МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ И SQL

Методические рекомендации к лабораторным работам для студентов специальности 6-05-0611-04 «Электронная экономика» очной и заочной форм обучения

Могилев 2025

УДК 004.65
ББК 32.973.26-0.18.2
Р36

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «10» сентября 2025 г., протокол № 2

Составители: канд. техн. наук, доц. К. В. Захарченков;
канд. техн. наук, доц. Т. В. Мрочек

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации содержат описание десяти лабораторных работ по дисциплине «Реляционные базы данных и SQL» для студентов специальности 6-05-0611-04 «Электронная экономика» очной и заочной форм обучения. Рассматриваются методологии IDEF1X, IE, UML и язык Transact-SQL в среде Microsoft SQL Server.

Учебное издание

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ И SQL

Ответственный за выпуск	В. В. Кутузов
Корректор	А. Т. Червинская
Компьютерная верстка	М. М. Дударева

Подписано в печать 23.12.2025. Формат 60x84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. 2,79. Уч.-изд. л. 2,94 . Тираж 21 экз. Заказ № 910.

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью учебной дисциплины «Реляционные базы данных и SQL» является формирование профессиональных компетенций, необходимых для создания и ведения современных баз данных, а также управления базами данных с ориентацией на решение различных прикладных задач.

В результате освоения учебной дисциплины студент

познает:

– основные понятия реляционных баз данных (БД), основы построения, функционирования и технологии организации реляционных БД;

– язык SQL;

– способы защиты данных;

– приемы работы в распределенных и многопользовательских реляционных БД;

научится:

– проектировать базы данных реляционного типа;

– реализовывать реляционную модель базы данных в используемой СУБД, используя основные конструкции структурированного языка запросов;

– организовывать ввод информации в БД, осуществлять поиск информации и вывод отчетов;

– работать с многопользовательской БД;

приобретет навыки:

– применения методов, средств и технологий разработки информационных моделей и их программной реализации в выбранной СУБД;

– применения теории и стандартов языков описания и манипулирования данными, теоретических основ построения выбранной модели данных;

– применения технологий и техники программной реализации реляционных баз данных, методов и языковых средств манипулирования данными, поддержания целостности, непротиворечивости и защиты информации.

Методические рекомендации содержат описание десяти лабораторных работ, задания, контрольные вопросы, список литературы [1–12], которую необходимо использовать при подготовке к защите лабораторных работ.

1 Разработка технического задания на проектирование информационной системы

Цель: разработать проект технического задания на проектирование БД информационной системы для выбранной предметной области.

Теоретические положения

База данных является основным компонентом любой информационной системы (ИС), поэтому проект технического задания разрабатывается на основе [1] и имеет структуру, основные элементы которой представлены далее в примере.

Пример технического задания на проектирование БД «Библиотека».

1 Общие сведения.

1.1 Объект автоматизации – университетская библиотека.

1.2 Документы, на основании которых создается система.

Систематический и предметный каталоги; должностные инструкции; правила пользования библиотечным фондом; акт на списание книг.

2 Назначение и цели создания системы.

2.1 Назначение системы.

Проектируемую БД предполагается использовать на рабочих местах библиотекарей для увеличения скорости обслуживания читателей. Система позволит облегчить процесс поиска книг, т. к. он будет вестись автоматизированно. Применение БД позволит упростить процессы подбора литературы, проверки наличия книг в фонде, слежения за сохранностью книжного фонда.

2.2 Цели создания системы.

Систему предполагается создать для улучшения качества обслуживания читателей и ускорения работы библиотекаря.

Критерии оценки достижения целей системы:

– увеличение числа обслуживаемых читателей за счет увеличения скорости обслуживания;

– уменьшение вероятности потери информации о книгах;

– уменьшение вероятности неверного закрепления книг за читателем.

3 Характеристика объектов автоматизации.

3.1 Краткие сведения.

Примечание – З*десь необходимо*:

– выполнить описание работы объекта автоматизации (*например*, *отдела предприятия*);

– перечислить основные функции объекта автоматизации и указать информацию, *подлежащую хранению*;

– перечислить категории пользователей будущей БД, *определить права доступа разных категорий пользователей к различной информации в БД*.

Университетская библиотека включает следующие отделы: отделы обслуживания (абонемент учебной литературы, абонемент научной литературы, читальный зал), отдел комплектования, справочно-библиографический отдел.

Отделы обслуживания работают с читателями дневного отделения по читательским билетам, заочного – по зачетной книжке. Номер читательского билета соответствует номеру формуляра, который содержит информацию о выданных книгах.

После окончания каждого курса студент должен сдать все неиспользуемые книги. По завершении обучения в университете студент сдает все библиотечные книги, подписывает обходной лист.

Библиотекой также могут пользоваться преподаватели – сотрудники университета, которым также выдаются читательские билеты. Кроме того, преподаватели могут заказывать учебную литературу.

Каждый отдел библиотеки выполняет свои функции.

В функции отделов обслуживания входят: запись студентов в библиотеку; выдача книг читателям и прием книг; ведение картотек читателей.

Информация, подлежащая хранению: инвентарный номер книги (шифр), автор, название, издательство, год издания, цена, отдел хранения книги, номер читательского билета (номер формуляра), имя и фамилия студента, год поступления, год окончания (отчисления), факультет и специальность, форма обучения (дневная или заочная).

Пользователи будущей БД: директор библиотеки, заместитель библиотеки, заведующие отделами, библиотекари.

Библиотекарь имеет доступ к информации о читателях, о книжном фонде.

3.2 Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды.

В отделах обслуживания и периодики БД будет использоваться для поиска книг, для поиска читателя по номеру читательского билета, просмотра его задолженностей, внесения сведений о выдаваемых книгах.

4 Требования к системе.

4.1 Требования к системе в целом.

Система должна удовлетворять следующим требованиям:

– надежности;

– безопасности;

– защиты информации от несанкционированного доступа;

– доступности БД с любого компьютера в библиотечной сети;

– защищенности информации, хранящейся в системе, от аварийных ситуаций, влияния внешних воздействий;

– к квалификации персонала. В библиотеке работают служащие с высшим и средним специальным образованием. Персонал должен быть обучен правилам работы с ИС. Наличие специального технического образования не требуется.

4.2 Требования к функциям (задачам), выполняемым системой.

Примечание – При перечислении функций рекомендуется указать форму получения информации по каждой функции (*в виде запроса* (*результатом выполнения которого является виртуальная таблица*) *либо отчета* (*или иных видов бумажной документации*)).

Функции, выполняемые подсистемами объекта автоматизации:

– запрос информации о книгах, выданных студенту, сотруднику, по их идентификатору;

– запрос информации о наиболее часто запрашиваемых книгах в данной предметной области;

– запрос информации (наименование, автор, ISBN, цена, количество, стоимость) для формирования отчета по книгам, закупленным библиотекой в определенный месяц.

4.3 Требования к видам обеспечения.

Программное обеспечение системы не должно зависеть от аппаратных средств компьютера. Необходимое программное обеспечение: MS Word 2019, MS SQL Server 2022.

Задание

Выбрать предметную область (согласовать с преподавателем тему) и разработать техническое задание на проектирование автоматизированной информационной системы для выбранной предметной области.

Содержание отчета: тема и цель работы; техническое задание объемом не менее 4 страниц.

Контрольные вопросы

1 Указать состав и содержание ТЗ на автоматизированную ИС.
2 Каковы правила оформления ТЗ?
3 Каков порядок разработки, согласования и утверждения ТЗ на ИС?

2 Основы использования CASE-средств концептуального проектирования информационной модели системы

Цель: изучить основные понятия баз данных; изучить основы применения нотации UML при проектировании баз данных; разработать информационную модель БД ИС.

Теоретические положения

Рассмотрим наиболее важные понятия, используемые при проектировании информационной модели БД.

Первичный ключ (Primary Key – PK) – это один или несколько атрибутов, однозначно идентифицирующих каждый экземпляр сущности.

Первичный ключ должен удовлетворять двум требованиям – уникальности и безызбыточности (минимальности). Уникальность ключа означает, что в любой момент времени таблица БД не может содержать никакие две различные записи, имеющие одинаковые значения ключевых полей. Требование безызбыточности ключевых полей означает, что только сочетание значений выбранных полей отвечает требованиям уникальности записей таблицы БД. Это означает

также, что ни одно из входящих в ключ полей не может быть исключено из него без нарушения уникальности.

Исходя из требований уникальности и безызбыточности атрибут (атрибуты), выбираемый для первичного ключа, должен:

1) уникально идентифицировать значение сущности;

2) не содержать неопределенное значение NULL;

3) сохранять уникальность с течением времени;

4) содержать как можно меньше символов, что облегчает индексацию и восстановление данных.

Составной ключ – первичный ключ, состоящий из нескольких атрибутов.

При формировании составного ключа не следует включать в состав ключа поля таблицы, значения которых сами по себе однозначно идентифицируют записи в таблице. Например, не стоит создавать ключ, содержащий одновременно поля «номер паспорта» и «уникальный номер налогоплательщика», т. к. каждый из этих атрибутов может однозначно идентифицировать записи в таблице.

В качестве первичного ключа может использоваться естественный ключ или суррогатный.

Естественный ключ – это атрибут или набор атрибутов, которые однозначно идентифицируют значение сущности (записи таблицы) и имеют некий физический смысл вне БД (номер детали и т. д.). Уникальность записи может достигаться не только значением уникального кода внешнего объекта, но, например, и датой вставки записи в таблицу (первичный ключ РК состоит из значения кода объекта и даты вставки записи в БД).

