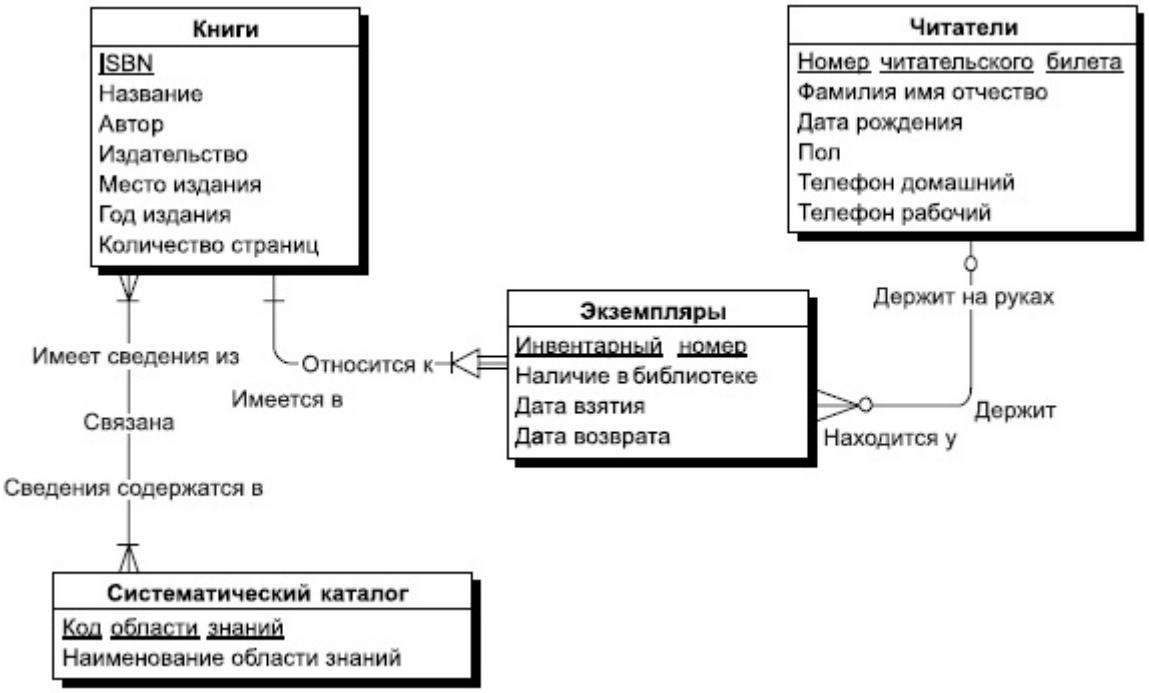
4. Поддержка целостности в реляционной модели данных

Оглавление

Одним из основополагающих понятий в технологии баз данных является понятие *целостности*. В общем случае это понятие прежде всего связано с тем, что *база данных* отражает в информационном виде некоторый *объект* реального мира или совокупность взаимосвязанных объектов реального мира. В реляционной модели объекты реального мира представлены в виде совокупности взаимосвязанных отношений. Под целостностью будем понимать соответствие *информационной модели предметной области*, хранимой в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени. Любое изменение в *предметной области*, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная *интерпретация* информационной модели в терминах *предметной области*.

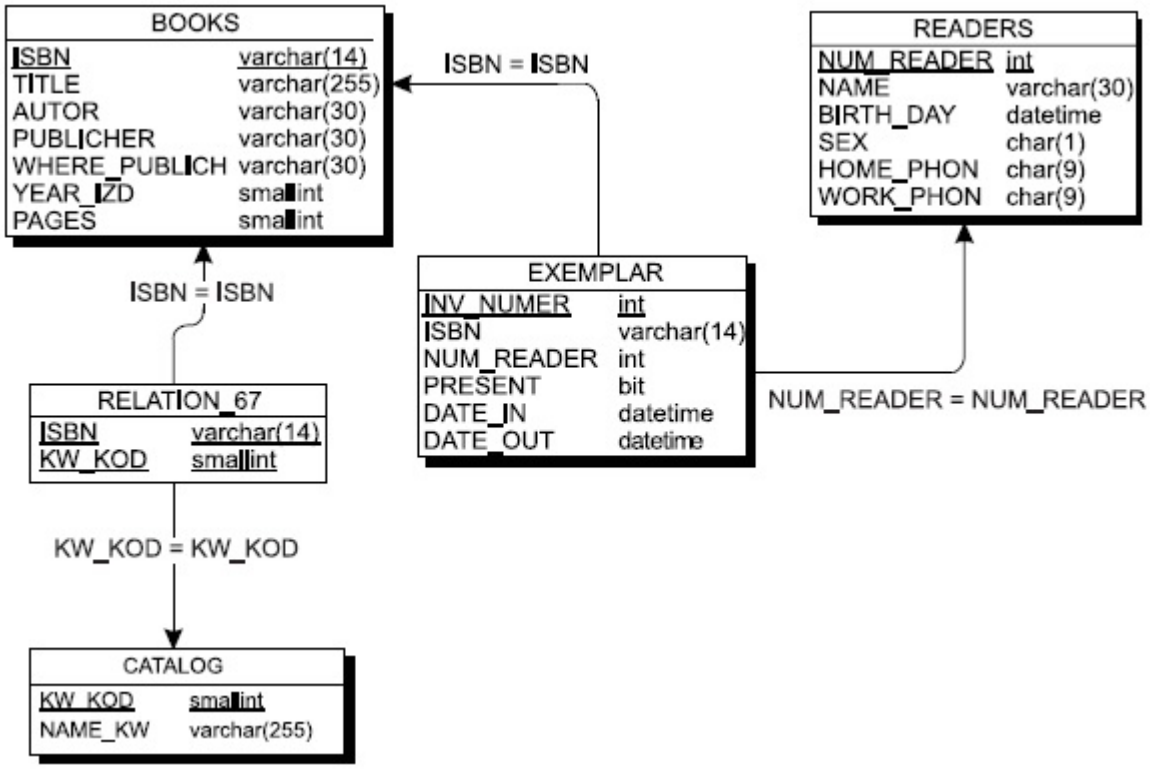
Инфологическая *модель предметной области* "Библиотека" представлена на следующем рисунке



