаторы в двух разных таблицах, то к этим именам через точку необходимо приписать имя таблицы. При необходимости сложному выражению можно присвоить псевдоним, который будет использован в дальнейшем тексте оператора. Все имена таблиц, имена столбцов которых использовались для построения выражений в списке

выбора, обязательно должны быть перечислены в тексте оператора после ключевого слова FROM.

Упорядочение строк. Строки, возвращаемые SELECT-запросом, могут быть упорядочены по возрастанию или убыванию значений определенных выражений. В качестве выражения может использоваться имя столбца. Для реализации этой процедуры используется конструкция ORDER BY, в которой указывается перечень, состоящий из одного или нескольких выражений, разделенных запятыми, по значениям которых и осуществляется упорядочение. По умолчанию извлекаемые строки упорядочиваются по возрастанию значений указанных в перечне выражений. Для того чтобы задать другой вариант упорядочения строк, после каждого выражения или группы выражений в перечне указывается либо ASC (по возрастанию), либо DESC (по убыванию значений), а перечисление этих групп осуществляется через запятую.

Условие выбора строк. В конструкции WHERE при построении условия, которому должны удовлетворять выбираемые строки, можно использовать весь имеющийся в языке SQL набор операций сравнения и логических операций. Для того чтобы задать конкретное значение в условии, можно воспользоваться переменной подстановки, которая позволяет ввести необходимое значение в момент выполнения запроса. Переменная подстановки определяется наличием символа & в начале ее имени.

Подзапросы. Подзапросы, или вложенные запросы, применяются для возврата группы строк или множества значений, которые будут использованы родительским запросом. В зависимости от формы построения подзапрос может выполняться либо один раз для родительского запроса, либо один раз для каждой строки, извлеченной родительским запросом. В последнем случае такой подзапрос называется коррелированным подзапросом. Характерным признаком коррелированного подзапроса является наличие в его фразе WHERE ссылок на столбцы родительского запроса.

Объединение строк. В многокомпонентном запросе можно определенным образом объединить в единое целое группы строк, извлекаемые отдельно выполняемыми запросами. Чтобы задать порядок объединения, используется одно из ключевых слов конструкции UNION | UNION ALL | INTERSECT | MINUS.

П р и м е р ы

1. Выбрать из таблицы BOOKS всю информацию о книгах:

```
SELECT * FROM BOOKS;
```

2. Выбрать из таблицы BOOKS всю информацию о первых трех книгах:

```
SELECT * FROM BOOKS WHERE ROWNUM < 4;
```

3. Выбрать из таблицы BOOKS информацию о книгах с указанием фамилии автора, названия и цены и упорядочить ее по возрастанию значений столбца AUTHOR и убыванию значений столбца PRICE; фамилии авторов и названия вывести заглавными буквами:

```
SELECT UPPER(AUTHOR), UPPER(TITLE), PRICE FROM BOOKS
ORDER BY AUTHOR ASC, PRICE DESC;
```

4. Выбрать из таблицы BOOKS информацию (фамилия автора, название) о книгах в жанре «Роман»;

```
SELECT AUTHOR, TITLE FROM BOOKS WHERE GENRE =
'Роман';
```

5. Выбрать из таблицы BOOKS информацию (фамилия автора, название, жанр) о книгах в жанре не «Роман»;

а)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS WHERE
GENRE != 'Роман';
```

б)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS WHERE
GENRE <> 'Роман';
```

в)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS
MINUS
SELECT AUTHOR, TITLE, GENRE FROM BOOKS
WHERE GENRE = 'Роман';
```

6. Выбрать из таблицы BOOKS информацию (фамилия автора, название, цена) о книгах стоимостью больше 260 и меньше 1000;

```
SELECT AUTHOR, TITLE, PRICE FROM BOOKS WHERE PRICE
BETWEEN 260 AND 1000;
```

7. Выбрать из таблицы BOOKS информацию (фамилия автора, название, жанр) о книгах в жанрах «Роман» и «Детектив»:

а)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS
WHERE GENRE = 'Роман' OR GENRE = 'Детектив';
```

б)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS
WHERE GENRE IN ('Роман', 'Детектив');
```

в)
```
SELECT AUTHOR, TITLE, GENRE FROM BOOKS
WHERE GENRE = 'Роман'
```

```
UNION
SELECT  AUTHOR, TITLE, GENRE  FROM  BOOKS
WHERE  GENRE = 'Детектив';
```

8. Выбрать из таблицы BOOKS информацию (фамилия автора, название, жанр) о книгах жанров «Роман», «Детектив» издательства «Аст»:

а)
```
SELECT  AUTHOR, TITLE, GENRE  FROM BOOKS  WHERE
(GENRE = 'Роман'  OR  GENRE = 'Детектив')  AND
 PUBLISH_HOUSE = 'Аст';
```

б)
```
SELECT  AUTHOR, TITLE, GENRE  FROM  BOOKS  WHERE
GENRE  IN  ('Роман', 'Детектив')  AND  PUBLISH_HOUSE = 'Аст';
```

9. Выбрать из таблицы BOOKS информацию (фамилия автора, название) о книгах, название которых начинается со слова «Гибель»:

а)
```
SELECT  AUTHOR, TITLE  FROM  BOOKS
WHERE  TITLE  LIKE  'Гибель%';
```

б)
```
SELECT  AUTHOR, TITLE  FROM  BOOKS
WHERE  SUBSTR(TITLE, 1, 6) = 'Гибель';
```

10. Выбрать из таблицы BOOKS информацию (фамилия автора, название, жанр) о книгах, относящихся к указанному жанру. Необходимое значение задать, используя переменную подстановки:

```
SELECT  AUTHOR, TITLE, GENRE  FROM  BOOKS
WHERE  GENRE = '&GANR'  ORDER  BY  AUTHOR;
```

В ответ на запрос, выдаваемый системой, набрать одно из значений столбца GENRE (Роман, Фантастика, Детектив).

11. Выбрать из таблицы BOOKS информацию о книгах, имеющихся в других издательствах, того же жанра, что и в издательстве «Нова»:

а)
```
SELECT * FROM  BOOKS  WHERE  GENRE  IN
(SELECT  DISTINCT  GENRE  FROM  BOOKS  WHERE
PUBLISH_HOUSE = 'Нова')  AND  PUBLISH_HOUSE <> 'Нова';
```

б)
```
SELECT * FROM  BOOKS  WHERE  GENRE  = ANY
(SELECT  DISTINCT  GENRE  FROM  BOOKS  WHERE
PUBLISH_HOUSE = 'Нова')  AND  PUBLISH_HOUSE <> 'Нова';
```

12. Выбрать из таблицы BOOKS список фамилий авторов, чьи книги имеются в каждом из издательств:

```
SELECT AUTHOR FROM  BOOKS  WHERE  PUBLISH_HOUSE = 'Аст'
    INTERSECT
```

```
SELECT AUTHOR FROM  BOOKS WHERE PUBLISH_HOUSE = 'Нова';
```

Группирование строк. Строки, возвращаемые SELECT-запросом, могут быть объединены в группы на основе значений определенного выражения для каждой строки. Примером такого группирования может служить объединение в группы книг одного жанра, информация о которых имеется в таблице BOOKS. Так как в таблице присутствуют книги трех жанров, то будут сформированы только три группы строк. Применив к каждой группе функцию SUM для столбца, содержащего значение цены, можно получить суммарную величину стоимости книг по каждому жанру. Для осуществления группирования используется конструкция GROUP BY оператора SELECT, в которой указывается перечень, состоящий из одного или нескольких выражений, разделенных запятыми, по значениям которых и осуществляется группирование. Если оператор SELECT содержит пункт GROUP BY, то список извлекаемых значений ограничен. Он может содержать константы, групповые функции, функцию SYSDATE и выражения, идентичные указанным в пункте GROUP BY. На формирование результирующих строк могут быть наложены определенные условия. Чтобы задать такое условие, используется ключевое слово HAVING.

П р и м е р ы

1. Выбрать из таблицы BOOKS информацию о количестве различных жанров:

```
SELECT  COUNT(DISTINCT  GENRE)  FROM  BOOKS;
```

2. Выбрать из таблицы BOOKS информацию о количестве, суммарной стоимости и максимальной стоимости имеющихся книг:

```
SELECT  COUNT(CODE_BOOK), SUM(PRICE), MAX(PRICE)
FROM BOOKS;
```

3. Выбрать из таблицы BOOKS по каждому издательству информацию о количестве и суммарной стоимости изданных им книг, сгруппировав ее по жанрам:

```
SELECT  PUBLISH_HOUSE, GENRE, COUNT(CODE_BOOK),
SUM(PRICE) FROM  BOOKS  GROUP  BY  PUBLISH_HOUSE,
GENRE;
```

4. Выбрать из таблицы BOOKS информацию о количестве и средней стоимости (округлив значение средней стоимости до двух знаков после запятой) книг в тех жанрах, где количество различных названий книг не менее 2:

```
SELECT GENRE, COUNT(DISTINCT TITLE),
ROUND(AVG(PRICE), 2) FROM BOOKS GROUP BY GENRE
HAVING COUNT(DISTINCT TITLE) >= 2;
```

5. Выбрать из таблицы BOOKS информацию о минимальной стоимости книг в жанре «Роман»:

```
SELECT MIN(PRICE) FROM BOOKS WHERE GENRE = 'Роман';
```

6. Выбрать из таблицы BOOKS информацию (фамилия автора, название, цена) о самой дешевой книге в жанре «Роман»:

```
SELECT AUTHOR, TITLE, PRICE FROM BOOKS
WHERE PRICE = (SELECT MIN(PRICE) FROM BOOKS
WHERE GENRE = 'Роман') AND GENRE = 'Роман';
```

7. Выбрать из таблицы BOOKS информацию (фамилия автора, название, жанр, цена) о книгах, имеющих максимальную стоимость в своем жанре:

а)
```
SELECT AUTHOR, TITLE, GENRE, PRICE FROM BOOKS
WHERE (PRICE, GENRE) IN (SELECT MAX(PRICE), GENRE
FROM BOOKS GROUP BY GENRE);
```

б)
```
SELECT AUTHOR, TITLE, BOOKS.GENRE, PRICE FROM
BOOKS, (SELECT GENRE, MAX(PRICE) МАКС FROM BOOKS
GROUP BY GENRE) P1
WHERE PRICE = МАКС AND BOOKS.GENRE = P1.GENRE;
```

в)
```
SELECT AUTHOR, TITLE, GENRE, PRICE FROM BOOKS P1
WHERE PRICE = (SELECT MAX(PRICE) FROM BOOKS
WHERE BOOKS.GENRE = P1.GENRE);
```

Третий запрос содержит коррелированный подзапрос. Поскольку в своем условии подзапрос содержит ссылку на столбец родительского запроса, он будет выполняться один раз для каждой строки, извлекаемой родительским запросом. В первом и втором вариантах подзапрос не является коррелированным, он выполняется только один раз для родительского запроса.

8. Выбрать из таблицы BOOKS информацию (фамилия автора, название, цена) о книгах стоимостью больше средней стоимости книг:

```
SELECT AUTHOR, TITLE, PRICE FROM BOOKS
WHERE PRICE > (SELECT AVG(PRICE) FROM BOOKS);
```

9. Выбрать из таблицы BOOKS список жанров, по которым имеется наибольшее количество различных книг, с указанием количества книг:

```
SELECT GENRE, COUNT(DISTINCT TITLE) FROM BOOKS
GROUP BY GENRE HAVING COUNT(DISTINCT TITLE) =
(SELECT MAX(COUNT(DISTINCT TITLE)) FROM BOOKS
GROUP BY GENRE);
```

10. Выбрать из таблицы BOOKS информацию о книгах (фамилия автора, название), относящихся к жанрам, по которым имеется наибольшее количество различных книг:

```
SELECT AUTHOR, TITLE FROM BOOKS WHERE GENRE IN
(SELECT GENRE FROM BOOKS  GROUP BY GENRE HAVING
COUNT(DISTINCT TITLE) =
(SELECT MAX(COUNT(DISTINCT TITLE)) FROM BOOKS
GROUP BY GENRE));
```

Выбор информации из нескольких таблиц (соединение). *Соединение* – это SELECT-запрос, который выбирает строки из двух или более таблиц. При этом запрос может извлекать любые столбцы из любой таблицы. Если хотя бы две из этих таблиц имеют одинаково названные столбцы, то имена таких столбцов должны уточняться именами таблиц, записываемых перед именами столбцов через точку. Большинство SELECT-запросов с соединениями содержат условия, в которых сравниваются значения столбцов из разных таблиц. Такие условия называются условиями соединения.

Эквисоединение – это соединение с использованием в условии соединения операции равенства. Таким образом, эквисоединение извлекает строки с эквивалентными значениями в указанных столбцах.

Декартово произведение таблиц строится при отсутствии в запросе условия соединения. В этом случае к каждой строке первой таблицы приписывается каждая строка второй таблицы.

Самосоединение соединяет таблицу саму с собой. При этом таблица появляется в списке FROM дважды и должна иметь дополнительное имя (псевдоним), чтобы можно было однозначно идентифицировать столбцы в условии соединения.

Внешнее соединение выдает все строки, которые удовлетворяют условию соединения, а также строки одной из таблиц, которые не удовлетворяют условию соединения. Чтобы записать запрос, который выполняет внешнее соединение таблиц A и B и выдает все строки из таблицы A, применим операцию внешнего соединения (+) ко всем столбцам из таблицы B в условиях соединения. Тогда для всех строк из

таблицы A, для которых нет соответствующих строк в таблице B, система предоставит строку, содержащую NULL во всех выражениях в списке столбцов, которые содержат столбцы из таблицы B.

П р и м е р ы

1. Выбрать из таблиц BOOKS и BOOKS_DELIVERY информацию о книгах, поставленных продавцам за период с 24.01.2006 по 12.02.2006, указав для выводимого значения даты специальный формат вывода:

```
SELECT  SALESMAN, AUTHOR, TITLE,
TO_CHAR(DATE_DELIVERY, 'DD MONTH YYYY')
FROM  BOOKS, BOOKS_DELIVERY  WHERE
BOOKS.CODE_BOOK = BOOKS_DELIVERY.CODE_BOOK  AND
DATE_DELIVERY  BETWEEN  '24-01-06'  AND  '12-02-06'
ORDER  BY  SALESMAN, AUTHOR;
```

2. Выбрать из таблиц BOOKS и BOOKS_DELIVERY по указанному продавцу перечень издательств и жанров имеющихся у него книг без повторения; чтобы задать фамилию продавца, использовать переменную подстановки:

```
SELECT  DISTINCT  SALESMAN, PUBLISH_HOUSE, GENRE
FROM BOOKS, BOOKS_DELIVERY
WHERE  BOOKS_DELIVERY.CODE_BOOK = BOOKS.CODE_BOOK
AND  SALESMAN = '&SALESMAN'
ORDER  BY  PUBLISH_HOUSE, GENRE;
```

3. Выбрать из таблиц BOOKS и BOOKS_DELIVERY список продавцов, у которых в наличии не менее 10 книг, указав данные об общем количестве и суммарной стоимости имеющихся у них книг:

```
SELECT  SALESMAN, SUM(QUANTITY), SUM(PRICE*QUANTITY)
FROM  BOOKS, BOOKS_DELIVERY
WHERE  BOOKS_DELIVERY.CODE_BOOK = BOOKS.CODE_BOOK
GROUP  BY  SALESMAN  HAVING  SUM(QUANTITY) >= 10;
```

4. Выбрать из таблиц BOOKS и BOOKS_DELIVERY по каждому издательству информацию об общем количестве и суммарной стоимости поставленных ими книг каждому продавцу:

```
SELECT  PUBLISH_HOUSE, SALESMAN, COUNT(QUANTITY),
SUM(PRICE*QUANTITY)  FROM  BOOKS, BOOKS_DELIVERY
WHERE  BOOKS.CODE_BOOK = BOOKS_DELIVERY.CODE_BOOK
GROUP  BY  PUBLISH_HOUSE, SALESMAN;
```

5. Выбрать из таблиц BOOKS и BOOKS_DELIVERY по каждой книге, сведения о которой имеются в таблице BOOKS, информацию о количестве продавцов, которым она была поставлена:

```
SELECT TITLE, AUTHOR, COUNT(SALESMAN) FROM BOOKS,
BOOKS_DELIVERY WHERE BOOKS_DELIVERY.CODE_BOOK
(+)= BOOKS.CODE_BOOK GROUP BY TITLE, AUTHOR;
```

6. Выбрать из таблиц BOOKS и BOOKS_DELIVERY по каждому продавцу информацию об отсутствующих у них книгах, общий перечень которых находится в таблице BOOKS:

```
SELECT SALESMAN, TITLE, PRICE FROM BOOKS,
BOOKS_DELIVERY
MINUS
SELECT SALESMAN, TITLE, PRICE FROM BOOKS,
BOOKS_DELIVERY WHERE BOOKS_DELIVERY.CODE_BOOK =
BOOKS.CODE_BOOK;
```

7. Выбрать из таблицы BOOKS информацию (название, цена) о трех самых дешевых книгах; предполагается, что в перечне книг не более трех различных книг с минимальной ценой:

```
SELECT A.TITLE, A.PRICE FROM BOOKS A, BOOKS B WHERE
A.PRICE >= B.PRICE
GROUP BY A.TITLE, A.PRICE HAVING COUNT(B.TITLE) <= 3
ORDER BY A.TITLE;
```

8. Выбрать из таблиц BOOKS и BOOKS_DELIVERY информацию (название, фамилия автора) о книгах, которые не были поставлены в магазин для продажи:

```
SELECT TITLE, AUTHOR FROM BOOKS WHERE
NOT EXISTS
(SELECT * FROM BOOKS_DELIVERY WHERE
BOOKS_DELIVERY.CODE_BOOK = BOOKS.CODE_BOOK);
```

13. ОСНОВЫ ЯЗЫКА PL/SQL

13.1. АЛФАВИТ И ЛЕКСЕМЫ ЯЗЫКА. СТРУКТУРА ПРОГРАММЫ

PL/SQL – это процедурный, блочно-структурированный язык программирования, являющийся расширением языка SQL СУБД Oracle. Он

предоставляет ряд возможностей, которые позволяют создать большие, многофункциональные приложения для работы с БД.

Алфавит языка включает следующий набор символов:

1) английские буквы верхнего и нижнего регистров A..Z, a..z;

2) арабские цифры 0..9;

3) символы + – * / <>= ; : . , ' ~ ! @ # $ % ^ & _| ( ) { } [ ] ;

4) символы табуляции, пробелы и символы возврата каретки.

Лексемы (группы символов алфавита) делятся на идентификаторы, литералы, разделители и комментарии.

Идентификаторы имеют длину до тридцати символов и состоят из прописных и строчных букв, цифр и знака подчеркивания, причем первой должна быть буква. Допускается, но не рекомендуется использовать специальные символы, такие как #, $. Некоторые из идентификаторов в языке PL/SQL имеют специальное синтаксическое значение. Такие идентификаторы называются зарезервированными и не должны переопределяться.

Литералы – это явно заданное число, символ, строка или логическое значение, не представленное идентификатором. Литералы делятся на числовые, строковые и логические.

Числовые литералы бывают двух типов: целые и действительные. Целые литералы – это знаковые числа без десятичной точки (6; –14). Действительные литералы – знаковые целые или дробные числа с десятичной точкой (6.667; –12.0). Допускается запись числовых литералов в экспоненциальной форме (1.0E–7; 2E5).

Строковые литералы – это последовательность символов, заключенных в одинарные кавычки (апострофы). Все строковые литералы, за исключением пустой строки ('') имеют тип CHAR. Если в строковом литерале необходимо указать одинарную кавычку, то при записи она просто удваивается.

Логические литералы – это предопределенные значения TRUE, FALSE и NULL. NULL указывает на неизвестное значение.

Разделитель – это совокупность одного или двух символов, которая имеет определенное значение в PL/SQL. Простые разделители содержат только один символ. К ним относятся, например, знаки арифметических операций (+, –, *, /), знаки операций отношения (=, >, <), признак конца выражения (;). Составные разделители содержат два символа. К ним относятся, например, оператор присваивания (:=), оператор конкатенации (||), операция возведения в степень (**), начало и конец метки (<< >>), оператор диапазона (..), операция отношения неравно (<>, !=, ~=, ^=),

операция отношения меньше или равно (<=), операция отношения больше или равно (>=).

Комментарии содержат пояснительный текст и делятся на однострочные и многострочные. Однострочный комментарий представляет собой строку, начинающуюся с двух символов дефис (-). В многострочном комментарии текст заключается в специальные разделители /* */.

Структура программы представляет собой набор блоков PL/SQL, рекурсивно вложенных друг в друга.

Структура блока:

```
[<<метка>>]
[DECLARE
    раздел объявлений]
BEGIN
    исполняемый раздел
[EXEPTION
    раздел обработки исключений]
END[<<метка>>];
```

Обязательным должен быть только исполняемый раздел, содержащий операторы языка. Существуют следующие типы блоков: анонимные, именованные, триггеры и подпрограммы (процедуры, функции, пакеты). Анонимные блоки, в отличие от именованных, не содержат меток. Именованные и анонимные блоки называются динамическими блоками.

13.2. ТИПЫ ДАННЫХ И ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ

Объявление переменных осуществляется в разделе объявлений, при этом помимо идентификатора переменной должен быть указан и ее тип. К основным типам данных языка PL/SQL относятся скалярные и составные. Среди составных типов наибольший интерес представляет тип RECORD (записи).

Объявление скалярных типов данных. Среди скалярных типов наиболее распространены числовые, символьные, тип DATE и логический (BOOLEAN) типы данных.

Числовые типы данных представлены в основном двумя типами: BINARY_INTEGER и NUMBER. Все другие числовые типы являются подмножествами двух этих типов.

Тип переменных BINARY_INTEGER используется для хранения целых знаковых чисел в двоичном виде. Диапазон этого типа от –2147483647 до 2147483647.

Тип переменных NUMBER используется для хранения чисел с фиксированной и плавающей точкой в диапазоне от 1E–130 до 10E125. Для объявления чисел с плавающей точкой можно просто указать NUMBER. Для объявления целого числа указывается NUMBER(точность), а для объявления числа с фиксированной точкой дополнительно указывается еще и масштаб, т. е. NUMBER(точность, масштаб). Точность представляет собой общее число знаков и не превосходит 38 десятичных знаков, масштаб указывает порядок округления и задается числом от –84 до 127. При положительном значении масштаба число округляется до указанного количества цифр, стоящих справа от запятой; при отрицательном значении – до указанного количества цифр, стоящих слева от запятой. Например: число 123.456 при значении масштаба, равном 2, округляется до 123.46, а при значении, равном –2, до 100.

К ***символьным*** типам данных в основном относятся типы CHAR и VARCHAR2.

Тип CHAR(длина) используется для хранения последовательности символов фиксированной длины не более 32 767 байт. Следует иметь в виду, что в языке SQL максимальное значение аналогичного типа равно 2000 байт, поэтому не все данные этого типа можно вставлять в столбцы таблицы с типом CHAR.

Тип VARCHAR2(длина) используется для хранения символьной последовательности переменной длины. Ограничение на длину составляет 32 767 байт.

Тип DATE используется для хранения значений даты и времени. Значение даты и времени хранится во внутреннем двоичном формате и при помещении его в переменную символьного типа автоматически преобразуется в строку, используя формат даты, установленный по умолчанию. Функция SYSDATE возвращает текущее значение даты и времени.

Тип BOOLEAN используется для хранения логических значений TRUE, FALSE, NULL. Над такими переменными можно выполнять только логические операции, причем в трехзначной логике.

Скалярные переменные объявляются явным и неявным образом.

Явное объявление переменной любого из скалярных типов осуществляется по следующему правилу:

имя_переменной [CONSTANT] тип [NOT NULL]

[{:= | DEFAULT} значение];

При использовании ключевого слова CONSTANT переменной должно быть присвоено значение, которое впоследствии не может быть изменено. Если указано ключевое слово NOT NULL, то переменную необхо-

димо проинициализировать, и впоследствии она не может принимать значение NULL. Переменная инициализируется значением своего типа либо с помощью оператора присваивания, либо с помощью ключевого слова DEFAULT. Каждая переменная объявляется отдельно.

Неявное объявление переменной скалярного типа осуществляется с помощью специального атрибута %TYPE, который позволяет объявить переменную, тип которой соответствует либо типу другой переменной, либо типу столбца таблицы БД.

П р и м е р ы

```
A1      NUMBER;
A11     NUMBER:=15;
A12     NUMBER NOT NULL DEFAULT 15;
A2      A1%TYPE;
A3      BOOKS.PRICE%TYPE;
```

Объявляя переменные, следует иметь в виду, что:

1) имена локальных переменных и формальных параметров имеют приоритет перед именами таблиц БД;

2) имена столбцов таблицы БД имеют приоритет перед именами локальных переменных и формальных параметров.

Объявление переменных типа RECORD (записи). Составной тип ***записи*** определяет структуру, содержащую некоторое количество переменных. Переменные могут быть любого типа, в том числе и ранее определенными записями. Ссылка на отдельные элементы этой структуры осуществляется с помощью точечной нотации.

Тип записи должен быть определен до того, как будут объявлены переменные этого типа. Для явного определения нового составного типа данных используется следующий общий синтаксис:

```
TYPE тип_записи IS RECORD
(поле1 тип1 [NOT NULL] [{:= | DEFAULT } значение1],
 поле2 тип2 [NOT NULL] [{:= | DEFAULT } значение2], …);
```

Для явного объявления переменной-записи этого типа необходимо указать имя переменной и имя типа. Допускается неявное определение переменных типа записи, выполняемое с помощью атрибута %ROWTYPE, что позволяет определять переменные-записи, структура которых идентична структуре записи указанной таблицы или структуре ранее определенной переменной-записи.

Рассмотрим переменную-запись, объявляемую явно:

```
DECLARE
```

```
TYPE  BOOK  IS  RECORD   --вводится тип записи – BOOK
(
AUTHOR   VARCHAR2(15),   --фамилия автора
TITLE       VARCHAR2(25),    --название книги
PRICE       NUMBER(6)          --цена
);
BOOK_FAN     BOOK;          --явное объявление
BOOK_ROM     BOOK;          --явное объявление
BOOK_POEM   BOOK_FAN%ROWTYPE;  --неявное объявление
```

Присваивание значений элементам, входящим в запись, необходимо выполнять поэлементно, используя точечную нотацию:

```
BOOK_FAN.PRICE :=16000;
```

Для присвоения значений сразу всем полям записи или нескольким из них можно воспользоваться однострочным оператором SELECT либо оператором выборки очередной строки из открытого курсора FETCH.

Возможно присвоение значений одной переменной-записи другой при условии, что они одного типа:

```
BOOK_FAN := BOOK_ROM;
```

Однако следует иметь в виду, что переменная-запись, тип которой определен явно, и переменная-запись, тип которой определяется с помощью атрибута %ROWTYPE, всегда несовместимы.

Не допускается сравнение переменных-записей.

Ошибочной будет попытка передать в качестве значений в команде INSERT запись целиком. Значения записи должны передаваться поэлементно:

```
INSERT   INTO  TEMP VALUES (BOOK_FAN.AUTHOR,
BOOK_FAN.TITLE, BOOK_FAN.PRICE);
```

13.3. ОПЕРАТОРЫ

Операторы языка PL/SQL представлены оператором присваивания, условным оператором IF и оператором цикла LOOP.

Оператор присваивания позволяет задать переменной некоторое значение и имеет следующий синтаксис:

```
Переменная := выражение;
```

Условный оператор IF, позволяющий проверить некоторый набор условий и выполнить соответствующие действия, имеет следующий синтаксис:

```
IF  логическое выражение1  THEN
     последовательность операторов1
ELSIF  логическое выражение2  THEN
     последовательность операторов2
ELSE
    последовательность операторов3
ENF IF;
```

Оператор цикла LOOP позволяет повторить выполнение заданной последовательности операторов необходимое количество раз. Существуют три формы записи оператора цикла LOOP.

В первом варианте условие завершения цикла находится внутри тела цикла и формируется с помощью ключевых слов EXIT и WHEN.

```
LOOP
   последовательность операторов
   [EXIT  [WHEN  условие]];
END LOOP;
```

Во втором варианте повторение операторов осуществляется до тех пор, пока остается истинным условие, указанное в начальной строке оператора LOOP. Дополнительно может быть сформировано еще одно условие выхода из цикла при помощи ключевого слова EXIT.

```
WHILE  условие  LOOP
    последовательность операторов
   [EXIT  [WHEN  условие]];
END LOOP;
```

В третьем варианте переменная цикла пробегает указанный диапазон значений от нижней границы до верхней, увеличивая каждый раз свое значение на единицу, после чего осуществляется выход из цикла.