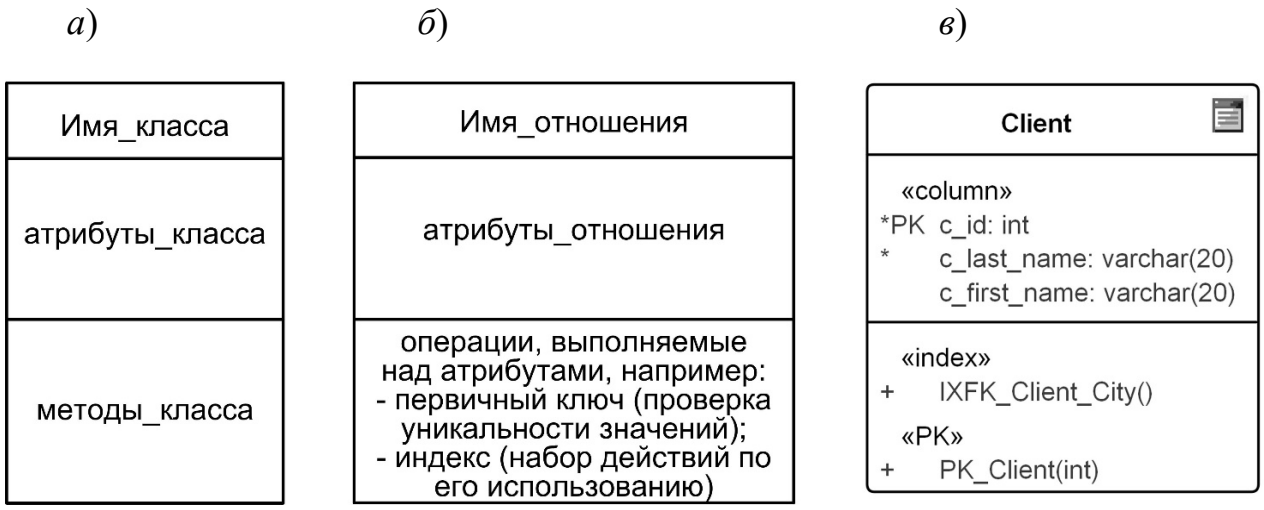
Суррогатный ключ – автоматически сгенерированное значение внутреннего ID – идентификатора сущности, никак не связанное с информационным содержанием записи, которое имеет некий физический смысл только внутри БД и уникально на всем протяжении жизни сущности; обычно в роли суррогатного ключа могут использоваться данные типа INT, SMALLINT, TINYINT.

Внешний ключ (Foreign Key – FK) – это столбец (или группа столбцов), содержащий значения, совпадающие со значениями первичного ключа в другой таблице. Он обеспечивает связь сущностей главной и подчиненной таблиц. Внешний ключ не обладает свойством уникальности и обычно является частью составного первичного ключа или неключевым столбцом.

На сегодняшний день существует большое количество инструментов для разработки схем баз данных, например, www.drawdb.app/editor, DbDiagram.io (https://dbdiagram.io) и т. д. В данной лабораторной работе разработка схемы базы данных будет вестись с использованием CASE-средства Sparx Systems Enterprise Architect [9, с. 286–293].

Разработка схемы базы данных в Enterprise Architect может выполняться с использованием нотаций (нотация – система условных графических обозначений и правил их использования для описания различных категорий моделируемой предметной области) UML, IDEF1X, Information Engineering (IE). Во всех указанных нотациях разработка схемы базы данных в Enterprise Architect ведется с использованием основных понятий языка UML «класс», «атрибут» и «метод».

На рисунке 2.1 показано, как основные понятия языка UML (Unified Modeling Language – унифицированный язык моделирования, язык графического описания для объектного моделирования в области разработки программного обеспечения) связаны с разработкой баз данных. Прямоугольник обозначает класс или отношение, атрибуты класса – атрибуты отношения, к методам отношения относят первичный ключ, внешний ключ, индексы, т. к., например, первичный ключ – это не просто поле, обладающее определенными свойствами, но и наличие необходимости выполнять проверку уникальности значений.



а – графическое изображение структуры класса в UML; *б* – графическое изображение структуры отношения в Enterprise Architect; *в* – пример отношения в Enterprise Architect

Рисунок 2.1 – Структура отношения в Enterprise Architect

Выбор той или иной нотации (UML, IDEF1X, IE) может определяться как корпоративными стандартами разработчика, так и требованиями заказчика.

При создании новой модели необходимо выбрать File → New project, задать имя файла и тип проекта Enterprise Architect Project (*.eap).

Далее в окне Model Wizard в разделе Database следует выбрать тип добавляемой модели: Data Model – SQLServer.

В браузере проекта появится папка Model, содержащая пакет Data Model, предназначенный для хранения перечня разрабатываемых объектов базы данных, распределенных по двум следующим пакетам:

1) пакет Logical Model – включает логическую модель БД, состоящую из таблиц, атрибутов, ограничений на значения атрибутов и связей между таблицами;

2) пакет «Database» SQLServer – содержит процедурную часть проектируемой БД, которая в различных СУБД состоит из различных компонентов и реализуется по-разному. Так, для MS SQL Server данный пакет может содержать хранимые процедуры, функции, запросы, представления, последовательности, триггеры, таблицы (разработанные с помощью инструмента Database Builder из папки Tools). Для MS Access данный пакет может содержать запросы, пред-

ставления и таблицы (разработанные с помощью инструмента Database Builder из папки Tools).

В папке Model могут храниться пакеты Data Model для различных СУБД, которые можно добавить, нажав правой клавишей мыши по папке Model и выбрав из контекстного меню Add → Add a Model using Wisard.

Добавить сразу несколько пакетов Data Model для различных СУБД можно непосредственно при создании проекта Enterprise Architect Project, отметив галочкой требуемые целевые СУБД. Для выполнения последующих лабораторных работ можно сразу отметить галочками Data Model – MSAccess и Data Model – SQLServer. Сформированные пакеты Data Model – MSAccess и Data Model – SQLServer появятся в браузере проекта.

Далее в пакете Logical Model на диаграмме Logical Model с помощью инструмента Table меню Diagram / Toolbox нужно создать таблицы базы данных.

Вначале на диаграмме Logical Model создадим таблицу Client и установим для нее целевую СУБД SQLServer2012.

Выделив созданную таблицу и выбрав в контекстном меню пункт Columns, для таблицы Client создадим первичный ключ c_id. В меню Client Columns and Constraints в разделе Columns необходимо указать тип данных int столбца c_id, задать галочкой ограничение первичного ключа PK. Одновременно в разделе Constraints появится автоматически сгенерированное название ограничения первичного ключа PK_Client.

Зададим для первичного ключа свойство IDENTITY, т. е. сделаем столбец первичного ключа идентификатором (счетчиком со свойством AutoNum = True) с начальным значением StartNum = 1 и приращением Increment = 1. Добавим в таблицу Client столбцы last_name и first_name. Звездочка слева от названия столбца на схеме таблицы (* PK id_client, * last_name) означает, что для такого столбца задано ограничение Not Null (рисунок 2.2).

Изменить цвет таблиц схемы базы данных со Standard на белый можно, выбрав Diagram → Appearance → White Board.

Изменить графическую нотацию можно, выбрав в пункте меню Diagram → Properties. Появится окно Data Modeling Diagram: Logical Model, в котором нужно выбрать раздел Connectors и указать выбранный тип Connector Notation: UML 2.1, Information Engineering или IDEF1X.

Связи. Типы данных столбца первичного ключа и соответствующего столбца внешнего ключа должны полностью совпадать (кроме автоинкрементируемости, которой не должно быть у внешнего ключа). Это позволяет СУБД не затрачивать дополнительные ресурсы на преобразование данных при сравнении значений первичного и внешнего ключа.

В Enterprise Architect на панели инструментов для создания связей между отношениями имеется только один инструмент – ассоциация. Поэтому для того, чтобы реализовать требуемый тип связи (1:1, 1:M, M:M) между отношениями с помощью одной лишь ассоциации, нужно хорошо понимать устройство и смысл различных типов связей.

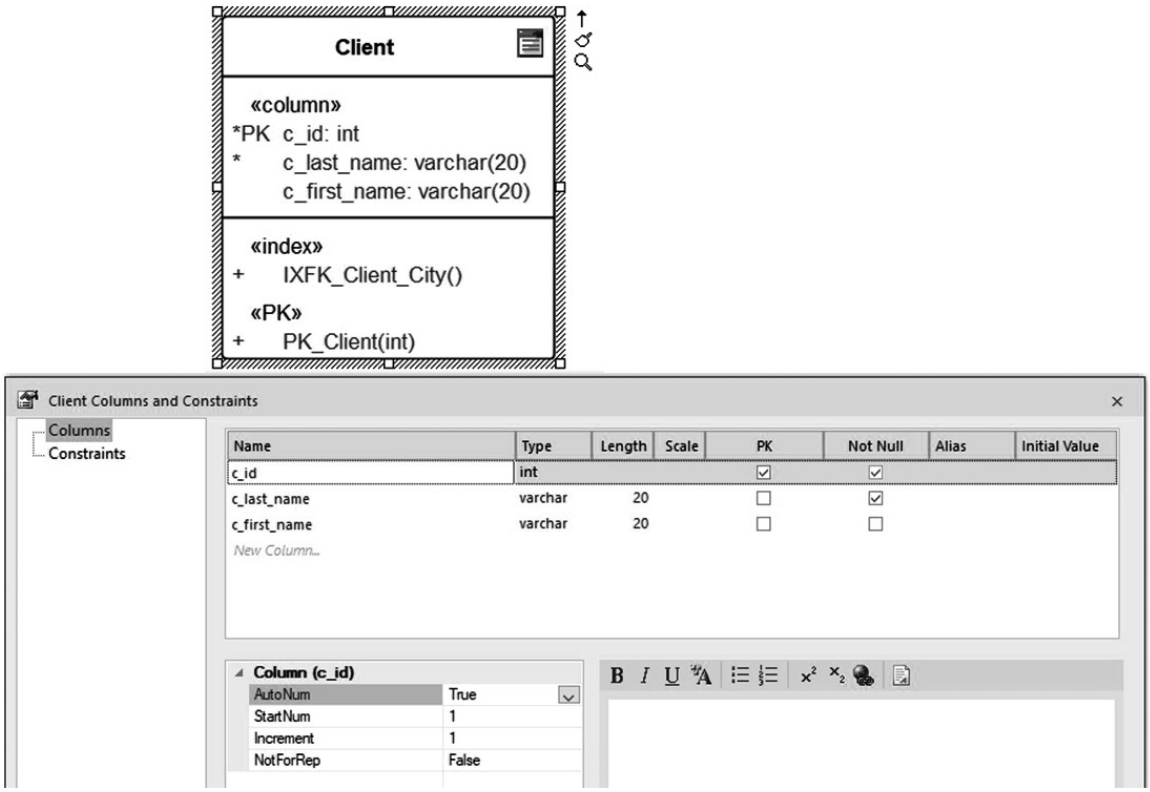


Рисунок 2.2 – Создание таблицы Client

Ассоциация – это самый общий тип связи, который просто показывает, что некие сущности взаимосвязаны, при этом тип и особенности этой взаимосвязи не отражаются, хотя некоторые параметры можно указать (направленность связи, мощность связи).

Связь, т. е. ассоциация, позволяющая установить некоторые взаимосвязи между сущностями, имеют следующие свойства:

1) **направленность**. Большинство средств проектирования схем баз данных (не только Enterprise Architect) при создании связи придерживается правила «от дочерней к родительской» таблице. В Enterprise Architect в нотациях UML, IDEF1X, IE стрелки связей на схемах изображаются просто **от дочернего отношения к родительскому**, а не от конкретного поля дочерней таблицы к конкретному полю родительской таблицы. Понять, по каким полям связаны дочернее и родительское отношения, можно из надписи на самой связи.