Инфологическая модель "Библиотека" разработана нами под те задачи, которые были перечислены ранее. В этих задачах мы не ставили условие хранения истории чтения книги, например, с целью поиска того, кто раньше держал книгу и мог нанести ей вред или забыть в ней случайно большую сумму денег. Если бы мы ставили перед собой задачу хранения и этой информации, то наша инфологическая модель была бы другой.

Разрешение связей типа "*многие-ко-многим*". Так как в реляционной модели данных поддерживаются между отношениями только связи типа "*один-ко-многим*", а в ER-

модели допустимы связи "*многие-ко-многим*", то необходим специальный механизм преобразования, который позволит отразить множественные связи, неспецифические для реляционной модели, с помощью допустимых для нее категорий. Это делается введением специального дополнительного связующего отношения, которое связано с каждым исходным связью "*один-ко-многим*", атрибутами этого отношения являются первичные ключи связываемых отношений. Так, например, в схеме "Библиотека" присутствует *связь* такого типа между сущностью "Книги" и "Системный каталог". Для разрешения этой неспецифической связи при переходе к реляционной модели должно быть введено специальное дополнительное *отношение*, которое имеет всего два атрибута: ISBN (*шифр* книги) и KOD (код области знаний). При этом каждый из атрибутов нового отношения является внешним ключом (FOREIGN KEY), а вместе они образуют *первичный ключ* (PRIMARY KEY) новой связующей сущности. На рисунке ниже представлена реляционная (физическая) модель, соответствующая представленной ранее инфологической модели "Библиотека".



Мы отметили, что только существенные или значимые изменения *предметной области* должны отслеживаться в информационной модели. Действительно, модель всегда представляет собой некоторое упрощение реального объекта, в модели мы отражаем только то, что нам важно для решения конкретного набора задач. Именно поэтому в информационной системе "Библиотека" мы, например, не отразили *место* хранения конкретных экземпляров книг, потому что мы не ставили задачу автоматической адресации библиотечных стеллажей. И в этом случае любое перемещение книг с одного места на другое не будет отражено в модели, это перемещение несущественно для наших задач. С другой стороны, процесс взятия книги читателем или возврат любой книги в библиотеку для нас важен, и мы должны его отслеживать в соответствии с изменениями в реальной *предметной области*. И с этой точки зрения наличие у экземпляра книги

указателя на его отсутствие в библиотеке и одновременное отсутствие записи о конкретном номере читательского билета, за которым числится этот экземпляр книги, является противоречием, такого быть не должно. И в модели данных должны быть предусмотрены средства и методы, которые позволят нам обеспечивать динамическое отслеживание в базе данных согласованных действий, связанных с согласованным изменением информации. Именно этим вопросам и посвящена данная лекция.

4.1. Общие понятия и определения целостности

Поддержка целостности в реляционной модели данных в ее классическом понимании включает в себя 3 аспекта.

Во-первых, это *поддержка структурной целостности*, которая трактуется как то, что реляционная *СУБД* должна допускать работу только с *однородными структурами* данных типа "реляционное *отношение*". При этом понятие "реляционного отношения" должно удовлетворять всем ограничениям, накладываемым на него в классической теории реляционной *БД* (отсутствие дубликатов кортежей, соответственно, обязательное наличие первичного ключа, отсутствие понятия упорядоченности кортежей).

В *дополнение* к структурной целостности необходимо рассмотреть проблему неопределенных *Null* значений. Как уже указывалось раньше, неопределенное *значение* интерпретируется в реляционной модели как *значение*, неизвестное на данный момент времени. Это *значение* при появлении дополнительной информации в любой момент времени может быть заменено на некоторое конкретное *значение*. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное *значение* никогда не считается равным другому неопределенному значению. Для выявления равенства значения некоторого атрибута неопределенному применяют специальные стандартные предикаты:

<имя атрибута>IS NULL и <имя атрибута> IS NOT NULL.

Если в данном кортеже (в данной строке) указанный *атрибут* имеет неопределенное *значение*, то *предикат* IS NULL принимает *значение* TRUE (*Истина*), а *предикат* IS NOT NULL — FALSE (*Ложь*), в противном случае *предикат* IS NULL принимает *значение* FALSE, а *предикат* IS NOT NULL принимает *значение*TRUE.