```
FOR   переменная_цикла
IN [REVERSE]   нижняя_граница..верхняя_граница
LOOP
    последовательность операторов
END LOOP;
```

Переменная цикла определяется системой неявно как переменная типа BINARY_INTEGER и не требует объявления. Значениями границ мо-

гут быть переменные, константы, выражения. Вариант REVERSE означает, что значения просматриваются в обратном порядке от верхней границы к нижней.

13.4. КУРСОРЫ

Для выполнения команды SELECT система выделяет определенную область, куда помещает помимо служебной информации и сами выбранные строки (активный набор строк). В PL/SQL имеется специальная конструкция, которая позволяет задать имя этой области и осуществить доступ к хранящейся там информации. Такая конструкция называется ***курсором***. Существуют курсоры двух типов: явные и неявные.

Явный курсор должен быть явно объявлен пользователем. Для его объявления используется следующий синтаксис:

CURSOR имя_курсора [(список_параметров)] IS SELECT…;

Оператор SELECT определяет набор извлекаемых строк. Список параметров может отсутствовать. Если же параметры используются, то они описываются следующим образом:

Имя_параметра тип [{:=| DEFAULT} значение];

Если указано значение параметра, то при открытии курсора этот параметр можно не указывать.

Управление явным курсором осуществляется в двух вариантах: явно и неявно.

При *явной форме управления курсором* необходимо явно открыть курсор, явно выбрать строки из активного набора и явно закрыть курсор.

Для того чтобы явно открыть курсор, используется следующая конструкция:

OPEN имя_курсора [(список_параметров)];

В момент открытия курсора система выполняет указанный оператор SELECT с учетом передаваемых значений параметров, т. е. выбирает соответствующий набор строк, и указатель текущей записи устанавливается в этом наборе на первую строку. Однако строки программе не передаются. Чтобы получить строки одну за другой, используется оператор FETCH. Выборка строк допускается только в прямом направлении; продвижение по набору строк в обратном направлении невозможно. Синтаксис оператора FETCH:

FETCH имя_курсора INTO {имя_записи | список_столбцов};

При выполнении оператора FETCH выбирается очередная строка, а указатель текущей записи передвигается на следующую строку в наборе строк. Если были выбраны все строки, то при попытке нового считывания ошибка не возникает, но и строка не выбирается.

Для того чтобы закрыть курсор, необходимо выполнить следующий оператор:

CLOSE имя_курсора;

После закрытия курсора все попытки считывания информации приведут к ошибке. Для организации явного управления можно использовать *курсорные атрибуты*, которые представляют собой функции, возвращающие определенное значение в зависимости от выполненных действий. К ним относятся:

1) %ISOPEN – возвращающий значение TRUE, если курсор открыт, и FALSE, если курсор закрыт;
2) %FOUND – возвращающий значение TRUE, если строка найдена, и FALSE, если строка не найдена;
3) %NOTFOUND – возвращающий значение TRUE, если строка не найдена, и FALSE, если строка найдена;
4) %ROWCOUNT – возвращающий числовое значение, показывающее количество выбранных строк в курсоре.

При *неявной форме управления курсором* используется специальная форма записи оператора цикл FOR – циклы FOR с курсором. Открытие, выборка и закрытие курсора в этом случае происходит автоматически. Возвращаемая переменная-строка определяется неявно и на нее нельзя ссылаться извне области видимости цикла. При неявной форме управления курсор может также иметь параметры.

Неявные курсоры создаются и открываются системой при выполнении операторов INSERT, UPDATE и DELETE, а также при выполнении однострочного оператора SELECT. Неявный курсор называется SQL-курсором. Он имеет свои атрибуты, аналогичные атрибутам явного курсора:

SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT

Однако в однострочном операторе SELECT нельзя использовать атрибут SQL%NOTFOUND, потому что если при его выполнении информация из таблицы не будет выбрана, то сгенерируется исключение NO_DATA_FOUND. Атрибут SQL%NOTFOUND обычно используется в операторах UPDATE и DELETE для проверки, успешно или нет выполнены соответствующие операторы.

Конструкция курсора позволяет помимо оперирования наборами данных таблиц выполнять и ***модификацию информации таблиц***. Для этого используются две дополнительных конструкции:

FOR UPDATE [OF столбец1, столбец2…] [NOWAIT]

указываемая в операторе SELECT, который определяет строки формируемого активного набора, и

WHERE CURRENT OF имя_курсора

которая указывается в операторах UPDATE или DELETE, выполняемых в процессе выборки и обработки строк из активного набора.

В обычной ситуации строки, выбираемые в активный набор, не блокируются, и любой другой пользователь может осуществить их модификацию в БД. Если после этого закрыть и вновь открыть курсор, то получаешь доступ к новым, модифицированным строкам.

При использовании варианта FOR UPDATE с перечнем столбцов блокируется доступ других пользователей к указанным столбцам, но и обновлять информацию можно только в этих столбцах. Если же используется FOR UPDATE без перечня столбцов, то блокируется вся таблица, а значит, обновлять можно информацию, хранящуюся в любых столбцах таблицы.

13.5. ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

В PL/SQL предусмотрены механизмы перехвата и обработки ошибок, возникающих при выполнении программы. При обнаружении ошибки генерируется исключительная ситуация, обработка которой производится в разделе EXCEPTION. Существуют два класса исключительных ситуаций: стандартные и определяемые пользователем.

Стандартные исключительные ситуации делятся на два типа: имеющие и не имеющие предопределенное имя. Имеющие предопределенное имя исключительные ситуации помимо кода имеют еще и стандартное имя, которое используется для идентификации исключения. Ниже приведены примеры некоторых стандартных исключительных ситуаций, имеющих предопределенные имена.

1) ZERO_DIVIDE – попытка деления на нуль;
2) NO_DATA_FOUND – предложение SELECT...INTO не возвращает ни одной строки;
3) TOO_MANY_ROWS – предложение SELECT...INTO возвращает более одной строки;

4) INVALID_CURSOR – попытка выполнения запрещенной операции с курсором (например, закрытие неоткрытого курсора);

5) CURSOR_ALREADY_OPEN – попытка открытия уже открытого курсора;

6) VALUE_ERROR – арифметическая ошибка, ошибка преобразования, усечения или ограничения;

7) INVALID_NUMBER – отказ в преобразовании строки символов в число.

Пользовательские исключительные ситуации описываются в разделе DECLARE, устанавливаются в выполняемом разделе, а обрабатываются в разделе EXCEPTION. Описание пользовательской исключительной ситуации выполняется заданием имени исключения и фразы EXCEPTION. Чтобы сгенерировать исключительную ситуацию и передать управление обработчику пользовательской исключительной ситуации в случае обнаружения ошибки, используется оператор

RAISE имя_пользовательского_исключения

Для перехвата исключительной ситуации любого типа в раздел EXCEPTION должна быть включена фраза

WHEN имя_исключения THEN текст_обработчика_исключения;

Тогда при возникновении соответствующей ошибки, вместо прекращения исполнения программы и выдачи типового сообщения об ошибке, будет выполняться созданный пользователем вариант обработки исключения. Если необходимо, чтобы две или более исключительные ситуации обрабатывались одинаково, то они должны быть записаны в одном операторе WHEN, разделенные ключевым словом OR. Для перехвата всех неописанных исключительных ситуаций используется специальный обработчик OTHERS, который записывается последним в разделе EXCEPTION.

Генерация исключительной ситуации с выдачей соответствующего сообщения в рабочую среду в случае обнаружения ошибки может быть выполнена с помощью следующего оператора:

RAISE_APPLICATION_ERROR (errnum, errtext);

где errnum – код ошибки, выбираемый в диапазоне –20000 .. –20999;
errtext – поясняющая символьная строка длиной до 512 байт.

При возникновении исключительной ситуации и отсутствии соответствующего обработчика в данном блоке система пытается найти такой обработчик в блоках, охватывающих этот блок. При отсутствии обработчика система вернет ошибку «необработанное исключение».

П р и м е р ы

Создать программу, которая осуществляет в таблице BOOKS повышение цен на книги жанра «Фантастика». При этом при стоимости книги менее 2000 руб., цена увеличивается на 20 %, а при стоимости больше или равной 2000 руб. – на 10 %.

Данная задача реализуется с помощью явно объявленного пользователем курсора. При этом в первых двух вариантах показываются возможности использования обычного курсора и курсора с параметром. Приведенное решение демонстрирует два варианта обработки явно объявленного курсора: явную и неявную формы. В явной форме обработки по завершении просмотра строк активного набора осуществляется выход из цикла обработки. При этом используется курсорный атрибут %NOTFOUND. В неявной форме обработки курсора используется конструкция цикл FOR с курсором. Модифицированное значение цены записывается обратно в таблицу BOOKS с использованием конструкции WHERE CURRENT OF, при этом системе с помощью конструкции FOR UPDATE OF PRICE указывается, что будет осуществляться обновление значений столбца PRICE таблицы BOOKS. В программе неявным способом объявлены переменная типа запись ZAP и скалярная переменная NEW_ PRICE.

а) Использование обычного курсора:

```
DECLARE
   CURSOR  KUR IS  --явное объявление курсора KUR
      SELECT  CODE_BOOK,  PRICE  FROM  BOOKS
      WHERE  GENRE = 'Фантастика'  FOR  UPDATE  OF  PRICE;
   ZAP  KUR%ROWTYPE; --объявление переменной-записи
   NEW_PRICE  BOOKS.PRICE%TYPE; --объявление переменной
BEGIN
   OPEN  KUR;   --явное открытие курсора
   LOOP
      FETCH  KUR  INTO  ZAP; --выборка текущей записи
      EXIT  WHEN  KUR%NOTFOUND; --выход из цикла
      IF  ZAP.PRICE < 2000  THEN  --изменение цены
         NEW_PRICE := ZAP.PRICE*1.2;
      ELSE
         NEW_PRICE := ZAP.PRICE*1.1;
      END  IF;
      UPDATE  BOOKS
          SET  PRICE = NEW_PRICE  --обновление цены
```

```
            WHERE CURRENT  OF  KUR;
    END  LOOP;
       CLOSE  KUR;  --явное закрытие курсора
       COMMIT;   --завершение транзакции
    END;
```

При запуске программы с помощью SQL*PLUS, необходимо указывать в строке, следующей за последним оператором, косую черту (/), чтобы программа выполнилась.

б) Использование курсора с параметром:

```
    DECLARE
       CURSOR  KUR  (GANR  BOOKS.GENRE%TYPE) IS  --курсор
имеет параметр
          SELECT  CODE_BOOK, PRICE  FROM  BOOKS
          WHERE  GENRE=GANR  FOR  UPDATE  OF  PRICE;
       ZAP  KUR%ROWTYPE;
       NEW_PRICE  BOOKS.PRICE%TYPE;
    BEGIN
       OPEN  KUR ('Фантастика'); -- значение параметра
          LOOP
             FETCH  KUR  INTO  ZAP;
             EXIT  WHEN  KUR%NOTFOUND;
             IF  ZAP.PRICE < 2000  THEN
                NEW_PRICE := ZAP.PRICE*1.2;
             ELSE
                NEW_PRICE := ZAP.PRICE*1.1;
             END  IF;
             UPDATE  BOOKS  SET  PRICE = NEW_PRICE
                WHERE  CURRENT  OF  KUR;
          END  LOOP;
       CLOSE  KUR;
       COMMIT;
    END;
```

в) Использование цикла FOR с курсором:

```
    DECLARE
        NEW_PRICE  BOOKS.PRICE%TYPE;
        CURSOR  KUR  IS
          SELECT  CODE_BOOK, PRICE  FROM  BOOKS
          WHERE  GENRE = 'Фантастика'  FOR  UPDATE  OF  PRICE;
    BEGIN
```

```
    FOR ZAP IN KUR LOOP  --неявная обработка курсора
      IF ZAP.PRICE < 2000 THEN --переменная ZAP неявно объявляется системой
        NEW_PRICE := ZAP.PRICE*1.2;
      ELSE
        NEW_PRICE := ZAP.PRICE*1.1;
      END IF;
      UPDATE BOOKS SET PRICE = NEW_PRICE
        WHERE CURRENT OF KUR;
    END LOOP;
    COMMIT;
  END;
```

13.6. ТРИГГЕРЫ БАЗЫ ДАННЫХ

Триггер базы данных – это оформленный специальным образом именованный блок PL/SQL, хранящийся в БД. Каждый триггер связан с определенной таблицей и автоматически запускается при выполнении одного из DML-операторов (INSERT, DELETE, UPDATE) или их совокупности над этой таблицей.

Назначение триггеров. Триггеры могут быть использованы:

1) для реализации сложных ограничений целостности данных, которые не могут быть осуществлены стандартным образом при создании таблицы;

2) предотвращения неверных транзакций;

3) выполнения процедур комплексной проверки прав доступа и секретности данных;

4) генерации некоторых выражений на основе значений, имеющихся в столбцах таблиц;

5) при реализации сложных бизнес-правил для обработки данных (возможность отследить «эхо», т. е. при изменении одной таблицы обновлять данные связанных с ней таблиц).

Создание и включение триггеров. Для создания и автоматического включения триггера применяется следующий общий синтаксис:

```
CREATE [OR REPLACE] TRIGGER имя_триггера
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE [OF список_столбцов]}
ON имя_таблицы [FOR EACH ROW] [WHEN условие]
< PL/SQL_блок >
```

При наличии ключевых слов OR REPLACE триггер создается заново, если он уже существует.

Конструкция BEFORE | AFTER указывает на момент запуска триггера. Вариант BEFORE означает, что триггер будет запускаться перед выполнением активизирующего DML-оператора; вариант AFTER означает, что триггер будет запускаться после выполнения активизирующего DML-оператора.

Конструкция INSERT | DELETE | UPDATE [OF список_столбцов] указывает тип активизирующего триггер DML-оператора. Разрешается, используя логическую операцию OR, задать совокупность активизирующих операторов, например INSERT OR DELETE. Если при использовании варианта UPDATE указан список столбцов, то триггер будет запускаться при модификации одного из указанных столбцов; если список столбцов отсутствует, то триггер будет запускаться при изменении любого из столбцов связанной с триггером таблицы.

Конструкция FOR EACH ROW указывает на характер воздействия триггера: строковый или операторный. Если конструкция FOR EACH ROW присутствует, то триггер является строковым; при отсутствии ее триггер является операторным. Операторный триггер запускается один раз до или после выполнения активизирующего триггер DML-оператора независимо от того, сколько строк в связанной с триггером таблице подвергается модификации. Строковый триггер запускается один раз для каждой из строк, которая подвергается модификации DML-оператором, активизирующим триггер.

С помощью ключевого слова WHEN можно задать дополнительное ограничение на строки связанной с триггером таблицы, при модификации которых может быть запущен триггер.

Конструкция PL/SQL_блок представляет блок PL/SQL, который запускается при активизации триггера.

Классификация триггеров. В основном различают двенадцать типов триггеров. Тип триггера определяется сочетанием следующих трех параметров:

1) характером воздействия триггера на строки связанной с ним таблицы (строковый или операторный);

2) моментом запуска триггера: до (BEFORE) или после (AFTER) исполнения активизирующего триггер DML-оператора;

3) типом активизирующего триггер DML-оператора (INSERT, DELETE, UPDATE).

Порядок активизации триггеров. Если у таблицы имеется несколько типов триггеров, то они активизируются по следующей схеме:

1) выполняется операторный триггер BEFORE (если их несколько, то ничего о порядке их выполнения сказать нельзя);

2) выполняется строковый триггер BEFORE;

3) выполняется активизирующий триггер DML-оператор с последующей проверкой всех ограничений целостности данных;

4) выполняется строковый триггер AFTER с последующей проверкой всех ограничений целостности данных;

5) выполняется операторный триггер AFTER.

Триггерные предикаты. Если в триггере указывается совокупность активизирующих триггер DML-операторов (например, INSERT OR DELETE), то для распознавания того, какой конкретно из DML-операторов выполняется над связанной с триггером таблицей, используются триггерные предикаты: INSERTING, DELETING, UPDATING. Они представляют собой логические функции, возвращающие TRUE, если тип активизирующего оператора совпадает с типом предиката, и FALSE – в противном случае. Для задания одних и тех же действий в случае выполнения различных DML-операторов в условном операторе триггерные предикаты объединяются с помощью логических операций.

Псевдозаписи. Для строковых триггеров существуют специальные конструкции, которые позволяют при выполнении DML-операторов над строкой таблицы обращаться как к старым значениям, которые находились в ней до модификации, так и к новым, которые появятся в строке после ее модификации. Эти конструкции называются псевдозаписями и обозначаются old и new. Структура этих псевдозаписей идентична структуре строки модифицируемой таблицы, но оперировать можно только отдельными полями псевдозаписи. Обращение к полям псевдозаписи происходит по следующей схеме: перед old или new ставится символ двоеточие (:), далее через точку указывается название поля. Значения, которые принимают поля псевдозаписи при выполнении активизирующих
DML-операторов, определяются следующим образом.

1. Оператор INSERT – псевдозапись :new эквивалентна вставляемой строке, а псевдозапись :old во всех полях имеет значение NULL.

2. Оператор DELETE – псевдозапись :old эквивалентна удаляемой строке, а псевдозапись :new во всех полях имеет значение NULL.

3. Оператор UPDATE – псевдозапись :new эквивалентна строке, полученной в результате модификации, а псевдозапись :old во всех полях имеет исходное значение строки.

Включение, выключение и удаление триггеров. Хранящийся в БД триггер можно временно отключить, не удаляя его из БД. Для этого используется следующая команда:

```
ALTER TRIGGER имя_триггера DISABLE;
```

Включить триггер через некоторый промежуток времени можно, используя команду

```
ALTER TRIGGER имя_триггера ENABLE;
```

Запретить или разрешить запуск всех триггеров, связанных с некоторой таблицей, можно с помощью команды

```
ALTER TABLE имя_таблицы
{DISABLE | ENABLE} ALL TRIGGERS;
```

где вариант DISABLE используется для отключения, а вариант ENABLE – для включения всех триггеров данной таблицы.

Уничтожение триггера, т. е. удаление триггера из БД осуществляется с помощью следующей команды:

```
DROP TRIGGER имя_триггера;
```

Получение информации о триггерах. Триггеры хранятся в БД, поэтому информацию о них можно получить из представления словаря данных USER_TRIGGERS, например, следующей командой:

```
SELECT * FROM USER_TRIGGERS;
```

П р и м е р ы

1. Создать триггер, который перед вставкой очередной строки в таблицу BOOKS_DELIVERY проверяет наличие указанного кода книги в таблице BOOKS. При отсутствии указанного кода книги в таблице BOOKS должно генерироваться исключение с выдачей соответствующего сообщения.

Добавление новых строк в таблицу BOOKS_DELIVERY выполняется оператором INSERT. Поскольку триггер должен запускаться перед выполнением каждого оператора INSERT, следовательно, он должен быть строковым BEFORE-триггером. Для сохранения целостности данных необходимо проверить, имеются ли вносимые коды книг и в таблице BOOKS. Для этого с помощью однострочного оператора SELECT осуществляется выборка информации из таблицы BOOKS, где в условии выборки используется поле CODE_BOOK псевдозаписи :new. Если количество строк с данным кодом книги в таблице BOOKS окажется рав-

ным нулю, будет сгенерировано исключение и выдано соответствующее сообщение.

Создание триггера TR1 выполняется вводом следующего оператора:

```
CREATE OR REPLACE TRIGGER TR1
   BEFORE INSERT ON BOOKS_DELIVERY
   FOR EACH ROW
DECLARE
   QUANTITY NUMBER(4);
BEGIN
   SELECT COUNT(*) INTO QUANTITY FROM BOOKS
      WHERE CODE_BOOK = :NEW.CODE_BOOK;
   IF QUANTITY = 0 THEN RAISE_APPLICATION_ERROR
      (–20212,'В таблице BOOKS нет информации о данной книге');
   END IF;
END TR1;
```

При создании триггера с помощью SQL*PLUS, необходимо указывать в строке, следующей за последним оператором, косую черту (/), чтобы оператор CREATE … TRIGGER выполнился.

Действие триггера TR1 может быть проверено выполнением следующего оператора, осуществляющего вставку строки в таблицу BOOKS_DELIVERY и тем самым вызывающего активизацию триггера TR2:

```
INSERT INTO BOOKS_DELIVERY VALUES
(21, 15, 'Иванов И. И. ',15, '20-01-06');
```

Поскольку код книги 15 отсутствует в таблице BOOKS, то будет сгенерировано исключение и выдано соответствующее сообщение.

2. Создать триггер, который запрещает вносить в таблицу BOOKS строки со значением поля PRICE больше, чем 5000 руб., а также осуществлять увеличение цены книг, информация о которых хранится в таблице BOOKS, более чем на 20 %. При нарушении данного требования должно генерироваться исключение с выдачей соответствующего сообщения.

Так как внесение новых строк в таблицу BOOKS осуществляется в результате выполнения оператора INSERT, а значение поля PRICE в таблице BOOKS, содержащего цену книги, может быть изменено в результате выполнения оператора UPDATE, то в триггере указывается совокупность запускающих DML-операторов. Поскольку триггер должен запускаться перед выполнением каждого из указанных DML-операторов,

следовательно, он является строковым BEFORE-триггером. Так как действия, выполняемые триггером, различны для каждого из запускающих DML-операторов, модифицирующих таблицу BOOKS, то для распознавания типа DML-оператора используются соответствующие триггерные предикаты INSERTING и UPDAITING. Вследствие того что при вставке новых строк проверке должно быть подвергнуто новое значение поля PRICE, а при модификации значения поля PRICE новое значение должно сравниваться со старым значением, необходимо использовать псевдозаписи :new и :old.

Создание триггера TR2 выполняется вводом следующего оператора:

```
CREATE OR REPLACE TRIGGER TR2
   BEFORE INSERT OR UPDATE OF PRICE ON BOOKS
   FOR EACH ROW
BEGIN
   IF INSERTING THEN
      IF :NEW.PRICE > 5000 THEN
         RAISE_APPLICATION_ERROR
         (–20102, 'В таблицу BOOKS нельзя вносить записи
                  с ценой книги > 5000');
      END IF;
   END IF;
   IF UPDATING THEN
      IF :NEW.PRICE > :OLD.PRICE*1.2 THEN
         RAISE_APPLICATION_ERROR
         (–20103, 'В таблице BOOKS нельзя изменять цену книги
                  более чем на 20 %');
      END IF;
   END IF;
END TR2;
```

Действие триггера TR2 может быть проверено выполнением следующих операторов, которые, осуществляя вставку строк в таблицу BOOKS и обновление строк в таблице BOOKS, тем самым вызывают его активизацию.

Оператор вставки строк в таблицу BOOKS, вызывающий активизацию триггера TR2:

```
INSERT INTO BOOKS VALUES
( 21, 'Дюна', 'Герберт Ф.', 5268, 'Аст',  'Фантастика');
```

Оператор обновления строк в таблице BOOKS, вызывающий активизацию триггера TR2:

```
UPDATE  BOOKS  SET PRICE = 6000;
```

Поскольку эти операторы нарушают требования, предъявляемые к значению и модификации цены книг, то во всех случаях будет сгенерировано исключение и выдано соответствующее сообщение.

3. Создать триггер, который в созданную таблицу STAT, содержащую столбцы

название издательства – PUBLISH_H,
количество книг в жанре «Роман» – QUANTITY_ROM,
количество книг в жанре «Фантастика» – QUANTITY_FAN,

при каждой модификации таблицы BOOKS формирует и заносит в соответствующие столбцы таблицы STAT суммарное количество книг по каждому из издательств в разрезе указанных тематик: «Роман» и «Фантастика».

Модификация таблицы BOOKS осуществляется выполнением следующих DML-операторов: INSERT, DELETE или оператора UPDATE, модифицирующего значения столбца GENRE в таблице BOOKS. Так как действия по формированию информации таблицы STAT выполняются после выполнения каждого из модифицирующих таблицу BOOKS операторов, то по типу это операторный AFTER-триггер. Поскольку действия, выполняемые триггером, одинаковы для всех типов активизирующих его операторов, то триггерные предикаты не используются. Перед созданием триггера должна быть создана таблица STAT.

Создание таблицы STAT может быть выполнено вводом следующей совокупности операторов:

```
DROP  TABLE  STAT;
CREATE  TABLE  STAT
(PUBLISH_H  VARCHAR2(15),
 QUANTITY_ROM  NUMBER(7),
 QUANTITY_FAN  NUMBER(7)
);
```

Создание триггера TR3 выполняется вводом следующего оператора:

```
CREATE  OR  REPLACE  TRIGGER  TR3
   AFTER  INSERT  OR  DELETE  OR  UPDATE  OF  GENRE
   ON  BOOKS
DECLARE
   CURSOR  V1  IS  SELECT  PUBLISH_HOUSE,
   COUNT(TITLE)  QUANTITY1
      FROM  BOOKS  WHERE  GENRE = 'Роман'
```

```
      GROUP BY PUBLISH_HOUSE;
   CURSOR V2 IS SELECT PUBLISH_HOUSE,
   COUNT(TITLE) QUANTITY2
      FROM BOOKS WHERE GENRE = 'Фантастика'
      GROUP BY PUBLISH_HOUSE;
BEGIN
DELETE FROM STAT;
   FOR Z1 IN V1 LOOP
      INSERT INTO STAT VALUES(Z1.PUBLISH_HOUSE,
      Z1.QUANTITY1, 0);
   END LOOP;
   FOR Z1 IN V2 LOOP
      UPDATE STAT SET QUANTITY_FAN = Z1. QUANTITY2
      WHERE PUBLISH_H = Z1.PUBLISH_HOUSE;
      IF SQL%NOTFOUND THEN
         INSERT INTO STAT VALUES(Z1.PUBLISH_HOUSE, 0,
         Z1.QUANTITY2);
      END IF;
   END LOOP;
END TR3;
```

Действие триггера может быть проверено выполнением следующих операторов, которые, осуществляя вставку строк в таблицу BOOKS, удаление строк и обновление строк в таблице BOOKS, тем самым вызывают активизацию триггера TR3.

Операторы вставки строк в таблицу BOOKS, вызывающие активизацию триггера TR3:

```
INSERT INTO BOOKS VALUES
(46, 'Еретики Дюны', 'Герберт Ф.', 368, 'Аст', 'Фантастика');
INSERT INTO BOOKS VALUES(42, 'Ингвар и Ольха',
 'Никитин Ю.', 168, 'Аст', 'Роман');
```

Операторы удаления строк из таблицы BOOKS, вызывающие активизацию триггера TR3:

```
DELETE BOOKS WHERE AUTHOR = 'Герберт Ф.';
DELETE BOOKS WHERE TITLE = 'Казаки';
```

Оператор модификации строк в таблице BOOKS, вызывающие активизацию триггера TR3:

```
UPDATE BOOKS SET GENRE='Фантастика' WHERE TITLE =
 'Ингвар и Ольха';
```

Просмотр информации в таблице STAT можно выполнить следующим оператором:

```
SELECT * FROM STAT;
```

13.7. ХРАНИМЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Функции и процедуры (подпрограммы) представляют собой оформленные специальным образом именованные блоки PL/SQL, которые могут быть вызваны для выполнения и которым могут быть переданы параметры. Как правило, процедуры и функции реализуют определенное законченное действие над некоторыми объектами БД.

Типы процедур и функций. Существуют два вида процедур и функций: локальные и хранимые. Локальные процедуры и функции могут использоваться только в тех блоках, где они определены. Хранимые процедуры и функции компилируются и хранятся в БД в скомпилированном виде. При необходимости они могут быть вызваны для выполнения анонимными и именованными блоками PL/SQL, процедурами и функциями обоих видов, триггерами, а также из интерактивной среды SQL* PLUS. Помимо этого хранимая функция может быть вызвана и в операторе SQL.

Создание хранимых процедур и функций. Для создания хранимой процедуры используется следующий общий синтаксис:

```
CREATE [OR REPLACE] PROCEDURE имя_процедуры
[(параметр1  [, параметр2, ...])] IS
    [раздел локальных объявлений]
BEGIN
    исполняемый раздел
[ EXCEPTION
    раздел обработки исключений]
END [имя процедуры];
```

Для создания хранимой функции используется следующий общий синтаксис:

```
CREATE [OR REPLACE] FUNCTION имя_функции
[(параметр1[, параметр2, …]] RETURN тип_данных IS
   [раздел локальных объявлений]
BEGIN
  исполняемый раздел
[ EXCEPTION
  раздел обработки исключений]
```

END [имя функции];

Хранимые процедуры и функции, вызываемые блоками PL/SQL, процедурами и функциями, триггерами, вызываются заданием имени функции или процедуры с указанием списка фактических параметров. Если вызывается функция, то она должна быть частью выражения; процедура вызывается как отдельный оператор.

Для вызова из SQL* PLUS хранимой процедуры используется следующая форма записи:

EXECUTE имя_процедуры (список_фактических_параметров);

Поскольку функция из интерактивного редактора не может быть вызвана непосредственно, для ее вызова необходимо использовать блок PL/SQL, анонимный или именованный, или оператор SQL.

Параметры процедур и функций (подпрограмм). Для передачи информации в подпрограмму используются параметры. Переменные или выражения, перечисленные в списке параметров в спецификации подпрограммы, называются формальными параметрами, а перечисленные в списке параметров при вызове подпрограммы называются фактическими аргументами. При вызове подпрограммы фактические аргументы вычисляются и результирующие значения присваиваются формальным параметрам, причем производятся необходимые преобразования типов, поэтому формальные параметры и фактические аргументы должны иметь совместимые типы.

Список параметров представляет собой перечисление этих параметров через запятую. Каждый формальный параметр может быть описан следующим синтаксисом:

Имя_параметра [вид] тип [{:= | DEFAULT} значение];

Параметр ***вид*** определяет режим передачи параметра. Имеются три режима передачи параметров: IN (по умолчанию), OUT и IN OUT. Они используются для обозначения соответствия входных, выходных и модифицируемых параметров. Желательно не использовать режимы OUT и IN OUT при написании функций, чтобы избежать побочных эффектов.

Фактический аргумент, указываемый на месте IN-параметра, должен быть константой, литералом, проинициализированной переменной либо выражением, и в отличие от OUT- и IN OUT-параметров IN-параметр может иметь значение по умолчанию. Если параметр передается с вариантом IN, то в подпрограмме ему нельзя присваивать значение.

На месте OUT- или IN OUT-параметра может быть указана только переменная. Как и переменные, OUT-параметры инициализируются

NULL-значением, и тип OUT-параметра не может быть подтипом, определенным как NOT NULL. В противном случае генерируется исключение VALUE_ERROR.

Если при выполнении процедуры или функции возникают исключительные ситуации, то управление передается в вызывающий блок. Когда осуществляется нормальный выход из подпрограммы, то фактическим OUT- и IN OUT-аргументам присваиваются значения, а если возникают необработанные исключения, то значения не присваиваются.

Параметр ***тип*** определяет допустимый тип данных для параметра. В качестве типа параметра могут использоваться практически все основные типы данных языка. Однако если используются типы CHAR, VARCHAR2 или NUMBER, то нельзя указывать размерность для этих типов данных, а для типа NUMBER – точность и масштаб.

Порядок задания параметров. При вызове подпрограмм можно записать список фактических аргументов, используя либо позиционную, именованную, смешанную нотации, либо передачу параметров по умолчанию:

1) позиционная нотация – это передача списка параметров простым перечислением, причем типы, количество и порядок следования параметров должны соответствовать объявленным раньше;

2) в именованной нотации стрелка => используется как оператор ассоциации, который связывает формальный параметр слева от стрелки с фактическим аргументом справа от нее. При именованной нотации параметры могут указываться в любом порядке;

3) нотации могут смешиваться (смешанная нотация), но в этом случае позиционное указание параметров должно предшествовать именованному;

4) существует возможность передачи параметров по умолчанию. При этом формальным параметрам должны быть присвоены значения либо оператором присваивания, либо через ключевое слово DEFAULT, и они в списке фактических параметров должны быть записаны последними.

Подпрограммы и зависимости. Перекомпиляция подпрограмм. Хранимые функции и процедуры хранятся в скомпилированном виде в БД. При этом, как правило, их исполнение затрагивает некоторые объекты БД. Для обеспечения достоверности работы таких процедур или функций система постоянно отслеживает для каждой процедуры или функции состояние объектов, с которыми она связана. Если какой-то из связанных с ней объектов подвергается модификации с помощью оператора DDL, то процедура или функция объявляется системой недействительной или недостоверной (INVALID). В этом случае процедуру или

функцию, объявленную недостоверной, надо обязательно перекомпилировать.

Для того чтобы перекомпилировать хранимую процедуру, используется команда:

ALTER PROCEDURE имя_процедуры COMPILE;

Команда ALTER FUNCTION перекомпилирует хранимую функцию:

ALTER FUNCTION имя_функции COMPILE;

Удаление подпрограмм из БД осуществляется следующей командой:

DROP {PROCEDURE | FUNCTION} имя_подпрограммы;

Получение информации о хранимых процедурах и функциях. Информацию о процедурах и функциях можно получить из представления словаря данных USER_OBJECTS, например, следующей командой:

SELECT * FROM USER_OBJECTS;

П р и м е р ы

1. Создать хранимую процедуру, увеличивающую стоимость указанной книги в таблице BOOKS на 10 %. Параметр: код книги.

Вариант 1.

```
CREATE OR REPLACE PROCEDURE INCREASE
(CODE_BOOK  BOOKS.CODE_BOOK%TYPE) AS
   Q NUMBER(1) := 0;
BEGIN
   SELECT COUNT(*) INTO Q FROM BOOKS
      WHERE BOOKS.CODE_BOOK = INCREASE.CODE_BOOK;
   IF Q <> 0 THEN
      UPDATE BOOKS SET PRICE = PRICE + PRICE*0.1
         WHERE BOOKS.CODE_BOOK = INCREASE.CODE_BOOK;
   ELSE
     RAISE_APPLICATION_ERROR(-20105, 'В таблице BOOKS
            отсутствует книга с указанным кодом');
   END IF;
END INCREASE;
```

Выполнение процедуры INCREASE (вариант 1) реализуется оператором

```
EXECUTE INCREASE (1);
```

Вариант 2.

```
CREATE OR REPLACE PROCEDURE INCREASE
(CODE_BOOK BOOKS.CODE_BOOK%TYPE) AS
BEGIN
   UPDATE BOOKS SET PRICE = PRICE*1.1
      WHERE BOOKS.CODE_BOOK = INCREASE.CODE_BOOK;
      IF SQL%NOTFOUND THEN
         RAISE_APPLICATION_ERROR(-20105, 'В таблице BOOKS
            отсутствует книга с указанным кодом');
      END IF;
END INCREASE;
```

Выполнение процедуры INCREASE (вариант 2) реализуется оператором

```
EXECUTE INCREASE (1);
```

2. Создать хранимую процедуру, которая при поступлении книг к продавцу либо добавляет указанное количество к количеству уже имеющихся у продавца таких же книг, обновляя соответствующую запись в таблице BOOKS_DELIVERY, либо вставляет новую запись в таблицу BOOKS_DELIVERY, если у продавца книг такого наименования нет. Параметры: код книги, количество единиц, фамилия продавца.

Создание процедуры:

```
CREATE OR REPLACE PROCEDURE ADD_BOOKS
(CODE_BOOK NUMBER, QUANTITY NUMBER,
SALESMAN VARCHAR2) IS
   Q NUMBER(5) := 0;
BEGIN
   SELECT COUNT(*) INTO Q FROM BOOKS_DELIVERY
   WHERE
   BOOKS_DELIVERY.CODE_BOOK = ADD_BOOKS.CODE_BOOK
   AND
   BOOKS_DELIVERY.SALESMAN = ADD_BOOKS.SALESMAN;
   IF Q <> 0 THEN
    UPDATE BOOKS_DELIVERY SET
    QUANTITY = QUANTITY + ADD_BOOKS.QUANTITY
    WHERE
       BOOKS_DELIVERY.CODE_BOOK =
       ADD_BOOKS.CODE_BOOK
    AND
```

```
    BOOKS_DELIVERY.SALESMAN = ADD_BOOKS.SALESMAN;
  ELSE
    INSERT INTO BOOKS_DELIVERY VALUES
     (CODE_OP.NEXTVAL,
     ADD_BOOKS.CODE_BOOK, ADD_BOOKS.SALESMAN,
     ADD_BOOKS.QUANTITY, SYSDATE);
  END IF;
END ADD_BOOKS;
```

Выполнение процедуры ADD_BOOKS реализуется следующими операторами:

```
EXECUTE ADD_BOOKS(1, 10, 'Иванов И. И.');
EXECUTE ADD_BOOKS(5, 10, 'Иванов И. И.');
```

При первом вызове процедуры осуществляется добавление 10 книг к тем, что уже имеются у указанного продавца. При втором вызове в таблицу BOOKS_DELIVERY вставляется новая запись.

3. Создать хранимую функцию, которая проверяет, входит ли для указанного продавца общее количество имеющихся у него книг в определенный диапазон, и если нет, то выдает соответствующее сообщение. Параметр: фамилия продавца. Минимальное и максимальное значения для проверки устанавливаются в теле функции с помощью переменных MIN_QUANTITY и MAX_QUANTITY.

Создание функции:

```
CREATE OR REPLACE FUNCTION TEST_B
(SALESMAN VARCHAR2)
   RETURN BOOLEAN IS
   MIN_QUANTITY NUMBER(5) :=10;
   MAX_QUANTITY NUMBER(5) := 100;
   Q NUMBER(5);
BEGIN
   SELECT SUM(QUANTITY) INTO Q FROM
   BOOKS_DELIVERY WHERE
   BOOKS_DELIVERY.SALESMAN = TEST_B.SALESMAN;
RETURN (Q >= MIN_QUANTITY AND Q <= MAX_QUANTITY);
END TEST_B;
```

Вызов функции TEST_B осуществляется с помощью тестирующей программы, приведенной ниже.

```
DECLARE
  ADD_SALESMAN VARCHAR2(20);
```

```
  ADD_QUANTITY  NUMBER(5);
  ADD_CODE_BOOK  NUMBER(5);
BEGIN
   ADD_SALESMAN := 'Иванов И. И.';
   ADD_QUANTITY := 20;
   ADD_CODE_BOOK := 3;
   IF  TEST_B(ADD_SALESMAN)  THEN
      UPDATE  BOOKS_DELIVERY  SET
      QUANTITY = QUANTITY + ADD_QUANTITY
      WHERE
       BOOKS_DELIVERY.CODE_BOOK = ADD_CODE_BOOK
      AND  BOOKS_DELIVERY.SALESMAN =
       ADD_SALESMAN;
END  IF;
END;
```

Программа добавляет указанное количество (переменная ADD_QUANTITY) книг указанному продавцу (переменная ADD_SALESMAN), если общее количество имеющихся у него книг не превосходит 100 и не менее 10.

13.8. ПАКЕТЫ

Пакет (package) – это совокупность процедур, функций и других программных объектов, предназначенная для решения определенного класса задач. Пакет позволяет объединить и хранить как отдельную единицу в базе данных несколько различных элементов: процедур, функций, типов данных, переменных, констант, курсоров, исключений. В целом пакет можно рассматривать как именованный раздел объявлений блока. Размещение типов, переменных, курсоров, процедур и функций в заголовке пакета позволяет ссылаться на них из других блоков PL/SQL. В отличие от процедур и функций, которые могут быть как локальными, так и хранимыми, пакеты никогда не бывают локальными, а бывают только хранимыми. Пакеты также нельзя вкладывать друг в друга, нельзя вызывать и пакетам нельзя передавать параметры.

Структура пакета. Пакет, или модуль, состоит из двух различных частей: заголовка, или спецификации, пакета (package) и тела (package body), каждая из которых хранится по отдельности в словаре данных. ***Заголовок пакета*** описывает его интерфейс, т. е. все те элементы, которые могут быть доступны всем пользователям пакета. Объявляемые в спецификации пакета объекты называются *общими* (public). Среди них присут-

ствуют описания общедоступных типов и объектов и спецификации общедоступных функций и процедур. К общим объектам можно обращаться как извне пакета, так и из других объектов в пакете.

Заголовок пакета создается следующей структурой:

```
CREATE [OR REPLACE] PACKAGE имя_пакета {IS|AS}
 --определение типа;
 --определение переменной, константы;
 --объявление курсора;
 --объявление исключительной ситуации;
 --объявление функции;
 --объявление процедуры
END [имя_пакета];
```

Наличие в заголовке модуля всех вышеперечисленных элементов необязательно. Например, заголовок модуля может состоять только из объявлений процедур и функций. Элементы пакета могут размещаться в нем в любом порядке, но если какие-то элементы пакета ссылаются на другие объекты того же пакета, то последние должны быть объявлены до ссылок на эти объекты. Объявления всех процедур и функций должны быть предварительными, т. е. в отличие от раздела объявлений блока, где могут находиться как предварительные объявления, так и реальные тексты процедур и функций, здесь содержатся только объявления подпрограмм.

Создадим заголовок пакета BOOKS_DELIV, поместив в него описания двух процедур и функции из раздела 13.7.

```
CREATE  OR  REPLACE PACKAGE  BOOKS_DELIV IS
  PROCEDURE  INCREASE
     (CODE_BOOK  NUMBER);
  PROCEDURE  ADD_BOOKS (CODE_BOOK  NUMBER, QUANTITY
     NUMBER,  SALESMAN  VARCHAR2);
  FUNCTION  TEST_B (SALESMAN  VARCHAR2)
  RETURN BOOLEAN;
END BOOKS_DELIV;
```

Тело пакета – это объект словаря данных, хранящийся отдельно от заголовка пакета. Успешная компиляция тела пакета возможна лишь при условии успешной компиляции заголовка пакета.

Тело пакета создается следующей структурой:

```
CREATE [OR REPLACE] PACKAGE  BODY имя {IS|AS}
  --описания закрытых типов и объектов;
  --определения локальных функций и процедур;
```

```
 --определения общедоступных функций и процедур;
END [имя];
```

Тело пакета реализует спецификацию пакета. В нем должны быть описаны все процедуры и функции, предварительно объявленные в заголовке пакета. При этом обязательно должны совпадать названия подпрограмм, набор параметров, порядок их следования и описание. Кроме этого, можно также объявить и определить дополнительные объекты пакета, которые называются *личными* (private) или *закрытыми*. Так как личные объекты объявляются в теле пакета, а не в его спецификации, к ним можно обращаться только из объектов пакета. Доступ к ним из других блоков PL/SQL невозможен. Тело пакета не является обязательной его частью и при отсутствии в заголовке объявлений процедур и функций, тело может отсутствовать. В этом случае в заголовке пакета можно объявить переменные, типы, курсоры и исключительные ситуации, которые будут рассматриваться как глобальные переменные и могут не объявляться в использующих их других процедурах и функциях.

Создадим тело пакета BOOKS_DELIV, поместив в него определения двух процедур и функции из раздела 13.7.