Главная (родительская) таблица находится на стороне «один» связи, а подчиненная (дочерняя) таблица находится на стороне «много». Связи многие-ко-многим симметричны;

2) **мощность** (кардинальность, cardinality) связи – свойство связи, которое служит для указания количества взаимосвязанных строк в таблицах, объединенных связью (например: 1:M, 1:5 (один к пяти) и т. д.).

Мощность связи задается при ее создании в окне Foreign Key Constraint и может принимать следующие значения: 1..*, 1.., 1, 0..1, 0..*, 0, *.

Значения мощности связи определяются исключительно требованиями предметной области, которую описывает база данных. Например:

– в системе обязательно должен быть хотя бы один модератор, но их может быть не более двух → «1 к 1..2»;

– сотрудник при отсутствии заказов может не вести ни один договор, но если заказы есть, то не более пяти договоров → «1 к 0..5»;

– детское автокресло может быть никем не арендовано, и каждый человек может арендовать не более трех автокресел → «0…1 к 0..3».

Ограничения мощности связей можно контролировать с помощью триггеров или из клиентского приложения;

3) **обязательность** (Nulls Allowed или No Nulls) показывает, может ли атрибут внешнего ключа принимать значение «Null» в таблице БД;

4) **тип**. Существует три классических вида связей между таблицами [2–11].

В связи 1:1 (один к одному) может участвовать не больше одного экземпляра каждой из связываемых сущностей, т. е. одной записи в таблице *A* может соответствовать не более одной записи в таблице *B*. Например, каждый преподаватель ведет не более одной дисциплины, и каждая дисциплина ведется не более чем одним преподавателем.

В связи 1:M (один ко многим) один экземпляр (или даже нуль экземпляров) родительской сущности может быть связан с любым количеством экземпляров дочерней сущности. И наоборот – для связи M:1 (многие к одному). Например, в связи 1:M один преподаватель может преподавать любое количество учебных дисциплин, но каждая дисциплина преподается не более чем одним преподавателем.

В связи M:M (многие-ко-многим или неспецифическая связь) каждый из экземпляров обеих сущностей может быть связан с любым числом экземпляров другой сущности. Например, каждый преподаватель может вести любое количество дисциплин, и каждая дисциплина может преподаваться любым количеством преподавателей.

Связь между сущностями может быть одного из следующих типов:

а) идентифицирующая связь;

б) неидентифицирующая (обязательная или необязательная);

в) категоризации (типизирующая);

г) рекурсивная (связь сущности самой с собой).

Внешний ключ в зависимости от типа связи может стать частью составного ключа дочерней сущности или неключевым атрибутом дочерней сущности. С помощью внешнего ключа экземпляр дочерней сущности ссылается на соответствующий экземпляр родительской сущности.

Создание идентифицирующей связи.

Идентифицирующей является связь между двумя сущностями, в которой каждый экземпляр подчиненной (дочерней) сущности идентифицируется (однозначно определяется) значениями атрибутов родительской сущности. Это означает, что экземпляр подчиненной сущности зависит от независимой родительской сущности и не может существовать без экземпляра родительской сущности. Соответственно, запись в дочерней таблице не может существовать без соответствующей записи в родительской таблице. Для гарантированного обеспечения этого свойства атрибуты первичного ключа родительской сущно-

сти мигрируют в состав *первичного ключа* дочерней сущности и помечаются там как внешний ключ (FK). И при генерации скрипта БД атрибутам внешнего ключа присваивается признак NOT NULL, что означает невозможность внесения записи в дочернюю таблицу без наличия идентификационной информации из родительской таблицы.

В качестве примера рассмотрим взаимосвязь между сущностями «Клиент» и «Заказ», которые представлены на рисунке двумя таблицами Client и Order.

Каждый клиент может иметь много заказов, потому на схеме базы данных это можно показать связью типа 1:M, а т. к. по бизнес-правилам каждый заказ обязан быть строго привязан к соответствующему клиенту и может быть определен только через связь с данным клиентом (и не имеет смысла без конкретного клиента), связь будет идентифицирующей.

Создадим связь типа 1:M. Для этого **от дочернего отношения** Order протянем ассоциацию **к родительскому отношению** Client.

В появившемся окне Foreign Key Constraint будет предложено создать в дочернем отношении Order столбец внешнего ключа c_id (*) и задать имя для ограничения внешнего ключа FK_Order_Client.

После нажатия OK ограничение внешнего ключа FK_Order_Client и связь «FK» будут созданы.

В соответствии с определением идентифицирующей связи внешний ключ нужно сделать частью составного первичного ключа. Открыв из контекстного меню таблицы Order окно Columns, нужно для столбца o_c_id поставить галочку в чекбоксе PK.

В результате будет создана идентифицирующая связь (типа 1:M). Столбец внешнего ключа o_c_id в составе первичного ключа будет помечен обозначением pfK (рисунок 2.3).

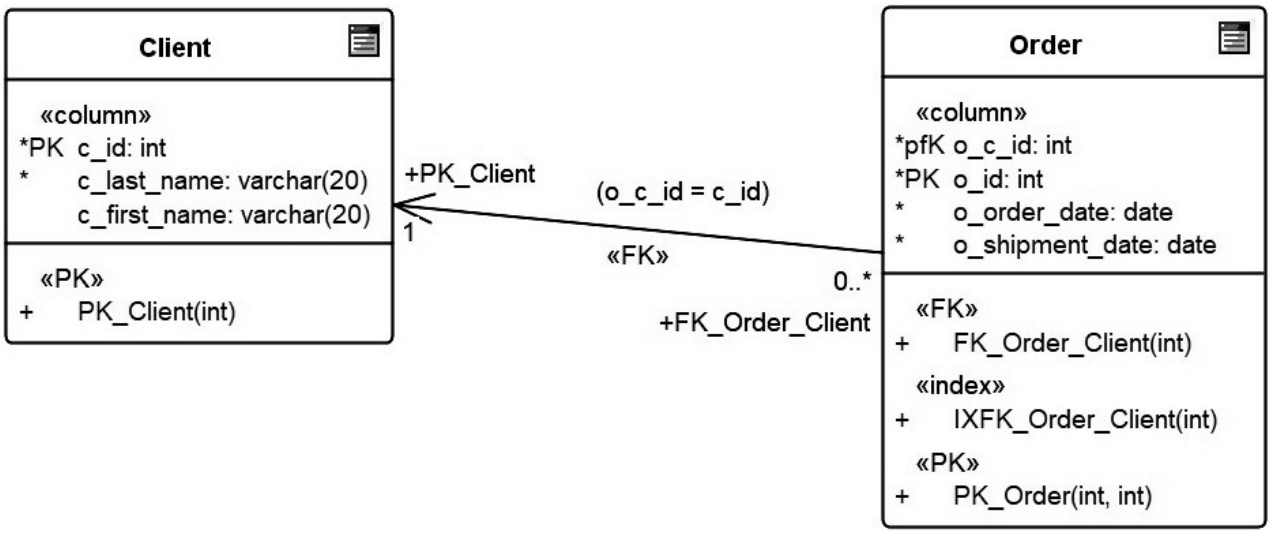


Рисунок 2.3 – Идентифицирующая связь 1:M

Если выделить созданную таблицу Client, выбрать в контекстном меню пункт Code Engineering → Generate DDL, в появившемся окне Generate DDL задать Generate To DDL Execution Engine и нажать кнопку Generate, то будет ав-

томатически сгенерирован код SQL, описывающий таблицу Client. Указанный код будет размещен на вкладке Execute DDL инструмента Database Builder создания физической модели базы данных.

Приведем SQL-код таблицы Client:

```
CREATE TABLE [Client]
(
    [c_id] int NOT NULL IDENTITY (1, 1),
    [c_last_name] varchar(20) NOT NULL,
    [c_first_name] varchar(20) NULL
)
```

В разделе Client.PK.CIient вкладки Execute DDL находится автоматически сгенерированный SQL-код ограничения первичного ключа PK_Client.

```
ALTER TABLE [Client]
ADD CONSTRAINT [PK_Client]
    PRIMARY KEY CLUSTERED ([c_id] ASC)
```

SQL-код таблицы Order:

```
CREATE TABLE [Order]
(
    [o_c_id] int NOT NULL,
    [o_id] int NOT NULL,
    [o_order_date] date NOT NULL,
    [o_shipment_date] date NOT NULL
)
```

SQL-код ограничения первичного ключа таблицы Order:

```
ALTER TABLE [Order]

ADD CONSTRAINT [PK_Order]
    PRIMARY KEY CLUSTERED ([o_c_id] ASC, [o_id] ASC)
```

SQL-код ограничения внешнего ключа таблицы Order:

```
ALTER TABLE [Order] ADD CONSTRAINT [FK_Order_Client]
    FOREIGN KEY ([o_c_id]) REFERENCES [Client] ([c_id])
```

SQL-код, обеспечивающий ссылочную целостность данных в обеих связанных таблицах в ограничении внешнего ключа:

```
ON DELETE No Action
```

ON UPDATE No Action

SQL-код некластерного индекса, созданного для столбца внешнего ключа таблицы Order:

```
CREATE NONCLUSTERED INDEX [IXFK_Order_Client]
ON [Order] ([o_c_id] ASC)
```

Приведем пример **идентифицирующей связи типа 1:1**.

На рисунке у каждого клиента может быть только один паспорт, и у каждого паспорта может быть только один владелец, поэтому между сущностями Client и Passport связь 1:1 (рисунок 2.4). Так как каждый паспорт обязан быть строго привязан к соответствующему клиенту и может быть определен только через связь с данным клиентом (и не имеет смысла без конкретного клиента), связь будет идентифицирующей.