Ведение *Null* значений вызвало необходимость модификации классической двузначной логики и превращения ее в трехзначную. Все *логические операции*, производимые с неопределенными значениями, подчиняются этой логике в соответствии с заданной таблицей истинности.

Таблица 11.1. Таблица истинности для логических операций с неопределенными значениями

A	**B**	**Not A**	**A & B**	$A \vee B$
TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE
TRUE	Null	FALSE	Null	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	Null	TRUE	FALSE	Null
Null	TRUE	Null	Null	TRUE
Null	FALSE	Null	FALSE	Null

Null	Null	Null	Null	Null

Во-вторых, это *поддержка языковой целостности*, которая состоит в том, что реляционная *СУБД* должна обеспечивать языки описания и манипулирования данными не ниже стандарта *SQL*. Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

Именно поэтому *доступ* к информации, хранимой в базе данных, и любые изменения этой информации могут быть выполнены только с использованием операторов языка *SQL*.

В-третьих, это *поддержка ссылочной целостности* (Declarative *Referential Integrity*, DRI), что означает обеспечение одного из заданных принципов взаимосвязи между экземплярами кортежей взаимосвязанных отношений:

- кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ними.
- кортежи основного отношения модифицируются при удалении кортежа основного отношения, связанного с ними, при этом на месте ключа родительского отношения ставится неопределенное Null значение.

Ссылочная *целостность* обеспечивает поддержку непротиворечивого состояния *БД* в процессе модификации данных при выполнении операций добавления или удаления.

Кроме указанных ограничений целостности, которые в общем виде не определяют семантику *БД*, вводится понятие *семантической поддержки целостности*.

Структурная, языковая и ссылочная *целостность* определяют правила работы *СУБД* с реляционными структурами данных. Требования поддержки этих трех видов целостности говорят о том, что каждая *СУБД* должна уметь это делать, а разработчики должны это учитывать при построении баз данных с использованием реляционной модели. И эти требования поддержки целостности достаточно абстрактны, они определяют допустимую форму представления и обработки информации в реляционных базах данных. Но с другой стороны, эти аспекты никак не касаются содержания *базы данных*. Для определения некоторых ограничений, которые связаны с содержанием *базы данных*, требуются другие методы. Именно эти методы и сведены в поддержку *семантической целостности*.

Давайте рассмотрим конкретный пример. То, что мы можем построить схему *базы данных* или ее концептуальную модель только из совокупности нормализованных таблиц, определяет структурную *целостность*. И мы построили нашу схему библиотеки из пяти взаимосвязанных отношений. Но мы не можем с помощью перечисленных трех методов поддержки целостности обеспечить ряд правил, которые определены в нашей *предметной области* и должны в ней соблюдаться. К таким правилам могут быть отнесены следующие:

1. В библиотеке должны быть записаны читатели не моложе 17 лет.
2. В библиотеке присутствуют книги, изданные начиная с 1960 по текущий год.
3. Каждый читатель может держать на руках не более 5 книг.

4. Каждый читатель при регистрации в библиотеке должен дать телефон для связи: он может быть рабочим или домашним.

Принципы семантической поддержки целостности как раз и позволяют обеспечить автоматическое выполнение тех условий, которые перечислены ранее.

Семантическая *поддержка* может быть обеспечена двумя путями: декларативным и процедурным путем. Декларативный *путь* связан с наличием механизмов в рамках *СУБД*, обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений, называемых чаще всего "бизнес-правилами" (*Business Rules*) или декларативными ограничениями целостности.

Выделяются следующие виды декларативных ограничений целостности:

- Ограничения целостности атрибута: значение по умолчанию, задание обязательности или необязательности значений (Null), задание условий на значения атрибутов.

Задание значения по умолчанию означает, что каждый раз при вводе новой строки в отношение, при отсутствии данных в указанном столбце этому атрибуту присваивается именно значение по умолчанию. Например, при вводе новых книг разумно в качестве значения по умолчанию для года издания задать значение текущего года. Например, для MS Access это выражение будет иметь вид: **YEAR(GETDATE() ())**

Здесь **GETDATE()** () — функция, возвращающая значение текущей даты, YEAR(data) — функция, возвращающая значение года указанной в качестве параметра даты.

В качестве условия на значение для года издания надо задать выражение, которое будет истинным только тогда, когда год издания будет лежать в пределах от 1960 года до текущего года. В конкретных СУБД это значение будет формироваться с использованием специальных встроенных *функций СУБД*.

Для MS SQL Server это выражение будет выглядеть следующим образом:

YEAR_PUBL Between 1960 ANDYEAR(GETDATE())

Здесь GETDATE() — функция MS SQL Server, возвращающая значение текущей даты, YEAR_PUBL — имя столбца, соответствующего году издания.