```
CREATE OR REPLACE PACKAGE BODY BOOKS_DELIV IS
 PROCEDURE INCREASE
 (CODE_BOOK NUMBER) AS
 BEGIN
    UPDATE BOOKS SET PRICE = PRICE*1.1
       WHERE BOOKS.CODE_BOOK = INCREASE.CODE_BOOK;
       IF SQL%NOTFOUND THEN
          RAISE_APPLICATION_ERROR(-20105, 'В таблице BOOKS
               отсутствует книга с указанным кодом');
       END IF;
 END INCREASE;
 PROCEDURE ADD_BOOKS (CODE_BOOK NUMBER, QUANTITY
 NUMBER, SALESMAN VARCHAR2) IS
    Q NUMBER(5) := 0;
 BEGIN
    SELECT COUNT(*) INTO Q FROM BOOKS_DELIVERY
    WHERE
    BOOKS_DELIVERY.CODE_BOOK = ADD_BOOKS.CODE_BOOK
    AND
    BOOKS_DELIVERY.SALESMAN = ADD_BOOKS.SALESMAN;
    IF Q <> 0 THEN
```

```
      UPDATE BOOKS_DELIVERY SET
      QUANTITY = QUANTITY + ADD_BOOKS.QUANTITY
      WHERE BOOKS_DELIVERY.CODE_BOOK
       = ADD_BOOKS.CODE_BOOK
      AND
      BOOKS_DELIVERY.SALESMAN = ADD_BOOKS.SALESMAN;
    ELSE
      INSERT INTO BOOKS_DELIVERY VALUES
       (CODE_OP.NEXTVAL,
       ADD_BOOKS.CODE_BOOK, ADD_BOOKS.SALESMAN,
       ADD_BOOKS.QUANTITY, SYSDATE);
    END IF;
END ADD_BOOKS;
FUNCTION TEST_B (SALESMAN VARCHAR2)
RETURN BOOLEAN IS
    MIN_QUANTITY NUMBER(5) :=10;
    MAX_QUANTITY NUMBER(5) := 100;
    Q NUMBER(5);
BEGIN
    SELECT SUM(QUANTITY) INTO Q FROM
    BOOKS_DELIVERY WHERE
    BOOKS_DELIVERY.SALESMAN = TEST_B.SALESMAN;
RETURN (Q >= MIN_QUANTITY AND Q <= MAX_QUANTITY);
END TEST_B;
END BOOKS_DELIV;
```

При создании спецификации или тела пакета с помощью SQL*PLUS, необходимо указывать в строке, следующей за последним оператором, косую черту (/), чтобы оператор CREATE … PACKAGE выполнился. Процедуры и функции модуля внутри модуля могут быть перегруженными. Это означает, что может существовать несколько процедур и функций с одним и тем же именем, но разными списками параметров. Это очень удобно, так как позволяет выполнять одну и ту же операцию над объектами различных типов. В этом случае все различные варианты процедур или функций описываются в заголовке пакета, а в его теле размещаются тексты этих подпрограмм.

Следует иметь в виду, что пакет допускает наличие перегруженных функций и процедур при выполнении ряда ограничений.

1. Нельзя перегружать две подпрограммы, если их параметры отличаются именами. Например, при наличии функции – FUNCTION AA(A1

IN NUMBER) нельзя объявить еще одну функцию – FUNCTION AA(K1 IN NUMBER).

2. Нельзя перегружать две подпрограммы, если их параметры отличаются видами. Например, при наличии функции – FUNCTION AA(A1 IN NUMBER) нельзя объявить еще одну функцию – FUNCTION AA(A1 OUT NUMBER).

3. Запрещается перегружать функции, если типы параметров относятся к одному семейству типов данных. Например, типы CHAR и VARCHAR2 входят в одно и то же семейство.

4. Нельзя перегружать функции, если они отличаются лишь типом возвращаемых данных.

Любой объект, объявленный в заголовке пакета, находится в области действия, и видим вне границ этого пакета. Для обращения к такому объекту необходимо перед именем объекта указать имя пакета, отделив его от имени объекта точкой (имя_пакета.имя_процедуры или имя_пакета.имя_функции). Например, процедуру INCREASE можно вызвать из анонимного блока PL/SQL следующим образом:

```
BEGIN
    BOOKS_DELIV.INCREASE(1);
END;
```

Приведем пример вызова функции TEST_B.

```
DECLARE
  ADD_SALESMAN  VARCHAR2(20);
  ADD_QUANTITY  NUMBER(5);
  ADD_CODE_BOOK  NUMBER(5);
BEGIN
    ADD_SALESMAN := 'Иванов И. И.';
    ADD_QUANTITY := 20;
    ADD_CODE_BOOK := 3;
    IF  BOOKS_DELIV.TEST_B (ADD_SALESMAN)  THEN
        UPDATE  BOOKS_DELIVERY  SET
        QUANTITY = QUANTITY + ADD_QUANTITY
        WHERE
         BOOKS_DELIVERY.CODE_BOOK = ADD_CODE_BOOK
        AND  BOOKS_DELIVERY.SALESMAN =
         ADD_SALESMAN;
END  IF;
END;
```

При этом вызов процедуры аналогичен вызову обычной хранимой процедуры. При вызове функций и процедур пакета список параметров подчиняется тем же условиям, что и список параметров обычных функций и процедур. Для модульных подпрограмм могут задаваться параметры по умолчанию, и вызывать такие подпрограммы можно при помощи как позиционного, так и именованного представления. Если процедура модуля использует объявления, заданные в заголовке, то эти объявления можно использовать как с уточняющими именами, так и без них.

Создание и инициализация пакета. Для создания пакета необходимо выполнить два отдельных шага:

1. Создать спецификацию пакета при помощи команды CREATE PACKAGE. В спецификации пакета объявить все общедоступные программные объекты.

2. Создать тело пакета при помощи команды CREATE PACKAGE BODY. В теле пакета объявить и определить программные объекты двух типов:

1) общедоступные объекты, которые были объявлены в спецификации пакета;

2) закрытые объекты пакета, к которым можно обращаться только из других объектов пакета, поскольку извне они недоступны.

Пакет хранится в скомпилированном виде (*p*-код) в словаре данных и при обращении к некоторому элементу пакета в первый раз конкретизируется (instantiated). Это означает, что пакет загружается в оперативную память, а затем запускается его *p*-код; при этом осуществляется инициализация пакета, при которой под переменные, определенные в пакете, выделяется память. Модуль всегда хранится в оперативной памяти в одном экземпляре, и у каждого сеанса, который использует данный модуль, будет собственная копия модульных переменных. Подобный подход гарантирует, что два сеанса, выполняющие подпрограммы одного и того же модуля, будут использовать различные области памяти.

Преимущества пакетов. Использование пакетов является альтернативой созданию процедур и функций как независимых объектов схем. Пакеты имеют преимущества перед независимыми процедурами и функциями:

1. Позволяют инкапсулировать логически связанные подпрограммы и переменные, относящиеся к одному или нескольким приложениям, что способствует более эффективной реализации программных продуктов.

2. Разделение объявлений процедур, переменных, констант и курсоров на общедоступные и личные позволяет скрывать детали реализации отдельных подпрограмм пакета.

3. Когда какая-либо процедура пакета вызывается в первый раз, весь пакет целиком загружается в память. Это выполняется как одна операция, в отличие от отдельных операций загрузки, выполняемых для хранимых процедур.

4. Организация модуля позволяет модифицировать объекты внутри пакетов без перекомпиляции зависимых объектов схем. Следовательно, можно изменить тело модуля, не меняя его заголовка, что и является преимуществом пакета. При этом остальные объекты, зависящие от заголовка модуля, перекомпилировать не надо. Помимо этого, пакеты останавливают каскадные зависимости и тем самым избегают излишних перекомпиляций.

5. Пакеты могут содержать глобальные переменные и курсоры, которые существуют в течение всего сеанса и доступны всем процедурам и функциям пользователя. В каждый момент времени хранится одна копия пакета, которая используется различными сеансами.

Зависимости и перекомпиляция модуля. Как отмечалось ранее, пакеты хранятся в скомпилированном виде в словаре данных. При этом, как правило, их исполнение затрагивает некоторые объекты БД. Для обеспечения достоверности работы пакета система постоянно отслеживает для каждого элемента пакета состояние объектов, с которыми он связан. При этом следует иметь в виду, что тело модуля зависит от используемых объектов и от заголовка пакета, а заголовок не зависит ни от чего. При изменении заголовка тело пакета автоматически становится недостоверным. Если какой-то из связанных с пакетом объектов подвергается модификации с помощью одного из следующих операторов DDL (ALTER, DROP, REPLACE), то тело пакета объявляется системой недействительным или недостоверным (INVALID). В этом случае пакет, объявленный недостоверным, после устранения причин, вызвавших такую ситуацию, надо обязательно перекомпилировать.

Для перекомпиляции пакета используется команда:

ALTER PACKAGE имя_пакета COMPILE;

Удаление пакета или одной из его частей из БД осуществляется следующей командой:

DROP { PACKAGE | PACKAGE BODY } имя_пакета;

Определить достоверность объектов можно, используя следующий оператор:

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
 FROM USER_OBJECTS WHERE OBJECT_NAME IN (‘BOOKS’,
```

‘BOOKS_DELIVERY’, ‘BOOKS_DELIV’);

13.9. ОБЪЕКТЫ

В СУБД Oracle обеспечивается возможность создания, хранения объектных данных и работы с ними. Управление объектными данными аналогично управлению реляционными данными. Для манипулирования как реляционными, так и объектными данными в объектно-реляционной базе данных Oracle используются языки SQL и PL/SQL.

Объекты (objects) представляют собой совокупность данных, объединенных с методами обработки этих данных, и отражают свойства реальных сущностей и операции, выполняемые над этими сущностями. Данные, входящие в объект, носят название ***атрибутов*** или ***свойств***. В качестве ***методов*** обычно используются процедуры и функции. Управление атрибутами объекта осуществляется только через методы.

Для работы с объектами вначале необходимо определить объектный тип (object type), который описывает атрибуты и методы конкретного вида объектов, после чего можно объявлять переменные, или экземпляры, данного типа. Каждый экземпляр имеет собственную область памяти и копию атрибутов объекта.

В объявляемых объектных типах можно использовать в качестве параметра ранее созданные объектные типы. Созданные объектные типы могут быть типом для переменных, использоваться в качестве атрибутов для других типов, являться строкой в таблице, являться столбцом в таблице, могут быть частью объектного представления.

Создание объектного типа. Объектный тип состоит:

1) из описания (заголовка, спецификации) объектного типа;
2) тела объектного типа.

В описании типа содержатся атрибуты и предварительные объявления методов, а тело состоит из определений методов. Каждая из этих частей компилируется отдельно, но пока не будет откомпилирован заголовок объектного типа, не будет компилироваться и его тело.

Для создания заголовка объектного типа используется следующий синтаксис:

```
CREATE [OR REPLACE] TYPE [схема.] имя_типа  AS OBJECT
(имя_атрибута тип_данных [, имя_атрибута тип_данных] …]
[{MAP | ORDER} MEMBER  описание_функции]
[MEMBER {описание_процедуры |  описание_функции}
[, MEMBER {описание_процедуры |  описание_функции}]...]
[PRAGMA RESTRICT_REFERENCES (имя_метода, ограничения)
```

[, PRAGMA RESTRICT_REFERENCES (имя_метода, ограничения)]...]
…);

где имя_типа – имя нового объектного типа, а схема – его владелец. Сначала через запятую перечисляются атрибуты типа:

имя_атрибута тип_данных

где имя_атрибута – это имя атрибута, а тип_данных – встроенный тип PL/SQL, или ранее определенный пользователем тип данных, или ссылка на объектный тип. После атрибутов указываются методы объектного типа. При описании методов каждый из них, кроме последнего, отделяется запятой и выглядит как обычная хранимая подпрограмма PL/SQL с добавлением ключевого слова MEMBER.

Прагма RESTRICT_REFERENCES используется для определения порядка вызова метода из SQL-оператора.

Для создания тела объектного типа используется следующий синтаксис:

```
CREATE [OR REPLACE] TYPE BODY имя_типа AS
[{MAP | ORDER} MEMBER определение_функции]
[MEMBER {определение_процедуры | определение_функции}
[MEMBER {определение_процедуры | определение_функции}]...]
END;
```

причем в теле объектного типа должны быть реализованы все объявленные в заголовке процедуры и функции.

З а м е ч а н и я

1. При создании спецификации или тела пакета с помощью SQL*PLUS, необходимо указывать в строке, следующей за последним оператором, косую черту (/), чтобы оператор CREATE … TYPE выполнился.

2. Описание атрибутов производится так же, как и описание полей типа записи PL/SQL или столбцов таблицы в операторе CREATE TABLE, но в отличие от полей записи атрибуты объектного типа нельзя ограничивать как NOT NULL и инициализировать значениями по умолчанию.

3. Так как объектные типы хранятся в словаре данных, то типы данных для атрибутов должны быть только те, которые есть в языке SQL, за исключением типов данных: LONG, LONG RAW, NCHAR, NVARCHAR2, NCLOB и ROWID.

4. Не допускается использование типов данных, доступных в языке PL/SQL: BINARY_INTEGER, BOOLEAN, PLS_INTEGER, RECORD, REF CURSOR.

5. Запрещается использование в атрибутах объектного типа атрибутов %TYPE и %ROWTYPE, но разрешается использование атрибута %TYPE с атрибутами экземпляра объектного типа.

6. Функции MAP и ORDER используются для задания порядка сортировки в данном объектном типе. Эти функции обсуждаются ниже.

7. Допускается создание объектных типов путем предварительного объявления типа, которое аналогично предварительному объявлению процедуры или метода:

```
CREATE TYPE имя_типа;
```

8. Тело объектного типа не может содержать операторы тех методов, которые не указаны в спецификации типа.

Создадим объектный тип для сущности «книга». Атрибутами данного типа являются код книги, название книги, фамилия автора, цена книги. Объявим два метода: FORMATTED_NAME и CHANGE_PRICE, первый из которых возвращает фамилию автора и название книги, а второй обновляет поле PRICE значением, указанным в поле NEW_PRICE. В качестве имени объектного типа используем идентификатор BOOK_T. Следующая последовательность операторов создает спецификацию объектного типа BOOK_T.

```
CREATE OR REPLACE TYPE BOOK_T AS OBJECT
( ID NUMBER(2),                 -- код книги
  TITLE VARCHAR2(15),           -- название книги
  AUTHOR VARCHAR2(15),          -- фамилия автора
  PRICE NUMBER(7),              -- цена книги
  MEMBER FUNCTION FORMATTED_NAME
  RETURN VARCHAR2,
  MEMBER PROCEDURE CHANGE_PRICE
  (NEW_PRICE IN NUMBER)
);
```

Создадим тело объектного типа BOOK_T:

```
CREATE OR REPLACE TYPE BODY BOOK_T AS
  MEMBER FUNCTION FORMATTED_NAME
  RETURN VARCHAR2 IS
   BEGIN
    RETURN SELF.AUTHOR||SELF.TITLE;
```

```
  END   FORMATTED_NAME;
  MEMBER PROCEDURE CHANGE_PRICE
  (NEW_PRICE  IN  NUMBER)  IS
  BEGIN
    PRICE := NEW_PRICE;
  END   CHANGE_PRICE;
END;
```

Объявление и инициализация объектов. Как и другие переменные PL/SQL, объект описывается в разделе объявлений блока, после имени объекта указывается соответствующий тип. Например:

```
DECLARE
  BOOK  BOOK_T;
BEGIN
  NULL;
END;
```

В этом блоке BOOK описывается как экземпляр объектного типа BOOK_T. По правилам PL/SQL экземпляр объекта, объявленный таким образом, инициализируется NULL-значением, т. е. создается NULL-объект, на атрибуты которого ссылаться нельзя. Подобная попытка приведет к ошибке. В следующем примере система выдаст ошибку «Ссылка на неинициализированный составной тип»:

```
DECLARE
  BOOK BOOK_T;
BEGIN
  BOOK. TITLE:= 'Война и мир';
END;
```

Для инициализации объектов используется специальная функция-***конструктор*** (constructor), возвращающая инициализированный объект. Система создает функцию-конструктор по умолчанию, имя функции совпадает с именем объектного типа, а список параметров – со списком атрибутов.