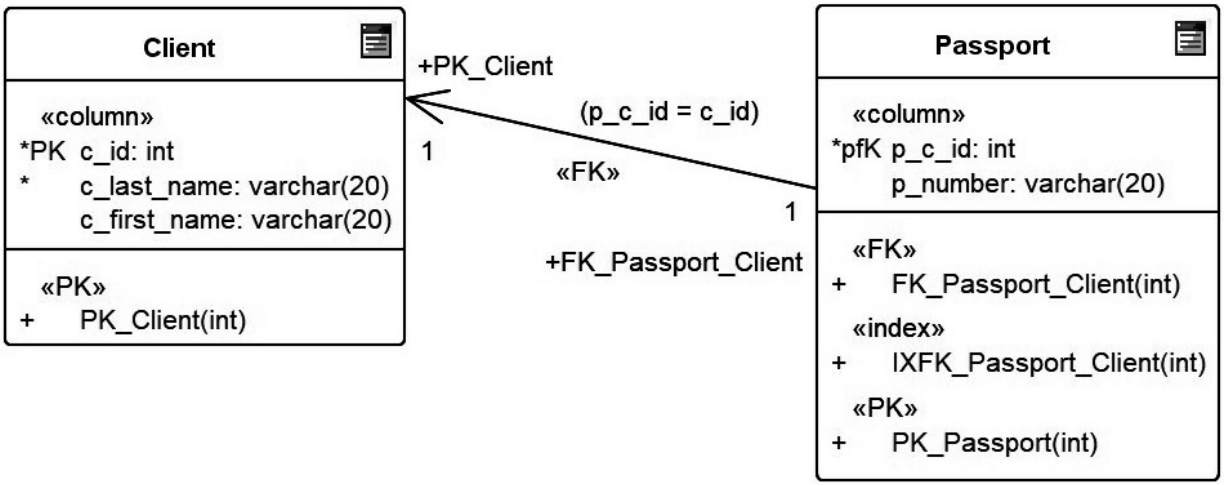


Рисунок 2.4 – Идентифицирующая связь 1:1

Идентифицирующая связь «один к одному» на рисунках изображается практически так же, как 1:M (отличие состоит в числах, указывающих мощность связи, и в том, что первичный ключ в дочерней таблице одновременно является и внешним).

Создание неидентифицирующей связи.

Неидентифицирующей называется связь между двумя сущностями, в которой каждый экземпляр подчиненной сущности не зависит от значений атрибутов родительской сущности и может существовать без экземпляра родительской сущности. Кортежу дочернего отношения может не соответствовать ни одного кортежа родительского отношения, а сам кортеж дочернего отношения может быть полностью определен без использования значения первичного ключа соответствующего кортежа родительского отношения [9]. Атрибуты первичного ключа родительской сущности мигрируют в подчиненную, чтобы стать там *внешним ключом* (неключевыми атрибутами).

Для неидентифицирующей связи необходимо указать **обязательность связи** (Nulls Allowed или No Nulls), которая показывает, может ли атрибут внешнего ключа принимать значение «Null» в таблице БД.

Неидентифицирующая связь называется **обязательной** (No Nulls), если все экземпляры дочерней сущности должны участвовать в связи. В случае обязательной связи несмотря на то, что внешний ключ не войдет в состав первичного ключа дочерней сущности, при генерации кода БД атрибут внешнего ключа получит признак Not Null.

Неидентифицирующая связь называется **необязательной** (Nulls Allowed), если некоторые экземпляры дочерней сущности могут не участвовать в связи. В случае необязательной связи внешний ключ дочерней сущности может принимать значение NULL. Это означает, что экземпляр дочерней сущности не будет связан ни с одним экземпляром родительской сущности.

При создании неидентифицирующей связи первичный ключ родительского отношения в процессе миграции в дочернее отношение становится неключевым атрибутом и внешним ключом.

На рисунке 2.5 показаны сущности Client и City. Поскольку эти сущности могут идентифицироваться независимо друг от друга, между ними имеется *неидентифицирующая* связь. Если существует бизнес-правило, в соответствии с которым клиент в обязательном порядке должен соотноситься с городом, то связь между отношениями Client и City будет *обязательной*. Поскольку один клиент относится к одному городу, а из одного города может быть множество клиентов, то устанавливаемая связь будет иметь тип 1:M.

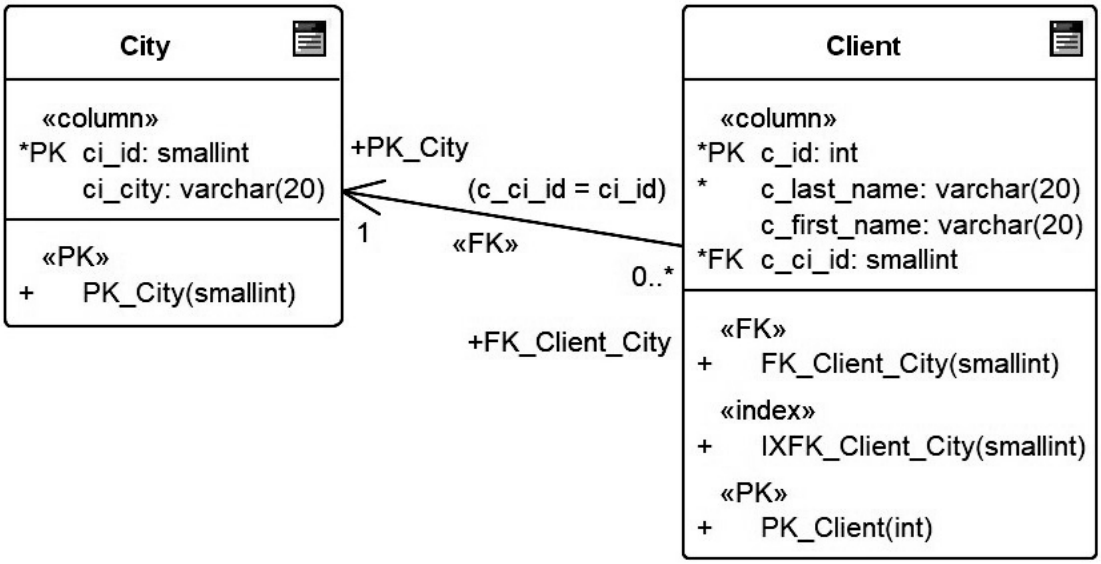


Рисунок 2.5 – Неидентифицирующая обязательная связь

Для того, чтобы создаваемую связь определить как **неидентифицирующую**, внешний ключ не включен в состав первичного ключа, а размещен среди неключевых атрибутов. Для того, чтобы создаваемую неидентифицирующую связь определить как **обязательную**, для столбца внешнего ключа c_ci_id в отношении Client установлен признак Not Null (c_ci_id помечен звездочкой).

Если существует бизнес-правило, в соответствии с которым клиент необязательно должен сопоставляться с городом, то неидентифицирующая связь между отношениями Client и City будет *необязательной*. Для того, чтобы создаваемую связь определить как **необязательную**, столбцу внешнего ключа c_ci_id в отношении Client разрешено принимать значения Null (рисунок 2.6).

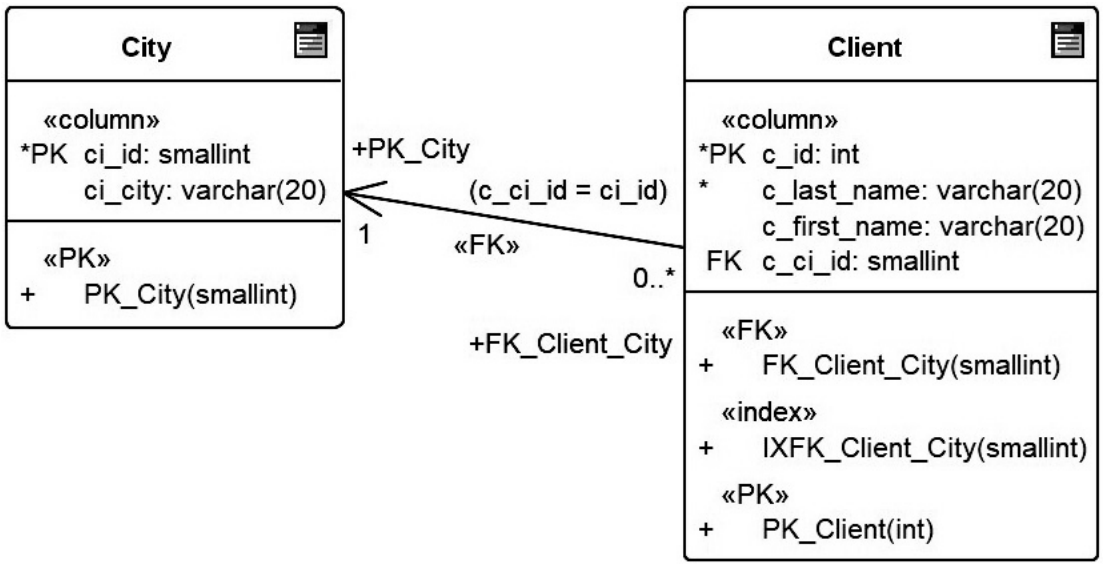


Рисунок 2.6 – Неидентифицирующая необязательная связь

Создание связи M:M.

Связь M:M – ассоциация, связывающая два отношения таким образом, что одному кортежу любого из связанных отношений может соответствовать произвольное количество кортежей второго отношения.

Связь M:M может существовать только на даталогическом уровне проектирования. На физическом уровне проектирования, поскольку СУБД не поддерживают связь M:M, такая связь организуется с помощью дополнительной ассоциативной сущности и двух идентифицирующих связей 1:M.

На рисунке 2.7 ассоциативная сущность m2m_seller_product отражает свойства связи M:M. Атрибут «дата поставки товара» не является ни свойством продавца, ни свойством товара, поэтому столбец m2m_delivery_date размещен в дополнительной ассоциативной сущности.

Создание рекурсивной связи. Рекурсивная связь – это неидентифицирующая необязательная связь сущности с самой собой (одна и та же сущность является и родительской, и дочерней одновременно), которая отражает тот факт, что экземпляр сущности может быть связан с другим экземпляром той же самой сущности. Рекурсивную связь создают с помощью рекурсивного внешнего ключа.

Рекурсивный внешний ключ – это разновидность внешнего ключа, использующая в случае, когда таблица ссылается на саму себя. Атрибут рекурсивного внешнего ключа размещен в той же таблице, что и первичный ключ, и содержит в себе копии значений первичного ключа этой же таблицы.