Ограничения целостности, задаваемые на уровне доменов, при поддержке доменной структуры. Эти ограничения удобны, если в базе данных присутствуют несколько столбцов разных отношений, которые принимают значения из одного и того же множества допустимых значений. Некоторые СУБД поддерживают подобную доменную структуру, то есть разрешают определять отдельно домены, задавать тип данных для каждого домена и задавать соответственно ограничения в виде бизнес-правил для доменов. А для атрибутов задается не примитивный первичный тип данных, а их принадлежность тому или другому домену. Иногда доменная структура выражена неявно и в ряде СУБД применяется специальная терминология для этого. Так, например, в MS SQL Server вместо понятия домена вводится понятие типа данных, определенных пользователем, но смысл этого типа данных фактически эквивалентен смыслу домена. В этом случае действительно удобно задать ограничение на значение прямо на уровне домена, тогда оно автоматически будет выполняться для всех атрибутов, которые принимают значения из этого домена. А почему удобно задать это ограничение на уровне домена? А если мы зададим это ограничение для каждого атрибута, входящего в домен, разве

наша система будет работать неправильно? Нет, конечно, она будет работать правильно, но представьте себе, что у вас в организации изменились правила работы, которые выражены в виде декларативных ограничений на значения. В нашем случае, например, мы будем комплектовать библиотеку более новыми книгами и теперь будем принимать в библиотеку книги, изданные не позднее 1980 года. А если это ограничение у нас задано не на один столбец, то нам надо просматривать все отношения и во всех отношениях менять старое правило на новое. Не легче ли заменить его один раз в домене, а все атрибуты, которые принимают значения из этого домена, будут автоматически работать по новому правилу.

Да, это действительно легче, тем более что в процессе работы схема базы данных разрастается и начинает содержать более сотни отношений, и задача нетривиальная — найти все отношения, в которых ранее установлено это ограничение и исправить его.

Одним из основных правил при разработке проекта базы данных, как мы уже упоминали раньше, является минимизация избыточности, а это означает, что если возможно информацию о чем-то, в том числе и об ограничениях, хранить в одном месте, то это надо делать обязательно.

- Ограничения целостности, задаваемые на уровне отношения. Некоторые семантические правила невозможно преобразовать в выражения, которые будут применимы только к одному столбцу. В нашем примере с библиотекой мы не сможем выразить требование наличия по крайней мере одного телефонного номера для быстрой связи с читателем. У нас под телефоны отведены два столбца, это в некотором роде искусственно, но специально так сделано, чтобы показать вам другой *тип ограничений*. Каждый из атрибутов является в общем случае необязательным и может принимать неопределенные значения: не обязательно должен быть задан как мобильный, так и домашний телефон. Мы хотим потребовать, чтобы из двух по крайней мере один телефон был бы задан обязательно. Попробуем сформулировать это в терминологии неопределенных значений баз данных. Домашний телефон должен быть задан (NOT NULL) или мобильный телефон должен быть задан (NOT NULL). Для MS Access или для MS SQL Server соответствующее выражение будет выглядеть следующим образом:

CELL_PHONE IS NOT NULL OR HOME_PHONE IS NOT NULL

- Ограничения целостности, задаваемые на уровне связи между отношениями: задание обязательности связи, принципов *каскадного удаления* и каскадного изменения данных, задание поддержки ограничений по *мощности связи*. Эти виды ограничений могут быть выражены заданием обязательности или необязательности значений внешних ключей во взаимосвязанных отношениях.

Декларативные ограничения целостности относятся к ограничениям, которые являются немедленно проверяемыми. Есть ограничения целостности, которые являются откладываемыми. Эти ограничения целостности поддерживаются механизмом транзакций и триггеров. Мы их рассмотрим в следующих лекциях.

4.2. Операторы DDL в языке SQL с заданием ограничений целостности

Декларативные ограничения целостности задаются на уровне операторов создания таблиц. В стандарте *SQL* оператор создания таблиц имеет следующий *синтаксис*:

<определение таблицы>::=CREATE TABLE <имя таблицы>

(<описание элемента таблицы> [{,<описание элемента таблицы>}...])

<описание элемента таблицы>::=<определение столбца>
<определение ограничений таблицы>
<определение столбца>::=<имя столбца> <тип данных>
[<значение по умолчанию>][<дополнительные ограничения столбца>...]
<значение по умолчанию>::=DEFAULT { <literal> | USER | NULL }
<дополнительные ограничения столбца>::=NOT NULL
[<ограничение уникальности столбца>]
<ограничение по ссылкам столбца>|
CHECK (<условия проверки на допустимость>)
<ограничение уникальности столбца>::= UNIQUE
<ограничение по ссылкам столбца>::=FOREIGN KEY <спецификация ссылки>
<спецификация ссылки>::= REFERENCES <имя основной таблицы>
(<имя первичного ключа основной таблицы>)

Давайте кратко прокомментируем оператор определения таблицы, *синтаксис* которого мы задали с помощью традиционной *формы Бэкуса—Наура*.

При описании таблицы задается *имя таблицы,* которое является идентификатором в базовом языке *СУБД* и должно соответствовать требованиям именования объектов в данном языке.

Кроме имени таблицы в операторе указывается *список* элементов таблицы, каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы. Требуется наличие хотя бы одного определения столбца. То есть таблицу, которая не имеет ни одного столбца, определить нельзя. Количество столбцов в одной таблице не ограничено, но в конкретных *СУБД* обычно бывают ограничения на количество атрибутов.

Оператор CREATE TABLE определяет так называемую базовую таблицу, то есть реальное *хранилище данных*.

Как видно, кроме обязательной части, в которой задается имя столбца и его *тип данных, определение* столбца может содержать два необязательных раздела: *значение* столбца *по* умолчанию и раздел дополнительных *ограничений целостности столбца*.