На атрибуты объекта, как и на поля записи, можно ссылаться, используя уточняющее имя, записываемое перед именем атрибута через точку. Создадим инициализированный экземпляр типа BOOK_T и изменим значение атрибута TITLE.

```
DECLARE
  BOOK  BOOK_T := BOOK_T (10, 'Казаки', 'Толстой Л.', 5000);
BEGIN
```

```
  BOOK.TITLE := 'Война и мир';
END;
```

Чтобы проверить, является объект NULL-объектом или нет, используется условие IS NULL. Например:

```
DECLARE
  BOOK  BOOK_T := BOOK_T (10, 'Казаки', 'Толстой Л.', 5000);
BEGIN
  IF  BOOK  IS  NULL  THEN
    RAISE_APPLICATION_ERROR(-20000,    'BOOK  IS  NULL');
  ELSE
    BOOK.TITLE := 'Ярость';
  END IF;
END;
```

После создания каждый экземпляр объекта находится в определенном, свойственном только ему состоянии. Для модификации его состояния используются методы объекта, которые должны ссылаться на конкретный экземпляр объекта. Следовательно, при вызове метода нужно использовать следующий синтаксис:

имя_объекта.имя_метода

где имя_объекта – имя объектной переменной, а имя_метода – имя вызываемого метода. Если метод не содержит аргументов, его можно вызывать без круглых скобок, как и обычную процедуру PL/SQL, либо со скобками и пустым списком аргументов. Это продемонстрировано на примере следующего блока PL/SQL. Для проверки правильности работы методов создадим специальную таблицу TEST_T, содержащую два поля: фамилию автора и название книги, цену книги.

```
CREATE TABLE TEST_T(FORM VARCHAR2(40), PRICE NUMBER
(7));
```

```
DECLARE
  BOOK1 BOOK_T := BOOK_T (10, 'Казаки', 'Толстой Л.', 5000);
  BOOK2 BOOK_T:= BOOK_T (20, 'Война и мир', 'Толстой Л.', 15000);
  FORM1 VARCHAR(40);
  FORM2 VARCHAR(40);
BEGIN
  -- Изменяет значение атрибута PRICE.
  BOOK1. CHANGE_PRICE(7000);
  BOOK2. CHANGE_PRICE(17000);
  -- Выбираем фамилию автора и название книги.
```

```
  FORM1 := BOOK1.FORMATTED_NAME;
  FORM2 := BOOK2.FORMATTED_NAME();
  -- Полученные данные помещаем в таблицу TEST_T.
  INSERT INTO TEST_T VALUES(FORM1, BOOK1.PRICE);
  INSERT INTO TEST_T VALUES(FORM2, BOOK2.PRICE);
END;
```

Осуществить проверку правильности работы методов можно, выбрав из таблицы TEST_T информацию оператором

```
SELECT * FROM TEST_T;
```

Все, что касается списка параметров, подчинено тем же правилам, что и для хранимых процедур и функций. Методы можно вызывать с использованием как именного, так и позиционного представления, а их параметры могут иметь значения по умолчанию. Методы могут быть перегружаемыми. Можно переопределять тип и число аргументов метода. Если в момент выполнения метода возникает исключение, то процедура завершает свою работу, причем выходным параметрам типа OUT и IN OUT ничего не присваивается; если же были выполнены какие-то изменения в SELF, то они также ликвидируются.

Ключевое слово SELF. Рассмотрим метод CHANGE_PRICE:

```
MEMBER PROCEDURE CHANGE_PRICE
(NEW_PRICE  IN  NUMBER)  IS
BEGIN
    PRICE := NEW_PRICE;
END  CHANGE_PRICE;
```

Этот метод вызывается для модификации атрибута PRICE типа BOOK_T. Для автоматической привязки идентификатора PRICE к экземпляру объекта внутри метода применяется ключевое слово SELF, т. е. в данном случае вместо оператора

```
PRICE := NEW_PRICE;
```

можно было написать оператор

```
SELF.PRICE := NEW_PRICE;
```

При передаче же текущего экземпляра объекта другой процедуре или функции в качестве аргумента указывать SELF необходимо. Приведем пример.

```
MEMBER PROCEDURE CHANGE_PRICE_ANOTHER
(BOOK  BOOK_T)  IS
```

```
BEGIN
    SELF.PRICE := BOOK.PRICE;
END   CHANGE_PRICE_ANOTHER;
```

Размещение объектов в базе данных. В PL/SQL имеется возможность для создания объектов двух типов: устойчивых и неустойчивых, что определяет их свойства, а также операции, которые разрешено выполнять над ними. ***Устойчивым*** (persistent) объектом называется объект, хранящийся в базе данных, а ***неустойчивым*** (transient) – объект, объявленный локально в блоке PL/SQL. С завершением работы блока PL/SQL неустойчивый объект разрушается. Устойчивый объект остается доступным до тех пор, пока не будет удален явным образом. Приведем пример неустойчивого объекта BOOK:

```
DECLARE
  FORM1  VARCHAR2(40);
  -- Создание объекта.
  BOOK  BOOK_T := BOOK_T (10, 'Война и мир', 'Толстой Л.', 5000);
BEGIN
  BOOK.CHANGE_PRICE(7000);
  FORM1 := BOOK.FORMATTED_NAME;
  -- Занесение значений атрибутов TITLE, AYTHOR, PRICE
  -- в таблицу TEST_T;
  INSERT INTO TEST_T VALUES(FORM1, BOOK.PRICE);
END;
```

По завершении работы блока объект BOOK будет разрушен. Сохранится лишь информация, занесенная в таблицу TEST_T.

Устойчивые объекты хранятся в таблицах базы данных, как и стандартные скалярные типы. Существует два различных способа хранения объектов в таблице: в качестве объекта-строки или объекта-столбца.

Объект-строка (row object) занимает целую строку таблицы базы данных, причем в строке не должно содержаться других полей. Таблица, состоящая из таких строк, называется объектной (object table) и создается при помощи следующего оператора:

```
CREATE TABLE имя_таблицы OF объектный_тип;
```

где имя_таблицы – это имя создаваемой таблицы, а объектный_тип – тип объекта-строки. В качестве примера создадим объектную таблицу BOOK_S на базе объектного типа BOOK_T:

```
CREATE   TABLE   BOOK_S  OF   BOOK_T;
```

В каждой строке таблицы BOOK_S находится экземпляр типа BOOK_T, являющийся строкой, атрибуты которой определены в объектном типе BOOK_T. Следовательно, в эту таблицу можно вводить только объекты. Ниже приведен ряд примеров ввода объектов в таблицу BOOK_S. Как указывалось выше вставка новых значений в таблицу осуществляется через функцию-конструктор.

```
INSERT INTO BOOK_S
VALUES (BOOK_T(10, 'Казаки', 'Толстой Л.',5000));
INSERT INTO BOOK_S
VALUES (BOOK_T(20, 'Война и мир', 'Толстой Л.',15000));
```

Объектные таблицы очень похожи на обычные реляционные таблицы. Поэтому при заполнении объектной таблицы в целях сопряжения с обычными реляционными таблицами, которые были до этого в БД, допускается и обычная вставка при условии, что в списке атрибутов нет других встроенных объектных типов. К примеру, можно ввести данные в таблицу BOOK_S с помощью следующего оператора INSERT:

```
INSERT INTO BOOK_S
VALUES(30, 'Дюна', 'Герберт Ф.', 8000);
```

Во время такой модернизации различные реляционные таблицы можно создать заново как объектные, причем существующие приложения изменять вовсе не обязательно.

Операции DML, выполняемые над таблицами, в которых содержатся объекты-строки, абсолютно идентичны реляционным операциям DML.

В операторах UPDATE и DELETE можно указывать объекты в качестве переменных привязки. В этом случае операции DELETE и UPDATE можно выполнять, как над обычными реляционными таблицами. Покажем это на примерах:

```
UPDATE BOOK_S SET TITLE = ’Дети Дюны’ WHERE ID = 30;
DELETE FROM BOOK_S WHERE ID = 30;
```

В запросе объект-строка может вести себя по-разному. Если просто написать оператор вывода информации, перечислив все нужные атрибуты, то система будет рассматривать их как обычный набор атрибутов, а не объект, поскольку объектная таблица описывается точно так же, как и реляционная. Поэтому для выбора объекта нужно использовать операцию VALUE.

Операция VALUE (значение) возвращает объект. В качестве аргумента VALUE используется переменная корреляции, которая в данном кон-

тексте представляет собой псевдоним таблицы. Использование VALUE и обычного варианта запроса иллюстрируется в следующем примере:

```
DECLARE
    CURRENT_ID          BOOK_S.ID%TYPE;
    CURRENT_TITLE       BOOK_S.TITLE%TYPE;
    CURRENT_AUTHOR   BOOK_S.AUTHOR%TYPE;
    CURRENT_PRICE       BOOK_S.PRICE%TYPE;
    CURRENT_BOOK        BOOK_T;
    BEGIN
    SELECT *  INTO CURRENT_ID, CURRENT_TITLE,
    CURRENT_AUTHOR, CURRENT_PRICE  FROM BOOK_S
    WHERE ID = 10;
    SELECT VALUE(B)  INTO CURRENT_BOOK  FROM
    BOOK_S  B  WHERE   ID   =   10;
 END;
```

При формировании объектной таблицы, где объекты – строки, каждому объекту присваивается идентификатор. ***Идентификатор объекта*** — это уникальный указатель на устойчивый объект конкретного типа, который однозначно определяет объект. Идентификаторы объектов уникальны по всему пространству БД Oracle: два объекта не могут иметь один и тот же идентификатор. Если даже объект удаляется, этот номер никакому другому объекту никогда не присваивается. Идентификатор объекта – это внутрисистемная структура; общее число идентификаторов составляет 2^{128} различных значений.

Идентификаторы объекта имеются только у объектов-строк и строк объектных представлений. Если объект имеет идентификатор, на этот объект можно ссылаться. Ссылка на объект описывается в разделе объявлений или указывается в описании таблицы и выглядит следующим образом:

имя_переменной REF объектный_тип,

где имя_переменной – это имя ссылки на объект, а объектный_тип – определенный ранее объектный тип.

Ссылки на объекты можно указывать в блоках PL/SQL и в SQL-операторах, используя при этом операции VALUE и REF.

Операции REF и DEREF. Результатом выполнения операции REF(аргумент) является ссылка на запрашиваемый объект. Как и для VALUE, аргумент REF – это переменная корреляции. Приведем пример, в котором выбирается ссылка на строку таблицы с конкретным значением атрибута ID:

```
DECLARE
   CURRENT_BOOK_REF  REF  BOOK_T;
BEGIN
   SELECT  REF(B)   INTO CURRENT_BOOK_REF
   FROM BOOK_S  B  WHERE    ID   =   10;
END;
```

Операция DEREF(ссылка) возвращает исходный объект по заданной на него ссылке. Продолжим предыдущий пример, в котором обновим значение атрибута TITLE у объекта, выбранного по ссылке:

```
DECLARE
   CURRENT_BOOK_REF  REF  BOOK_T;
   CURRENT_BOOK   BOOK_T;
BEGIN
   SELECT  REF(B)   INTO CURRENT_BOOK_REF
   FROM BOOK_S  B  WHERE    ID   =   10;
    SELECT    DEREF(CURRENT_BOOK_REF)
    INTO   CURRENT_BOOK   FROM   DUAL;
    CURRENT_BOOK.TITLE  :=  'Анна Каренина';
END;
```

Если объект, на который указывает REF, удален, то ссылку называют ***висячей*** (dangling), поскольку теперь она указывает на несуществующий объект. Использовать операцию DEREF для висячей ссылки нельзя. Чтобы проверить, не является ли ссылка висячей, применяется предикат IS DANGLING. Все рассмотренные операции (VALUE, REF, DEREF и IS DANGLING) можно использовать только в SQL-операторах. В процедурных операторах это запрещено.

В операторах INSERT и UPDATE используется конструкция – RETURNING. Она применяется для считывания информации из вновь введенной или обновленной строки; при этом формировать дополнительный запрос не требуется. Синтаксис конструкции RETURNING таков:

RETURNING список_выбора INTO список_ввода;

где список_выбора аналогичен списку выбора запроса, а список_ввода – это то же самое, что и оборот INTO запроса. Например, если в объектную таблицу вводится некоторый объект, то можно вернуть ссылку на вновь вводимый объект следующим образом:

```
DECLARE
   CURRENT_BOOK_REF  REF  BOOK_T;
BEGIN
```

```
    INSERT INTO BOOK_S  B VALUES
    (BOOK_T(40, 'Игрок', 'Достоевский Ф.',  3000))
    RETURNING REF(B)  INTO  CURRENT_BOOK_REF  ;
END;
```

Объект-столбец (column object) занимает ровно один столбец таблицы. Для создания таблицы, содержащей объект-столбец, нужно просто указать объектный тип в качестве типа столбца в операторе CREATE TABLE. Допускается в такой таблице наличие и скалярных типов. Для примера создадим таблицу BOOK_DELIVERY на базе описанного выше типа BOOK_T:

```
CREATE  TABLE  BOOK_DELIVERY
(SALESMAN  VARCHAR2(15),
QUANTITY  NUMBER(7),
BOOK  BOOK_T);
```

Таблица BOOK_DELIVERY состоит из трех столбцов, один из которых является объектным типом. В данном случае ввести информацию в таблицу без конструктора нельзя, т. е. для ввода новых значений необходимо использовать следующую конструкцию:

```
INSERT INTO BOOK_DELIVERY
VALUES ('Иванов И. И.', 5, BOOK_T(10, 'Казаки', 'Толстой Л.', 5000));
```

В противном случае будет выдана ошибка.