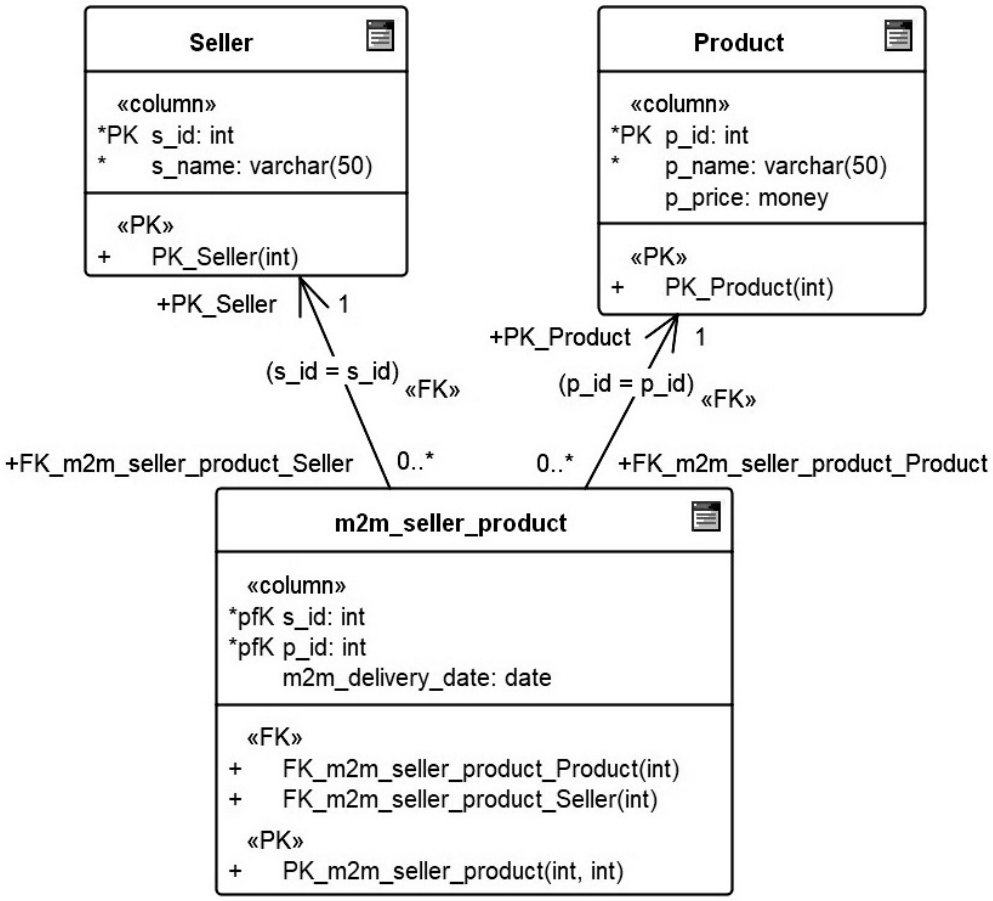


Рисунок 2.7 – Создание связи M:M

Мигрировавший в качестве внешнего ключа атрибут не может появиться дважды в одной сущности под одним именем, поэтому должен получить другое имя (имя роли). На рисунке 2.8 показан пример создания рекурсивной связи типа «руководит / подчиняется», когда информация о руководителе (senior manager) содержится в той же сущности, что и информация о подчиненных (managers). Чтобы показать присутствие руководителя в таблице, создается рекурсивная связь, и в окне создания внешнего ключа в разделе Child определяется новый атрибут для внешнего ключа с именем m_senior_manager_id (при этом создавать индекс для внешнего ключа здесь не нужно).

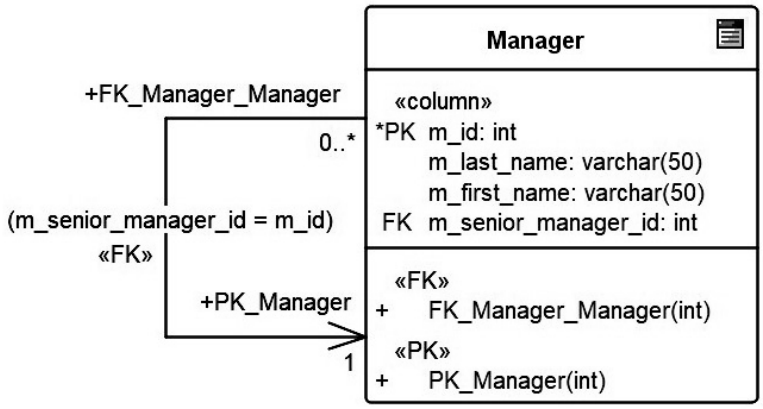


Рисунок 2.8 – Рекурсивная связь

Имя роли (Rolename, функциональное имя) – синоним атрибута внешнего ключа, который показывает, какую роль играет атрибут внешнего ключа в дочерней сущности.

Имя роли в обязательном порядке используется в двух случаях: если используется рекурсивная связь или если два или более атрибута одной сущности имеют одинаковые домены значений, но разный смысл. Например, на рисунке 2.9 отображены сущности, характеризующие процесс обмена валюты, в котором участвуют проданная и купленная валюты.

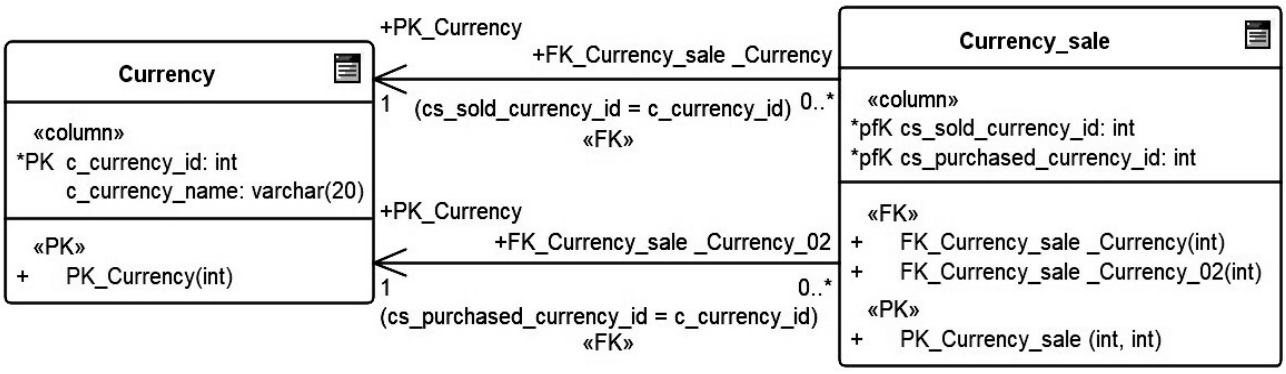


Рисунок 2.9 – Использование атрибутов, имеющих общий домен значений, но разный смысл

Сущности Currency (валюта) и Currency_sale (продажа валюты) должны быть связаны дважды, и первичный ключ сущности Currency дважды мигрирует в качестве внешнего ключа в сущность Currency_sale. Один мигрировавший атрибут будет содержать идентификатор проданной валюты, а другой – идентификатор купленной валюты, т. е. оба атрибута имеют общий домен значений, но разный смысл. Поэтому эти атрибуты имеют различные имена – sold (проданная) и (purchased).

Обеспечение ссылочной целостности базы данных. **Ссылочная целостность** (referential integrity) – свойство реляционной базы данных, заключающееся в том, что для каждого значения внешнего ключа дочернего отношения должно существовать соответствующее значение первичного ключа в родительском отношении (т. е. запись в дочернем отношении не может ссылаться на несуществующую запись в родительском отношении). Либо значение внешнего ключа должно быть неопределенным (т. е. не ссылаться на кортеж родительского отношения) [9, с. 71–77].

Ограничения ссылочной целостности – это правила, которые ограничивают выполнение операций вставки (INSERT), обновления (UPDATE) и удаления (DELETE) экземпляров родительской и дочерней сущностей.

Выбор стратегии обеспечения ссылочной целостности определяется бизнес-правилами, действующими в моделируемой предметной области, и типом операции: INSERT, UPDATE или DELETE.

В общем случае (для большинства CASE-средств и СУБД) возможны следующие стратегии обеспечения ссылочной целостности:

1) C – Cascade – разрешить выполнение требуемой операции в **родительском** отношении (затрагивающей первичный ключ), но внести при этом необходимые поправки в дочернем отношении так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Например, при удалении кортежа родительского отношения каскадом удалять все ссылающиеся на него кортежи дочернего отношения. Или при обновлении значения первичного ключа в родительской таблице обновить соответствующее значение внешнего ключа в дочерней таблице. Вставка данных в родительскую или дочернюю таблицу (или выборка данных из любых таблиц) не активирует каскадную операцию. Изменение в родительской таблице полей, не входящих в состав первичного ключа, не активирует каскадную операцию. На одной связи не может быть разрешено несколько различных каскадных операций (например, одновременно и каскадное удаление, и установка значений по умолчанию), кроме каскадного обновления, которое может совмещаться со всеми остальными операциями;

2) Set Null (SN) – разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на неопределенные (NULL) значения (установка пустых внешних ключей). Применяется только в случае, если NULL-значения в соответствующем внешнем ключе разрешены;

3) Set Default (SD) (установить по умолчанию) – разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию при создании столбца внешнего ключа или вычисляемое при выполнении данной операции;

4) Restrict (R) – не разрешать выполнение операции, приводящей к нарушению ссылочной целостности (например, запретить удаление кортежей в родительском отношении при наличии хотя бы одного ссылающегося кортежа);

5) No Action (NOA) – значение по умолчанию, означает, что никакие действия (UPDATE, DELETE) по модификации строк в родительской таблице не должны быть выполнены при наличии связанных строк в дочерней таблице. Позволяет изменять или удалять только те значения в родительской таблице, с которыми не связаны соответствующие значения в столбце внешнего ключа дочерней таблицы;

6) NONE (условного обозначения нет) – не требуется специальное определение ссылочной целостности.

Консистентность базы данных (database consistency) – свойство реляционной базы данных, заключающееся в строгом соблюдении в любой момент времени всех ограничений, заданных неявно реляционной моделью или явно конкретной схемой базы данных. Консистентность контролирует СУБД (контролирует типы данных, ограничения на значения столбцов, ссылочную целостность, корректность выполнения триггеров, корректность запросов и т. д.).

Пример разработки информационной модели. В информационной модели библиотеки используются следующие сущности:

– List_of_events – для хранения информации о мероприятиях, проводимых справочно-библиографическим отделом: номер события, дата, описание;

– Department – для хранения информации об отделах библиотеки: номер отдела, название отдела;

– Library_employee – для хранения данных о сотрудниках библиотеки: табельный номер, фамилия, имя, отчество, дата рождения, должность;

– Student – для хранения информации о студентах, которые пользуются библиотекой: номер читательского билета, фамилия, имя, отчество, год поступления, год окончания, факультет, группа, форма обучения;

– Copy_of_book – для хранения информации об экземплярах книг, зарегистрированных в отделах библиотеки: шифр книги, отметка о списании, отметка о замене;

– Replacement_of_copies – хранит номера актов замены книг;

– Lecturer – для хранения информации о преподавателях-пользователях библиотеки: читательский номер, фамилия, имя, отчество, кафедра, должность;

– Decommissioned_book – хранит информацию о протоколах списания книг: шифр книги, табельный номер списавшего книгу сотрудника библиотеки, причина списания, номер протокола списания;

– Book – для хранения информации о книгах: ISBN, фамилия автора, инициалы автора, название, предметная область, год издания, издательство, количество страниц, цена;

– Order_of_book – заказ преподавателей новой литературы: количество, дата заказа.