В разделе значения *по* умолчанию указывается *значение*, которое должно быть помещено в строку, заносимую в данную таблицу, если *значение* данного столбца явно не указано. В соответствии со стандартом языка *SQL значение по* умолчанию может быть указано в виде литеральной *константы* с типом, соответствующим типу столбца; путем задания ключевого слова USER, которому при выполнении оператора занесения строки соответствует *символьная строка,* содержащая имя текущего пользователя (в этом случае столбец должен иметь *тип символьных строк*); или путем задания ключевого слова NULL, означающего, что значением *по* умолчанию является неопределенное *значение*. Если *значение* столбца *по* умолчанию не специфицировано и в разделе *ограничений целостности столбца* указано NOT NULL (то есть наличие неопределенных значений запрещено), то попытка занести в таблицу строку с незаданным значением данного столбца приведет к ошибке.

Задание в разделе *ограничений целостности столбца* выражения NOT NULL приводит к неявному порождению *проверочного ограничения* целостности для всей таблицы "CHECK (C IS NOT NULL)" (где C — имя данного столбца). Если ограничение NOT NULL не указано и раздел умолчаний отсутствует, то неявно порождается раздел

умолчаний DEFAULT NULL. Если указана спецификация уникальности, то порождается соответствующая спецификация уникальности для таблицы.

При задании ограничений уникальности данный столбец определяется как возможный *ключ*, что предполагает уникальность каждого вводимого значения в данный столбец. И если это ограничение задано, то *СУБД* будет автоматически осуществлять проверку на отсутствие дубликатов значений данного столбца во всей таблице.

Если в разделе ограничений целостности указано ограничение *по* ссылкам данного столбца, то порождается соответствующее *определение* ограничения *по* ссылкам для таблицы: FOREIGN KEY(<имя столбца>) < спецификация ссылки>, что означает, что значения данного столбца должны быть взяты из соответствующего столбца *родительской таблицы*. *Родительской таблицей* в данном случае называется *таблица*, которая связана с данной таблицей связью "*один-ко-многим*" (1:M). При этом каждая строка *родительской таблицы* может быть связана с несколькими строками определяемой таблицы. *Трансляция* операторов *SQL* проводится в режиме интерпретации, поэтому важно, чтобы сначала была бы описана *родительская таблица*, а потом уже все подчиненные (*дочерние) таблицы*, связанные с ней. Иначе *транслятор* определит ссылку на неопределенный *объект*.

Наконец, если указано *проверочное ограничение* столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец, и неявно порождается соответствующее *проверочное ограничение* для всей таблицы. В *проверочных ограничениях*, накладываемых на столбец, нельзя задавать сравнение со значениями других столбцов данной таблицы.

Попробуем написать простейший оператор создания таблицы BOOKS из *базы данных* "Библиотека".

При этом будем предполагать наличие следующих ограничений целостности:

- Шифр книги — последовательность символов длиной не более 14, однозначно определяющая книгу, значит, это — фактически первичный ключ таблицы BOOKS.
- Название книги — последовательность символов, не более 120. Обязательно должно быть задано.
- Автор — последовательность символов, не более 30, может быть не задан.
- Соавтор — последовательность символов, не более 30, может быть не задан.
- Год издания — целое число, не менее 1960 и не более текущего года. По умолчанию ставится текущий год.
- Издательство — последовательность символов, не более 20, может отсутствовать.
- Количество страниц — целое число не менее 5 и не более 1000.

```
CREATE TABLE BOOKS
(
 ISBN varchar(14) NOT NULL PRIMARY KEY,
 TITLE varchar(120) NOT NULL,
 AUTOR varchar(30) NULL,
 COAUTOR varchar(30) NULL,
 YEAR_PUBL smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL between 1960
AND YEAR(GetDate())),
```

```
 PUBLICH varchar(20) NULL,
 PAGES smallint CHECK(PAGES > = 5 AND PAGES <= 1000)
      );
```

Почему мы не задали обязательность значения для количества страниц в книге? Потому что это является следствием *проверочного ограничения*, заданного на количество страниц, количество страниц всегда должно лежать в пределах от 5 до 1000, значит, оно не может быть незаданным и система это контролирует автоматически.

Теперь зададим описание таблицы "Читатели", которой соответствует *отношение* READERS:

- Номер читательского билета — это целое число в пределах 32 000 и он уникально определяет читателя.
- Имя, фамилия читателя — это последовательность символов, не более 30.
- Адрес — это последовательность символов, не более 50.
- Номера телефонов рабочего и домашнего — последовательность символов, не более 12.
- Дата рождения — календарная дата. В библиотеку принимаются читатели не младше 17 лет.

```
CREATE TABLE READERS
(
 READER_ID Smallint(4) PRIMARY KEY,
 FIRST_NAME char(30) NOT NULL,
 LAST_NAME char(30) NOT NULL,
 ADRES char(50),
 HOME_PHON char(12),
 WORK_PHON char(12),
 BIRTH_DAY date CHECK(DateDiff(year, BIRTH_DAY, GetDate()) >=17)
 );
```

Здесь DateDiff (часть даты, начальная дата, конечная дата) — *функция* MS *SQL Server*, которая определяет *разность* между начальной и конечной датами, заданную в единицах, определенных первым параметром — часть даты. Мы задали в качестве параметра Year, что значит, что мы *разность* определяем в годах.