Информацию из таблицы можно выбрать обычным способом с помощью оператора SELECT:

```
SELECT * FROM  BOOK_DELIVERY;
```

На экран будет выведено:

```
Иванов И. И.   5   BOOK_T(10, 'Казаки', 'Толстой Л.', 5000));
```

То есть столбец BOOK_T рассматривается как единое целое, обратиться непосредственно к какому-то отдельному элементу нельзя.

Кроме того, можно сослаться на объект-столбец в условии WHERE, указав перечень значений атрибутов этого объекта. Ниже приведен пример использования оператора SELECT.

```
SELECT SALESMAN FROM BOOK_DELIVERY
WHERE BOOK = BOOK_T(10, 'Казаки', 'Толстой Л.', 5000);
```

То же самое касается операторов DELETE, UPDATE.

Отметим, что в блоке PL/SQL для выполнения данного оператора для таблицы необходимо указать псевдоним.

```
DECLARE
   CURRENT_BOOK  BOOK_T;
BEGIN
    SELECT  BOOK  INTO CURRENT_BOOK
    FROM BOOK_DELIVERY  D
    WHERE  D.BOOK.ID = 10;
END;
```

Иногда на практике такие таблицы, в которых имеются объекты-столбцы, учитывая, что существует таблица, где эти строки присутствуют, строят следующим образом. В столбце указывается не объектный тип, а ссылка на этот тип.

```
CREATE  TABLE  BOOK_DELIVERY_M
(SALESMAN  VARCHAR2(15),
QUANTITY  NUMBER(7),
BOOK  REF  BOOK_T);
```

Для вставки информации в такую таблицу используют следующую конструкцию:

```
INSERT INTO BOOK_DELIVERY_M
SELECT 'Иванов И. И.', 5, REF(B)
FROM BOOK_S  B
WHERE ID = 10;
```

Строка с ID = 10 имеет уникальный идентификатор, который извлекается с помощью операции REF(B) и размещается в столбце. В данном случае также необходимо использование синонима. Для просмотра информации таблицы можно использовать следующий оператор:

```
SELECT * FROM BOOK_DELIVERY_M;
```

Но в этом случае на экран выведется вместо третьего атрибута очень длинное число – идентификатор объекта. Поэтому для выдачи информации лучше использовать следующий вариант:

```
SELECT SALESMAN,  QUANTITY  ,
DEREF(BOOK)  FROM  BOOK_DELIVERY_M;
```

В результате получим следующую строку:

Иванов И. И.   5   BOOK_T(10, 'Казаки', 'Толстой Л.',5000));

В данном случае операция DEREF возвращает объект по ссылке. Если необходимо изменить информацию в таблице, то оператор UPDATE записывают в следующем виде:

```
UPDATE BOOK_DELIVERY_M  SET SALESMAN = 'Петров П. П.'
WHERE  DEREF(BOOK) = BOOK_T(10, 'Казаки', 'Толстой Л.', 5000);
```

Методы MAP и ORDER. Методы MAP и ORDER – специальные методы, предназначенные для реализации упорядочения создаваемых на базе объектного типа реальных экземпляров объекта. При объявлении объектного типа можно использовать только один из этих методов. Когда создается объект, который является структурой, то необходимо определить порядок проверки условий, и именно MAP и ORDER устанавливают этот порядок. Если один из этих методов присутствует, то можно осуществить сравнение объектов не только в операторах SQL, но и в операторах PL/SQL. Если их нет, то объекты можно проверять только на эквивалентность и только в операторах SQL. Методы MAP и ORDER дают возможность упорядочивать объекты, а также сравнивать их в процедурных операторах. Вызов любого из этих методов происходит автоматически.

Метод MAP всегда представляется функцией, которая не имеет параметров, и типом возвращаемого значения которой могут быть только DATE, NUMBER, VARCHAR2. На основании анализа данного объекта эта функция возвращает значение указанного типа, которое может быть использовано для упорядочивания объектов.

Ниже приведен пример использования метода MAP для объектного типа BOOK_TM, аналогичного по структуре объектному типу BOOK_T. Создадим заголовок объектного типа BOOK_TM.

```
CREATE  OR  REPLACE  TYPE  BOOK_TM  AS  OBJECT
( ID   NUMBER(2),
  TITLE VARCHAR2(15),
  AUTHOR VARCHAR2(12),
   PRICE  NUMBER(7),
  MAP  MEMBER  FUNCTION  RETURN_TITLE
  RETURN  VARCHAR2
);
```

Создадим тело объектного типа BOOK_TM.

```
CREATE  OR  REPLACE  TYPE  BODY  BOOK_TM  AS
MAP  MEMBER  FUNCTION  RETURN_TITLE
RETURN  VARCHAR2  IS
BEGIN
   RETURN  TITLE;
END  RETURN_TITLE;
END;
```

Функция RETURN_TITLE возвращает название книги. Создадим объектную таблицу BOOK_K на базе объектного типа BOOK_TM.

```
CREATE TABLE BOOK_K OF BOOK_TM;
```

Внесем в таблицу BOOK_K следующие записи с помощью оператора INSERT:

```
INSERT INTO BOOK_K VALUES
(BOOK_TM(10, 'Казаки', 'Толстой Л.', 5000));
INSERT INTO BOOK_K VALUES
(BOOK_TM(20, 'Война и мир', 'Толстой Л.', 15000));
INSERT INTO BOOK_K VALUES (30, 'Дюна', 'Герберт Ф.', 8000);
```

После создания этой таблицы для упорядочения выводимых записей по названию книг можно выполнить, например, такой оператор SELECT:

```
SELECT VALUE(B) FROM BOOK_K B ORDER BY TITLE;
```

В результате чего будет выведена следующая информация:

```
VALUE(B) (ID, TITLE, AUTHOR, PRICE)
-----------------------------------------------------------
BOOK_TM(20, 'Война и мир', 'Толстой Л.', 15000)
BOOK_TM(30, 'Дюна', 'Герберт Ф.', 8000)
BOOK_TM(10, 'Казаки', 'Толстой Л.', 5000)
```

Метод ORDER всегда имеет два параметра: первый является встроенным (так называемый SELF), а второй параметр (объектного типа) передается в эту функцию. Метод ORDER возвращает следующие значения, по которым можно судить о соотношении этих двух переданных функции объектов:

- -1, если параметр больше SELF;
- 1, если параметр меньше SELF;
- 0, если параметр равен SELF.

Метод ORDER используется аналогично методу MAP. Создадим метод ORDER для объекта BOOK_TM; этот метод упорядочивает сведения о книгах по названиям книг. Пересоздадим заголовок объектного типа BOOK_TM.

```
CREATE OR REPLACE TYPE BOOK_TM AS OBJECT
( ID NUMBER(2),
  TITLE VARCHAR2(15),
  AUTHOR VARCHAR2(12),
   PRICE NUMBER(7),
  ORDER MEMBER FUNCTION COMPARE_BOOKS
```

```
  (CUR_BOOK  IN  BOOK_TM)
   RETURN  NUMBER
);
```

Пересоздадим тело объектного типа BOOK_TM.

```
CREATE  OR  REPLACE  TYPE  BODY  BOOK_TM  AS
ORDER  MEMBER  FUNCTION  COMPARE_BOOKS
(CUR_BOOK  IN  BOOK_TM)
RETURN  NUMBER  IS
BEGIN
    IF CUR_BOOK.TITLE=SELF.TITLE THEN
      RETURN 0;
    ELSIF CUR_BOOK.TITLE>SELF.TITLE THEN
   RETURN  -1;
  ELSE
    RETURN  1;
END IF;
END  COMPARE_BOOKS;
END;
```

После создания метода ORDER выполним следующий оператор:

```
   SELECT   VALUE(B)  FROM  BOOK_K  B ORDER BY TITLE;
```

В результате чего будет выведена следующая информация:

```
VALUE(B)(ID, TITLE,   AUTHOR,   PRICE)
-----------------------------------------------------------------
BOOK_TM(20, 'Война и мир', 'Толстой Л.', 15000)
BOOK_TM(30, 'Дюна', 'Герберт Ф.', 8000)
BOOK_TM(10, 'Казаки', 'Толстой Л.', 5000)
```

Изменение и удаление типов. Для изменения объектного типа используется оператор ALTER TYPE, который можно использовать для перекомпиляции описания или тела объектного типа, либо для добавления в тип новых методов.

Команда ALTER TYPE имеет следующий синтаксис:

```
   ALTER TYPE имя_типа COMPILE [SPECIFICATION | BODY];
```

где имя_типа – имя изменяемого типа. С помощью этой команды можно перекомпилировать спецификацию или тело типа, которые в скомпилированном виде хранятся в словаре данных. Если не указывается ни SPECIFICATION, ни BODY, то перекомпилироваться будут как

описание, так и тело типа. Например, следующая команда вызывает перекомпиляцию тела типа BOOK_T:

```
ALTER TYPE BOOK_T COMPILE BODY;
```

Перекомпиляция объекта может потребоваться в случае, когда объект в результате изменения связанных с данным объектом других объектов.

Другая форма команды ALTER TYPE используется для добавления к типу новых методов. Она имеет следующий синтаксис:

```
ALTER TYPE имя_типа REPLACE AS OBLECT
(спецификация_объектного_типа);
```

где имя_типа – это имя объектного типа, а спецификация_объектного_типа – полное описание типа, определенное командой CREATE TYPE. Новое описание должно во всем, кроме дополнительных методов, совпадать с исходным. Должны быть указаны первоначальные атрибуты и типы. Если тело типа уже существует, оно становится недостоверным, поскольку в нем не описаны новые методы.

Использование команды ALTER TYPE ... REPLACE AS OBLECT иллюстрируется на следующем примере. Запросим статус объекта BOOK_T.

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'BOOK_T';
```

В результате получим сообщение о достоверности заголовка и тела объекта BOOK_T.

```
OBLECT_NAME    OBJECT_TYPE    STATUS
-------------------------------------------------------------

BOOK_T              TYPE              VALID
BOOK_T              TYPE BODY      VALID
```

Изменим тип, добавив к нему новый метод. При этом тело типа становится недостоверным.

```
ALTER  TYPE BOOK_T  REPLACE AS OBJECT
(  ID  NUMBER(2),              -- код книги
   TITLE  VARCHAR2(15),     -- название книги
   AUTHOR  VARCHAR2(15), -- фамилия автора
   PRICE  NUMBER(7),   -- цена книги
--Возвращает фамилию автора и название книги.
   MEMBER FUNCTION FORMATTED_NAME
```

```
RETURN VARCHAR2,
--Обновляет поле PRICE значением, указанным в поле NEW_PRICE.
  MEMBER  PROCEDURE  CHANGE_PRICE
  (NEW_PRICE  IN  NUMBER),
--Возвращает код книги
MEMBER FUNCTION BOOK_ID  RETURN NUMBER
);
```

Запросим еще раз статус объекта BOOK_T.

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'BOOK_T';
```

В результате получим сообщение о недостоверности тела объекта BOOK_T.

```
OBLECT_NAME    OBJECT_TYPE     STATUS
---------------------------------------------------------------

BOOK_T                TYPE              VALID
BOOK_T                 TYPE BODY       INVALID
```

Пересоздадим тело типа BOOK_T с добавленным новым методом.

```
CREATE  OR  REPLACE  TYPE  BODY  BOOK_T  AS
MEMBER  FUNCTION  FORMATTED_NAME
RETURN  VARCHAR2  IS
BEGIN
   RETURN  SELF.AUTHOR||SELF.TITLE;
END  FORMATTED_NAME;
MEMBER PROCEDURE CHANGE_PRICE
(NEW_PRICE  IN  NUMBER)  IS
BEGIN
   PRICE := NEW_PRICE;
END  CHANGE_PRICE;
MEMBER  FUNCTION  BOOK_ID  RETURN  NUMBER  IS
BEGIN
   RETURN ID;
END BOOK_ID;
END;
```

Запросим статус объекта BOOK_T.

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
```

```
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'BOOK_T';
```

В результате получим сообщение о достоверности тела объекта BOOK_T.

```
OBLECT_NAME     OBJECT_TYPE    STATUS
-------------------------------------------------------------

BOOK_T                  TYPE                  VALID
BOOK_T                   TYPE BODY        VALID
```

Команда DROP TYPE используется для удаления объектного типа и имеет следующий синтаксис:

DROP TYPE имя_типа [FORCE];

При этом если используется опция FORCE, то объектный тип удаляется принудительно, несмотря на имеющиеся связи, и делая при этом недействительными все зависимое от него объекты. Если данный параметр отсутствует, то объектный тип будет удален только в том случае, если на него нет ссылок, т. е. при отсутствии в схеме других объектов, зависящих от данного объектного типа.

Команда

DROP TYPE BODY имя_типа

удаляет только тело объектного типа, не трогая спецификацию типа и все зависимые от него объекты.

Литература

Основная

1. Дейт К. Введение в системы баз данных, 7-е издание – М.: Вильямс, 2001. – 1072с.
2. Конолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е издание. – М.: Вильямс, 2000. – 1120с.
3. Карпова Т.С. Базы данных. Модели, разработка, реализация. – С-Пб.: Питер, 2001. – 304с.
4. Хансен Г., Хансен Дж. Базы данных: разработка и управление. – М.: Бином, 1999. – 504с.
5. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений / Под ред. проф. А.Д. Хомоненко. – СПб.: КОРОНА принт, 2000. – 416 с.
6. Бэлтон Д., Гокмен М., Ингрем Дж. Внутренний мир Oracle8. Проектирование и настройка: Пер. с англ. – К.: ДиаСофт, 2000. – 800 с.
7. Корнеев В.В., Гарев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. – М.: Нолидж, 2000. – 352с.
8. Грофф Д.Р. Вайнберг П.Н. SQL: полное руководство. – К.: BHV, 1999. – 608с.
9. Маклаков С.В. BPwin, ERwin. CASE-средства разработки информационных систем. – М.: Диалог-МИФИ, 2000. – 256с.
10. Дунаев С.П. Доступ к базам данных и техника работы в сети. – М.: Диалог-МИФИ, 1999. – 416 с.

Дополнительная

1. Пэйдж В. Дж. Использование Oracle 8/8i – М.: Вильямс, 2000. – 1024с.
2. Oracle 8. Энциклопедия пользователя. – К.: Диасофт, 1999. – 864с.
3. Урман Л. Oracle 8. Программирование на языке PL/SQL. – К.: Лори, 1999. – 608с.
4. Баженова И.Ю. Oracle 8/8i. Уроки программирования. – М.: Диалог-Мифи, 2000. – 304с.
5. Вейскас Д. Эффективная работа с Microsoft Access 2000. – С-Пб.: Питер, 2000. – 1040с.
6. Архангельский А.Я. Программирование в Delphi 7. – М.: Бином, 2003. – 1152с.

Содержание.