Информационная модель БД ИС отображена на рисунке 2.10.

Связь между сущностями List_of_events и Library_employee неидентифицирующая необязательная, т. к. эти сущности могут существовать независимо друг от друга и за определенным мероприятием необязательно может быть закреплен сотрудник. Тип связи 1:M, т. к. за проведение одного мероприятия могут отвечать несколько сотрудников.

Связь между сущностями Department и Library_employee неидентифицирующая обязательная, т. к. каждый сотрудник закреплен за определенным отделом. Тип связи 1:M, т. к. в одном отделе могут работать много сотрудников.

Связь между сущностями Department и Copy_of_book неидентифицирующая обязательная, т. к. каждый экземпляр закреплен за определенным отделом. Тип связи 1:M, т. к. в одном отделе может храниться много экземпляров.

Связь между сущностями Book и Copy_of_book неидентифицирующая обязательная, т. к. каждый экземпляр – это зарегистрированная книга. Тип связи 1:M, т. к. одна книга может быть представлена несколькими экземплярами.

Связь между сущностями Replacement_of_copies и Copy_of_book идентифицирующая, т. к. для замены экземпляров необходима информация о нем. Тип связи 1:M, т. к. замена осуществляется для одного экземпляра.

Связь между сущностями Decommissioned_book и Copy_of_book идентифицирующая, т. к. для списания экземпляров необходима информация о нем. Тип связи 1:1, т. к. списание осуществляется для одного экземпляра.

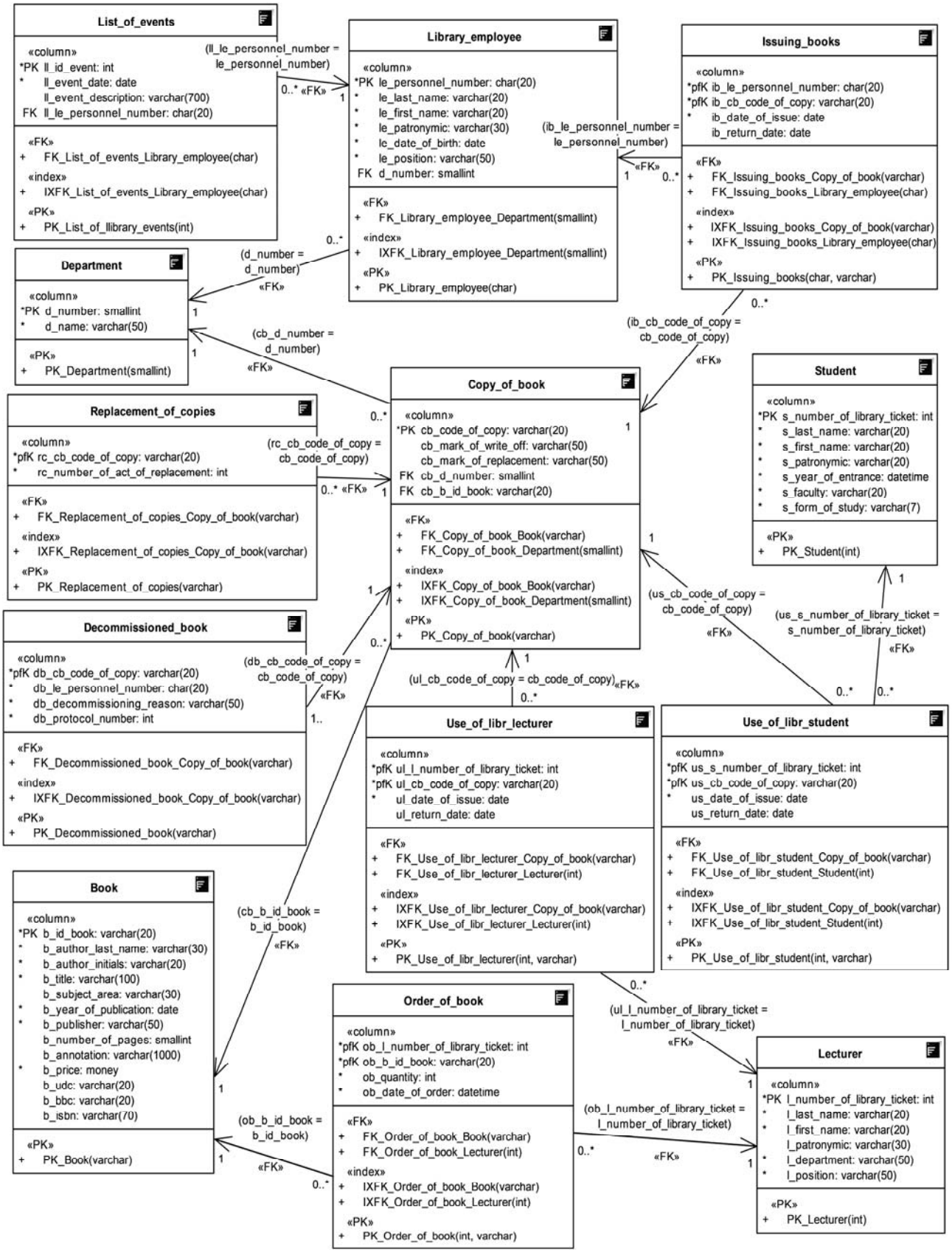


Рисунок 2.10 – Информационная модель

Связь между сущностями Lecturer и Order_of_book идентифицирующая, т. к. для заказа книг необходима информация о заказчике. Тип связи 1:M, т. к. один преподаватель может заказать много книг.

Связь между сущностями Book и Order_of_book идентифицирующая, т. к. для заказа книг необходима информация о заказе. Тип связи 1:M, т. к. одна книга может быть во многих заказах.

Связь между сущностями Lecturer и Copy_of_book – M:M, т. к. один преподаватель может пользоваться многими экземплярами, а один экземпляр может быть у многих преподавателей.

Связь между сущностями Student и Copy_of_book – M:M, т. к. один студент может пользоваться многими экземплярами, а один экземпляр может быть у многих студентов.

Связь между сущностями Library_employee и Copy_of_book – M:M, т. к. один сотрудник библиотеки может пользоваться многими экземплярами, а один экземпляр может быть у многих сотрудников библиотеки.

Для разрешения связей M:M были введены дополнительные зависимые сущности Issuing_books, Use_of_libr_student, Use_of_libr_lecturer.

Ссылочная целостность реализована с учетом следующих ограничений предметной области:

– при изменении информации о каком-либо отделе из таблицы Department в таблицах Library_employee и Copy_of_book информация будет автоматически меняться (каскадное обновление), удалять запрещено;

– при изменении информации о каком-либо сотруднике из таблицы Lecturer в таблице Use_of_libr_lecturer информация будет автоматически изменяться (каскадное обновление);

– в таблице Book разрешено изменение записей (каскадное обновление), удаление данных из этой таблицы запрещается (запрет удаления);

– в таблице Copy_of_book разрешено изменение записей (каскадное обновление), удаление данных из этой таблицы запрещается (запрет удаления);

– в таблице Student при изменении информации о студенте происходит каскадное обновление данных. Разрешается удаление информации только в случае, когда на данную информацию нет ссылок в других связанных таблицах;

– в таблице Lecturer при изменении данных происходит каскадное обновление. Разрешается удаление информации только в случае, если на данную информацию нет ссылок в других связанных таблицах;

– в таблице Order_of_book разрешается обновление.

Задание

Разработать информационную модель выбранной предметной области. Информационная модель должна быть представлена схемой БД, разработанной с использованием методологии UML. На схеме БД должны быть отображены сущности (не менее пяти), связи между сущностями (в том числе не менее одной связи M:M), атрибуты, типы данных атрибутов, NULL-значения атрибутов, мощность связей.

Содержание отчета: тема и цель работы; информационная модель; перечень всех сущностей, входящих в модель, с описанием их назначения и указанием атрибутов каждой сущности; подробное обоснование выбранных типов

связей (идентифицирующих и неидентифицирующих (обязательных и необязательных)) между сущностями, принятых ограничений ссылочной целостности.

Контрольные вопросы

1 Каковы особенности применения нотации UML при проектировании БД?

2 Охарактеризовать основные понятия модели «сущность – связь»: сущности, атрибуты, виды ключей (первичный, составной, естественный, суррогатный, альтернативный, внешний), идентифицирующие и неидентифицирующие (обязательные и необязательные) связи, направленность и мощность связи, связь категоризации, связь многие-ко-многим и ее разрешение.

3 Как можно выбрать тип данных атрибута?

4 Что такое ссылочная целостность?

3 Создание базы данных на основе реляционной СУБД

Цель: изучить основы нормализации данных; изучить основы генерации скрипта SQL; получить навыки установки MS SQL Server и MS SQL Server Management Studio (SSMS).

Теоретические положения

Нормализация данных. **Нормализация** – это метод организации реляционной базы данных с целью сокращения избыточности и устранения аномалий ввода, обновления и удаления [2, 7].

Первая нормальная форма (1НФ). Отношение находится в 1НФ, если все его атрибуты являются атомарными (т. е. имеют единственное значение).

Требования к таблице в 1НФ:

– таблица не должна иметь повторяющихся групп полей;

– в таблице должны отсутствовать повторяющиеся записи. Для выполнения этого условия каждая таблица должна иметь первичный ключ;

– в таблице в 1НФ каждая ячейка на пересечении строки и столбца в таблице должна содержать лишь одно значение, а не список значений.

Вторая нормальная форма (2НФ). Отношение находится в 2НФ, если оно находится в 1НФ, и каждый неключевой атрибут функционально полно (полностью) зависит от первичного ключа (составного). Каждый столбец, не входящий в ключ, должен находиться в зависимости от всего первичного ключа (составного), а не от его части. Группа полей, зависящих от части первичного ключа, вместе с этой частью ключа перемещается в отдельную таблицу.

Третья нормальная форма (3НФ). Отношение находится в 3НФ, если оно находится во 2НФ, и каждый неключевой атрибут нетранзитивно зависит от первичного ключа (т. е. не зависит через другой промежуточный атрибут). В таблице в 3НФ столбцы, не являющиеся ключевыми, должны не только зависеть от всего первичного ключа, но и быть независимыми друг от друга. После

определения поля, при изменении которого меняются другие поля, для каждого такого поля (или группы полей) создается новая таблица, в которую перемещаются данное поле и группа зависящих от него полей.