Теперь зададим операцию создания таблицы *EXEMPLAR* (экземпляры книги). В этой таблице первичным ключом является *атрибут*, задающий инвентарный номер экземпляра книги. В такой постановке мы полагаем, что при поступлении книг в библиотеку им просто присваиваются соответствующие порядковые номера. Для того чтобы не утруждать библиотекаря все время помнить, какой номер был последним, мы можем воспользоваться тем, что некоторые *СУБД* допускают специальный инкрементный *тип данных*, то есть такой, значения которого автоматически увеличиваются или уменьшаются на заданную величину при каждом новом вводе данных. В *СУБД* MS Access такой *тип данных* называется "*счетчик*" (*counter*) и он всегда имеет начальное *значение* 1 и шаг, равный тоже 1, то есть при вводе каждого нового значения *счетчик* увеличивается на 1, значит, практически считает вновь введенные значения. В *СУБД* MS *SQL Server* это свойство IDENTITY, которое может быть присвоено ряду целочисленных типов данных. В отличие от "счетчика" свойство IDENTITY позволяет считать с любым шагом, положительным или отрицательным, но обязательно целым. Если мы не задаем дополнительных параметров этому свойству, то оно начинает работать как *счетчик* в MS Access, начиная с единицы и добавляя при каждом вводе тоже единицу.

Кроме того, *таблица EXEMPLAR* является подчиненной двум другим ранее определенным таблицам: BOOKS и READERS. При этом с таблицей BOOKS *таблица EXEMPLAR* связана обязательной связью, потому что не может быть ни одного экземпляра книги, который бы не был приписан конкретной книге. С таблицей READERS *таблица EXEMPLAR* связана необязательной связью, потому что не каждый экземпляр в данный момент находится на руках у читателя. Для моделирования этих связей при создании таблицы *EXEMPLAR* должны быть определены два внешних ключа (FOREIGN KEY). При этом *атрибут*, соответствующий шифру книги (мы его назовем так же, как и в *родительской таблице* — ISBN), является обязательным, то есть не может принимать неопределенных значений, а *атрибут*, который является внешним ключом для связи с таблицей READERS, является необязательным и может принимать неопределенные значения.

Необязательными там являются два остальных атрибута: дата взятия и дата возврата книги, оба они имеют *тип данных*, соответствующей календарной дате. *Атрибут*, который содержит информацию о присутствии или отсутствии книги, имеет *логический* тип. Напишем оператор создания таблицы *EXEMPLAR* в синтаксисе MS *SQL Server*:

```
CREATE TABLE EXEMPLAR
(
 EXEMPLAR_ID INT IDENTITY PRIMARY KEY,
 ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
 READER_ID Smallint NULL FOREIGN KEY references READERS (READER_ID),
 DATA_IN date,
 DATA_OUT date,
 EXIST bit
 );
```

Как мы уже знаем, не все декларативные ограничения могут быть заданы на уровне столбцов таблицы, часть ограничений может быть задана только на уровне всей таблицы. Например, если мы имеем в качестве первичного ключа не один *атрибут*, а последовательность атрибутов, то мы можем определить ограничение типа PRIMARY KEY (*первичный ключ*) только на уровне всей таблицы.

Допустим, что мы считаем экземпляры книги не подряд, а отдельно для каждого издания, тогда *таблица EXEMPLAR* в качестве первичного ключа будет иметь набор из двух атрибутов: это *шифр* книги (ISBN) и порядковый номер экземпляра данной книги (ID_EXEMPL), в этом случае оператор создания таблицы *EXEMPLAR* будет выглядеть следующим образом:

```
CREATE TABLE EXEMPLAR (
ID_EXEMPLAR int NOT NULL,
ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
READER_ID Smallint(4) NULL FOREIGN KEY references READERS (READER_ID),
DATA_IN date,
DATA_OUT date,
EXIST bit,
PRIMARY KEY (ID_EXEMPLAR, ISBN) );
```

Мы видим, что один и тот же *атрибут* ISBN, с одной стороны, является внешним ключом (FOREIGN KEY), а с другой стороны, является частью первичного ключа (PRIMARY KEY). И ограничение типа *первичный ключ* (PRIMARY KEY) задается не на

уровне одного атрибута, а на уровне всей таблицы, потому что оно содержит набор атрибутов.