Рассмотрим пример разработки структуры базы данных и нормализации таблиц. Необходимо автоматизировать процесс формирования накладной (рисунок 3.1), по которой аптеке со склада выдаются товары. Анализ накладной позволил выделить две сущности – «Аптека» и «Выдача_товара». Нужно будет учитывать статистику по разным городам, поэтому из поля «адрес» часть данных (название города) выделена в поле «город» (рисунок 3.2). В таблице «Выдача товара» однозначно идентифицировать каждую запись будет комбинация полей «номер_накладной» и «id_товара».

Требование-накладная на аптечный склад № 1

Дата	Аптека	Адрес
2018-06-27	«Медуница»	г.Н-ск, ул. Ромашковая, 20

Товар	Ед. изм.	Кол-во	Цена, руб.	Стоимость, руб.	Фактически отпущено
Бинт стерильный	шт.	200	0,15	30	30
Бинт нестерильный	шт.	250	0,1	25	25

Итого, руб. 55

Рисунок 3.1 – Образец накладной

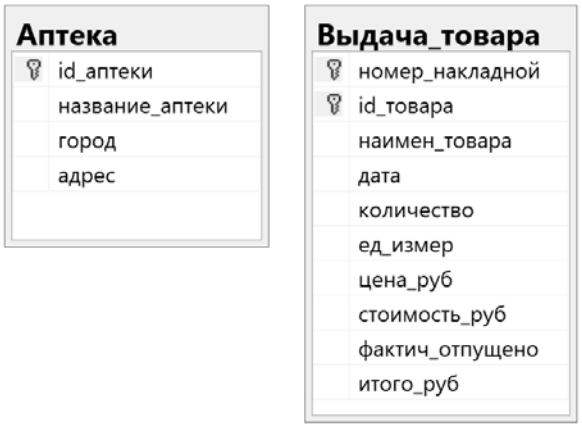


Рисунок 3.2 – База данных в 1НФ

Далее проверяется выполнение требований ко 2НФ и 3НФ. Поле «дата» зависит только от номера накладной. Для приведения к 2НФ выделим указанные поля в отдельную таблицу «Накладная» (рисунок 3.3). Аналогично поля «ед_измер» и «цена_руб» зависят только от поля «наименование_товара», поэтому их выделим в таблицу «Товары».

В таблице «Выдача_товара» есть зависимость полей «стоимость_руб» и «итого» от значения поля «количество», т. е. нарушено требование 3НФ. Поэтому из таблицы «Выдача_товара» нужно удалить указанные поля. Их значения должны вычисляться в результате выполнения представления или хранимой процедуры. Схема БД, приведенной к 3НФ, представлена на рисунке 3.3.

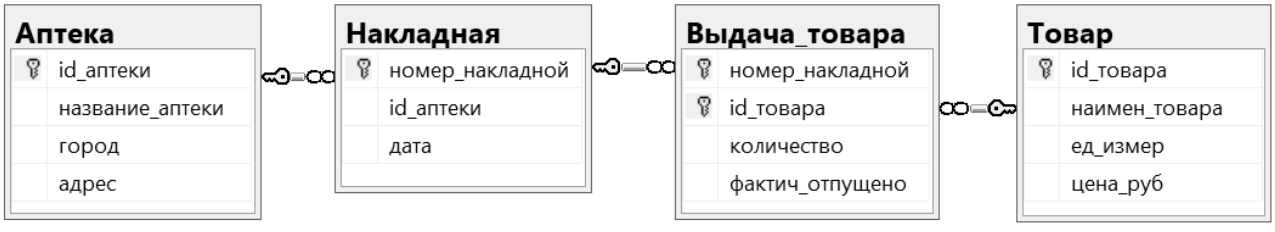


Рисунок 3.3 – Схема базы данных в 3НФ

Microsoft SQL Server – система управления реляционными базами данных (СУБД), использующая язык структурированных запросов Transact-SQL (T-SQL). Установить Microsoft SQL Server Developer Edition можно с официального сайта (https://docs.microsoft.com/ru-ru/sql/tools/).

Для подключения к серверу баз данных и выполнения большинства действий над БД используется среда MS SQL Server Management Studio (SSMS).

Генерация SQL-скрипта для создания схемы базы данных на основе модели в CASE-средствах. В Sparx Enterprise Architect необходимо выбрать пункт меню «Package» → «Database Engineering → Generate Package DDL…». Откроется диалоговое окно «Generate DDL», в котором можно установить ряд параметров. Опции генерации кода можно просмотреть на вкладке «Options». Выбирается способ записи в один файл «Single File» и указывается к нему путь. После этого следует нажать кнопку «Generate» для автоматической генерации SQL-скрипта. Полученный файл можно просмотреть с помощью программы «Блокнот» и использовать в СУБД MS SQL Server для автоматического создания схемы БД.

Для генерации SQL-скрипта БД из ERwin необходимо перейти к вкладке «Physical», выбрав нужный пункт из выпадающего списка на панели инструментов. В меню «Database» выбрать пункт «Choose Database». В открывшемся диалоговом окне в разделе «Target Server» выбрать целевую СУБД (MS SQL Server) и нажать ОК. В меню «Tools» выбрать пункт «Forward Engineer/Schema Generation…». В открывшемся диалоговом окне «Schema Generation» выбрать пункт «Schema» и нажать «Report». В появившемся окне «Generate Schema Report» ввести имя файла .ddl и нажать кнопку «Сохранить».

БД в SQL Server состоит из двух частей:

– файл данных – файл, имеющий расширение .mdf, в котором находятся все основные объекты БД (таблицы, индексы, представления и т. д.);

– файл журнала транзакций – файл, имеющий расширение .ldf, который содержит журнал, где фиксируются все действия с БД (записываются сведения о процессе работы с транзакциями (контроль целостности данных, состояния базы данных до и после выполнения транзакций). Данный файл предназначен для восстановления БД в случае ее выхода из строя.

Для создания файла БД в обозревателе объектов следует нажать правую клавишу мыши на папке «Databases» (Базы данных) и в контекстном меню выбрать пункт «New Database» или «Создать базу данных». В появившемся окне нужно указать имя БД, определить ее владельца (по умолчанию), задать путь доступа к файлам БД .mdf и .ldf. Далее на вкладке создания нового запроса нужно выполнить сгенерированный SQL-скрипт.

Отсоединение и присоединение БД используется для переноса базы БД между различными экземплярами SSMS. Разработанная под управлением сервера S1 БД может быть отсоединена от S1, скопирована на диск другого компьютера и присоединена к серверу S2 на втором компьютере. Для присоединения БД нужно в обозревателе объектов SSMS выбрать «Базы данных» → «Присоединить», для отсоединения БД – выбрать «Базы данных», выделить имя отсоединяемой БД и в контекстном меню выбрать «Задачи» → «Отсоединить».

Создать таблицу в SSMS можно на диаграмме баз данных либо с помощью обозревателя объектов. В обозревателе нужно раскрыть вкладку с созданной БД, раскрыть вкладку «Таблицы» и в появившемся после нажатия правой клавиши мыши контекстном меню выбрать «Создать таблицу». В появившемся окне определения полей новой таблицы указать следующее:

– Column Name – имя поля, начинающееся с буквы и не содержащее различных специальных символов и знаков препинания. Если имя поля содержит пробелы, то оно автоматически заключается в квадратные скобки;

– Data Type – тип данных поля [2, 4];

– Allow NULL – разрешить значения NULL. Если эта опция поля включена, то в случае незаполнения поля в него будет подставлено значение NULL.

После создания таблиц можно перейти к построению схемы БД. Для этого в обозревателе объектов нужно выбрать «Диаграммы баз данных» и в контекстном меню – «Создать диаграмму базы данных» добавить все таблицы в окно схемы БД. Для определения связей следует «ухватиться» за поле главной таблицы и «перетащить» его на поле подчиненной таблицы. При определении связи появляется окно «Create Relationship», в котором задается название связи в поле «Relationship name».

Задание

Продемонстрировать в SSMS базу данных, все таблицы которой должны находиться в третьей нормальной форме.

Содержание отчета: тема и цель работы; схема БД в SSMS; обоснование в письменном виде того, что все таблицы находятся в 3НФ.

Контрольные вопросы

1 Охарактеризовать этапы приведения БД к 3НФ.
2 Как выполнить генерацию SQL-скрипта в CASE-средствах?
3 Указать типы и назначение файлов, которые используются базой данных.
4 Для чего применяется отсоединение и присоединение базы данных?

4 Создание таблиц, связей между ними и индексов средствами SQL

Цель: изучить основы создания таблиц, связей и индексов средствами SQL.

Теоретические положения

Для создания таблицы используется следующая инструкция T-SQL.

```
CREATE TABLE  [ database_name . [ schema_name ] Table_name
( { <column_definition> )
```

Определение каждого столбца таблицы, в синтаксисе команды обозначенное как <column_definition>, имеет следующий формат:

```
{column_name data_type}
[[ NULL | NOT NULL ]  DEFAULT constant_expression
| [IDENTITY [(seed, increment) [NOT FOR REPLICATION]]]
[<column_constraint>]
```

Прежде всего, следует определить имя столбца (column_name), а также тип хранимых в нем данных (data_type).

DEFAULT – определяет значение по умолчанию (constant_expression), используемое, если при вводе строки явно не указано другое значение.

IDENTITY – предписывает системе осуществлять заполнение столбца автоматически. При этом также указать начальное значение (seed) и приращение (increment). В случае, когда указано NOT FOR REPLICATION, этот столбец не будет автоматически заполняться для строк, вставляемых в таблицу в процессе репликации, так что эти строки сохранят свои значения.

Для столбца можно определить ограничения на значения column_constraint при помощи следующих механизмов: первичный ключ (PRIMARY KEY); внешний ключ (FOREIGN KEY); уникальность (UNIQUE); проверочное ограничение на значение столбца (CHECK); значение по умолчанию (DEFAULT), возможность принимать значения NULL (NOT NULL).