То же самое можно сказать и о проверочных (CHECK) ограничениях, если условия проверки предполагают сравнения значений нескольких столбцов таблицы. Введем дополнительное ограничение для таблицы BOOKS, которое может быть сформулировано следующим образом: соавтор не может быть задан, если не задан *автор*. При описании книги допустимо не задавать ни автора, ни соавтора, или задать и автора, и соавтора, или задать только автора. Однако задание соавтора в отсутствие задания автора считается ошибочным. В этом случае оператор создания таблицы BOOKS будет выглядеть следующим образом:

```
CREATE TABLE BOOKS (
ISBN varchar(14) NOT NULL PRIMARY KEY,
TITLE varchar(120) NOT NULL,
AUTOR varchar(30) NULL,
COAUTOR varchar(30) NULL,
YEAR_PUBL smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL >= 1960 AND
 YEAR_PUBL <= YEAR(GetDate())),
PUBLICH varchar(20) NULL,
PAGES smallint CHECK(PAGES > = 5 AND PAGES <= 1000),
CHECK (NOT (AUTOR IS NULL AND COAUTOR IS NOT NULL))
);
```

Для анализа ошибок целесообразно именовать все ограничения, особенно если *таблица* содержит несколько ограничений одного типа. Для именования ограничений используется ключевое *слово* CONSTRAINT, после которого следует уникальное имя ограничения, затем *тип ограничения* и его выражения. Для идентификации ограничений рекомендуют использовать систему именования, которая легко позволит определить при получении сообщения об ошибке, которое вырабатывает *СУБД*, какое ограничение нарушено. Обычно имя ограничения состоит из краткого названия типа ограничения, далее через символ подчеркивания идет имя атрибута или таблицы, в зависимости от того, к какому уровню относится ограничение, и, наконец, порядковый номер ограничения данного типа, если к одному объекту задается несколько ограничений одного типа.

Сокращенные обозначения ограничений состоят из одной или двух букв и могут быть следующими:

- PK — для первичного ключа;
- FK — для внешнего ключа;
- CK — для *проверочного ограничения*;
- U — для ограничения уникальности;
- DF — для ограничения типа значение по умолчанию.

Приведем пример оператора создания таблицы BOOKS с именованными ограничениями:

```
CREATE TABLE BOOKS
(
 ISBN varchar(14) NOT NULL ,
 TITLE varchar(120) NOT NULL,
 AUTOR varchar (30) NULL,
 COAUTOR varchar(30) NULL,
```

```
 YEAR_PUBL smallint NOT NULL DEFAULT (YEAR(GETDATE())),
 PUBLICH varchar(20) NULL,
 PAGES smallint NOT NULL,
 CONSTRAINT PK_BOOKS PRIMARY KEY (ISBN),
 CONSTRAINT CK_YEAR_PUBL CHECK (YEAR_PUBL between 1960 AND YEAR(GetDate())),
 CONSTRAINT CK_PAGES CHECK (PAGES between 5 AND 1000),
 CONSTRAINT CK_BOOKS CHECK (NOT ((AUTOR IS NULL) AND (COAUTOR IS NOT NULL)))
 );
```

Операторы языка *SQL*, как указывалось ранее, транслируются в режиме интерпретации, в отличие от большинства алгоритмических языков, трансляторы для которых выполнены *по* принципу компиляции. В режиме интерпретации каждый оператор отдельно транслируется, то есть переводится в машинные коды, и тут же выполняется. В режиме компиляции вся *программа*, то есть совокупность операторов, сначала переводится в машинные коды, а затем может быть выполнена как единое целое. Такая особенность *SQL* накладывает ограничение на порядок описания создаваемых таблиц. Действительно, если при трансляции оператора описания *подчиненной таблицы* с указанным внешним ключом и соответствующей ссылкой на *родительскую таблицу* эта *родительская таблица* не будет обнаружена, то мы получим *сообщение об ошибке* с указанием ссылки на несуществующий *объект*. Сначала должны быть описаны все основные таблицы, а потом подчиненные таблицы.

В нашем примере с библиотекой порядок описания таблиц следующий:

1. Таблица BOOKS
2. Таблица READERS
3. Таблица CATALOG (системный каталог)
4. Таблица *EXEMPLAR*
5. Таблица RELATION_1 (дополнительная связующая таблица между книгами и системным каталогом).

Набор операторов языка *SQL* принято называть не программой, а скриптом. Тогда *скрипт*, который добавит набор из 5 взаимосвязанных таблиц *базы данных* "Библиотека" в существующую базу данных, будет выглядеть следующим образом:

```
CREATE TABLE BOOKS
(
 ISBN varchar(14) NOT NULL ,
 TITLE varchar(120) NOT NULL,
 AUTOR varchar (30) NULL,
 COAUTOR varchar(30) NULL,
 YEAR_PUBL smallint NOT NULL DEFAULT (YEAR(GETDATE())),
 PUBLICH varchar(20) NULL,
 PAGES smallint NOT NULL,
 CONSTRAINT PK_BOOKS PRIMARY KEY (ISBN),
 CONSTRAINT CK_YEAR_PUBL CHECK (YEAR_PUBL between 1960 AND YEAR(GetDate())),
 CONSTRAINT CK_PAGES CHECK (PAGES between 5 AND 1000),
 CONSTRAINT CK_BOOKS CHECK (NOT ((AUTOR IS NULL) AND (COAUTOR IS NOT NULL)))
 );

CREATE TABLE READERS
(
 READER_ID Smallint PRIMARY KEY,
 FIRST_NAME char(30) NOT NULL,
 LAST_NAME char(30) NOT NULL,
 ADRES char(50),
 HOME_PHONE char(12),
```

```
 CELL_PHONE char(12),
 BIRTH_DAY date CHECK( DateDiff(year, GetDate(),BIRTH_DAY) >=17 ),
 CONSTRAINT CK_READERS CHECK (HOME_PHONE IS NOT NULL OR CELL_PHONE IS NOT NULL)
);

CREATE TABLE CATALOG
(
 ID_CATALOG Smallint PRIMARY KEY,
 KNOWELEDGE_AREA varchar(150)
);

CREATE TABLE EXEMPLAR
(
 ID_EXEMPLAR int NOT NULL,
 ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
 READER_ID Smallint NULL FOREIGN KEY references READERS (READER_ID),
 DATA_IN date,
 DATA_OUT date,
 EXIST bit,
 PRIMARY KEY (ID_EXEMPLAR, ISBN)
);

CREATE TABLE RELATION_1
(
 ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
 ID_CATALOG smallint NOT NULL FOREIGN KEY references CATALOG(ID_CATALOG),
 CONSTRAINT PK_RELATION_1 PRIMARY KEY (ISBN,ID_CATALOG)
);
```

При написании скрипта мы добавили в оператор создания таблицы "Читатели" ограничение на уровне таблицы, которое связано с обязательным наличием хотя бы одного из двух телефонов.

4.3. Средства определения схемы базы данных

В стандарте SQL1 задается спецификация оператора описания схемы *базы данных*, но не указывается способ создания собственно *базы данных*, поэтому в различных *СУБД* используются неодинаковые подходы к этому вопросу.

Например, в *СУБД ORACLE база данных* создается в ходе установки программного обеспечения собственно *СУБД*. Все таблицы пользователей помещаются в единую базу данных. Однако они могут быть разделены на группы, объединенные в подсхемы. Понятие подсхемы не стандартизировано в *SQL* и не используется в других *СУБД*.

В состав *СУБД INGRES* входит специальная *системная утилита*, имеющая имя CREATEDB, которая позволяет создавать новые *базы данных*. *Права* на использование этой утилиты имеет *администратор* сервера. Для удаления *базы данных* существует соответствующая *утилита* DESTROYDB.

В *СУБД* MS *SQL Server* существует специальный оператор CREATE DATABASE, который является частью языка определения данных, для удаления *базы данных* в языке определен оператор DROP DATABASE. Правами на создание баз данных наделяются администраторы баз данных, которых в общем случае может быть несколько. Правами более высокого уровня обладает *администратор* сервера баз данных (*SQL Server*), который и может предоставить *права* администратора *базы данных* другим пользователям сервера. Администраторы баз данных могут удалить только свою базу данных. Приведем пример оператора создания схемы *базы данных* в MS *SQL Server*:

CREATE DATABASE database_name

[ON [PRIMARY][<спецификация файла>[,...n]][,<группа файлов> [,...n]]]

[LOG ON { спецификация файла> [,...n]}][FOR LOAD | FOR ATTACH]

<спецификация файла> ::=

([NAME = логическое имя файла,]FILENAME = 'физическое имя файла'

[, SIZE = размер][, MAXSIZE = { максимальный размер | UNLIMITED }]

[, FILEGROWTH = инкремент увеличения файла]) [,...n]

<группа файлов>::= FILEGROUP имя группы файлов спецификация файла> [,...n]

Здесь

- database_name — имя базы данных, идентификатор в системе;
- ON — ключевое слово, которое означает, что далее будут заданы спецификации файлов, которые будут использованы для размещения базы данных;
- PRIMARY — ключевое слово, которое определяет первичное файловое пространство, в котором будет размещена собственно база данных;
- LOG ON — ключевое слово, которое задает спецификацию файлов, которые будут использованы для хранения журналов транзакций;
- FOR LOAD — ключевое слово, которое определяет, что после создания базы данных будет произведена загрузка базы данных данными;
- FOR ATTACH — предложение, которое определяет, что база данных для управления будет подсоединена к другому серверу.

Почти все параметры, кроме имени *базы данных,* являются необязательными, поэтому оператор создания простой *базы данных* "Библиотека" может выглядеть следующим образом:

CREATE DATABASE Library

Для изменения схемы *базы данных* в MS *SQL Server* может быть использована *команда*:

ALTER DATABASE database

{ ADD FILE спецификация файлов> [,...n] [TO FILEGROUP filegroup_name]

| ADD LOG FILE спецификация файлов> [,...n]

| REMOVE FILE имя_файла

| ADD FILEGROUP имя группы файлов REMOVE FILEGROUP имя группы_файлов

| MODIFY FILE спецификация файлов>

| MODIFY FILEGROUP имя_группы_файлов имя_свойства_группы файлов}

Здесь свойства группы файлов определяет одно из допустимых ключевых слов:

- READONLY — только для чтения;
- READWRITE — для чтения и записи;
- DEFAULT — назначает данную группу файлов в качестве группы по умолчанию, в которой размещаются данные, если не задано дополнительных условий размещения информации.

Как видно, при изменении схемы *базы данных* в нее могут быть добавлены (ADD) дополнительные файлы и файловые группы или удалены (REMOVE) ранее определенные файлы или файловые группы. Назначение этих файлов нам будет более понятно

после того, как мы познакомимся с физическими моделями и файловыми структурами, используемыми для хранения данных в базах данных.

Сейчас мы познакомимся с последней командой, которая предназначена для удаления *базы данных*. В MS *SQL Server* это *команда* имеет следующий *синтаксис*:

DROP DATABASE database_name

После выполнения этой команды уничтожается вся *база данных* вместе с содержащимися в ней данными.