Пример создания таблицы с ограничениями на значения столбцов:

```
CREATE  TABLE  Customers (
id  INT  CONSTRAINT  PRIMARY KEY  IDENTITY  NOT NULL,
   age  INT
      CONSTRAINT  DF_Customer_Age  DEFAULT 18
      CONSTRAINT  CK_Customer_Age  CHECK(age >0  AND  age < 100))
```

Синтаксис ограничения внешнего ключа на уровне столбца имеет вид [4]:

```
[ FOREIGN KEY ]
REFERENCES [<схема>.]<таблица> [(<столбец>)]
[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

Предложение ON DELETE определяет, что должно произойти со строками дочерней таблицы (содержащей FK) при удалении соответствующей строки родительской таблицы. Можно задать один из следующих вариантов.

NO ACTION – означает, что не должно быть выполнено никаких действий. Используется по умолчанию. Если удаляется строка родительской таблицы, для которой существует некоторое количество подчиненных строк дочерней таблицы (значение внешнего ключа в этих строках совпадает со значением первичного или уникального ключа удаляемой строки родительской таблицы), то

произойдет нарушение декларативной целостности данных в БД. Внешний ключ здесь будет ссылаться на несуществующую строку родительской таблицы. Для устранения нарушения целостности данных предусматривают некоторые дополнительные действия (разработку триггеров).

CASCADE – приводит к удалению всех соответствующих подчиненных строк дочерней таблицы, т. е. тех строк, которые имеют значение внешнего ключа, совпадающее со значением первичного или уникального ключа, на который ссылается этот внешний ключ, удаляемой родительской записи.

SET NULL – приводит к установке в значение NULL всех столбцов внешнего ключа дочерней таблицы, совпадающих по значению с ключом удаляемой строки родительской таблицы. Используется, если, несмотря на отсутствие соответствующей строки родительской таблицы, строки дочерней таблицы должны оставаться в БД.

SET DEFAULT – устанавливает в значение по умолчанию все столбцы внешнего ключа дочерней таблицы.

Предложение ON UPDATE указывает, что происходит со строками дочерней таблицы, когда изменяется значение любого столбца, входящего в состав ключа родительской таблицы, на который ссылается внешний ключ дочерней таблицы. Варианты те же, что и в случае задания предложения ON DELETE.

Для реализации связей 1:1 и 1:M необходимо, чтобы ссылающаяся (дочерняя) таблица содержала ссылку (внешний ключ) на ссылочную (родительскую) таблицу с соответствующим первичным ключом.

Например, создадим две таблицы, связанные отношением 1:1. Столбец ссылки id_a_link таблицы TableB нужно сделать уникальным внешним ключом. Это гарантирует, что в таблице TableB может быть только одна запись, которая соответствует значению в столбце PRIMARY KEY в таблице TableA.

```
CREATE TABLE TableA (
id_a INT PRIMARY KEY IDENTITY(1,1),
name VARCHAR(255));

CREATE TABLE TableB (
id_b INT PRIMARY KEY IDENTITY(1,1),
name VARCHAR(255),
id_a_link INT UNIQUE,
FOREIGN KEY (id_a_link) REFERENCES TableA (id_a)
ON DELETE CASCADE
ON UPDATE CASCADE);
```

В случае связи 1:M внешний ключ должен быть неуникальным.
Формат команды CREATE INDEX на T-SQL имеет вид [4]:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name ON Table (column [...n])
```

UNIQUE – при указании этого ключевого слова будет создан уникальный индекс. При создании такого индекса сервер выполняет предварительную проверку столбца на уникальность значений. Если в столбце есть хотя бы два одинаковых значения, индекс не создается и сервер выдает сообщение об ошибке. В индексируемой столбце также желательно запретить хранение значений NULL, чтобы избежать проблем, связанных с уникальностью значений. Сервер не разрешает выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующихся значений. В одной таблице может быть один уникальный кластерный индекс и множество уникальных некластерных индексов.

CLUSTERED – создаваемый индекс будет кластерным, т. е. физически данные будут располагаться в порядке, определяемом этим индексом. Кластерным может быть только один индекс в таблице.

NONCLUSTERED – создаваемый индекс будет некластерным. В таблице можно определить до 249 некластерных индексов. Однако в большинстве случаев следует ограничиться 4-5 индексами.

index_name – имя индекса, по которому он будет распознаваться командами Transact-SQL. Имя индекса должно быть уникальным в пределах таблицы.

table (column [...n]) – имя таблицы, в которой содержатся одна или несколько индексируемых колонок. В скобках указываются имена колонок, на основе которых будет построен индекс. Не допускается построение индекса на основе колонок с типом данных TEXT, NTEXT, IMAGE или BIT.

Задание

Необходимо создать средствами T-SQL три таблицы, подобные уже имеющимся в базе данных, но содержащие все виды ограничений на значения столбцов, определить связи между ними.

Для таблиц, созданных в лабораторной работе № 2, следует разработать пять индексов, при этом нужно обосновать выбор индексируемых столбцов.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 С помощью каких команд T-SQL можно создать, изменить или удалить таблицу?

2 Для чего предназначены предложения FOREIGN KEY и REFERENCES?

3 Какие ограничения на значения столбцов можно накладывать?

4 Что такое кластерный, некластерный и уникальный индексы?

5 Перечислить общие рекомендации при планировании стратегии индексирования.

6 Как создаются связи 1:1, 1:M, M:M?

5 SQL-скрипт (сценарий) создания и заполнения базы данных

Цель: приобрести навыки создания sql-скриптов заполнения БД.

Теоретические положения

Сценарий (скрипт) – последовательность операторов T-SQL. Для подготовки сценария, отладки и выполнения используется SSMS.

Автоматическая генерация скриптов из обозревателя объектов.

Для таблицы можно сгенерировать скрипты для создания таблицы, удаления таблицы, выборки данных из таблицы, скрипты для добавления новых данных в таблицу, а также для изменения и удаления существующих записей.

Для генерации скрипта таблицы текущей БД из контекстного меню выбирается команда «Создать сценарий для таблицы» → «Используя CREATE».

Для генерации скрипта БД после выделения имени БД из контекстного меню выбирается команда «Задачи» → «Сформировать скрипты» → «Создать скрипт для всей базы данных и всех ее объектов» → «Указание порядка сохранения скриптов: Открыть в новом окне запроса» → «Дополнительные параметры создания скрипта: Типы данных для внесения в скрипт – Схема и данные».

При запуске SSMS и подключении к SQL серверу по умолчанию выполняется команда Use Master, т. е. работа идет с системной БД Master. Для выбора иной БД следует выполнить команду Use Имя_Базы_Данных. Перед запуском на исполнение сгенерированного скрипта БД аналогичным образом в первой строке скрипта нужно указывать Use Имя_Базы_Данных.

Для создания скрипта для административных операций (например, резервное копирование базы данных, создание учетной записи и т. д.) можно воспользоваться контекстным меню «Задачи» → «Создать резервную копию…» → «Скрипт», что позволит автоматически создать скрипт, в который будут подставлены введенные в полях формы на экране значения.

Задание

Необходимо, используя команды INSERT, внести в базу данных не менее 100 записей. Пример команды INSERT:

INSERT INTO Table1(id, product, date) VALUES (3, ’Коробка’, 2020-12-26)

Для заполненной БД сгенерировать скрипт заполнения.

Содержание отчета: тема и цель работы; скрипт заполнения БД.

Контрольные вопросы

1 Что такое скрипт (сценарий) и для решения каких задач он используется?
2 Какие виды скриптов существуют?

6 Язык SQL. Добавление, изменение и удаление данных в таблицах средствами SQL

Цель: научиться добавлять, изменять и удалять данные в таблицах с помощью команд T-SQL.

Теоретические положения

Для вставки данных в таблицу средствами T-SQL используются операторы INSERT, UPDATE, DELETE, синтаксис которых рассмотрен в [4, 5, 7, 9–12].

В таблице 6.1 приведены примеры **оператора INSERT** для вставки данных в таблицу, SQL-код определения которой приведен ниже:

```
CREATE TABLE Book
(
    b_id_book int PRIMARY KEY IDENTITY (1, 1),
    b_title varchar (100) NULL,
    b_publisher varchar (50) NOT NULL DEFAULT 'ИНФРА-М'
)
```

Таблица 6.1 – Примеры инструкций INSERT

Задача	SQL-код решения
Добавление в таблицу строки с указанием имен столбцов	INSERT INTO Book (b_title, b_publisher) VALUES ('Война и мир', 'BHV');
Добавление в таблицу строки без указания имен столбцов	INSERT INTO Book VALUES ('Война и мир', 'BHV');
Добавление в таблицу строки с подстановкой значения, заданного ограничением DEFAULT	INSERT INTO Book VALUES ('Анна Каренина', DEFAULT);
Добавление в таблицу строки с указанием измененного порядка следования столбцов	INSERT INTO Book (b_publisher, b_title) VALUES ('BHV', 'Война и мир');
Добавление в таблицу строки в том случае, если все столбцы имеют значения по умолчанию (второй столбец в таблице Book тоже имеет значение по умолчанию – NULL)	INSERT INTO Book DEFAULT VALUES;
Добавление в таблицу нескольких строк	INSERT INTO Book VALUES ('Исповедь', DEFAULT), ('Воскресение', DEFAULT);
Добавление в таблицу строки с заданным значением в столбце идентификаторов	SET IDENTITY_INSERT Book ON; INSERT INTO Book (b_id_book, b_title, b_publisher) VALUES (17285, 'Детство', 'BHV')
Добавление строк, значения которых определяются на основе подзапроса	INSERT INTO Copy_Book VALUES ('Исповедь', (SELECT b_publisher FROM Book WHERE b_id_book = 1)), ('Воскресение', (SELECT b_publisher FROM Book WHERE b_id_book = 2));

Окончание таблицы 6.1

Задача	SQL-код решения
Добавление в копию таблицы строк, отобранных в подзапросе на основе некоторого условия	INSERT INTO Copy_Book SELECT * FROM Book WHERE b_title = 'Воскресение';
Добавление данных во вновь создаваемую таблицу Book_copy – с помощью инструкции SELECT INTO, которая является вариацией простой инструкции SELECT	SELECT b_id_book, b_publisher, b_title INTO Book_copy FROM Book WHERE b_id_book = 1

Для создания набора вставляемых данных можно использовать инструкцию выполнения вместе с хранимой процедурой или пакетом SQL.

Если столбец имеет свойство IDENTITY (столбец идентификаторов, счетчик), при вставке строки имя этого столбца и значение поля для этого столбца в команде INSERT не указывают. Для такого столбца сервер автоматически вычисляет новое значение. Оператор SET IDENTITY_INSERT < имя таблицы > { ON | OFF } отключает (ON) или включает (OFF) автоинкремент.

В таблице 6.2 приведены примеры использования **оператора UPDATE**.

Операция обновления столбца со свойством IDENTITY представляет собой комбинацию инструкции DELETE с удалением ненужного значения из ячейки столбца идентификаторов и инструкции INSERT со вставкой требуемого значения в ячейку столбца идентификаторов.

Таблица 6.2 – Примеры инструкций UPDATE

Задача	SQL-код решения
Обновление в таблице Book_copy всех значений столбца b_publisher на основе выражения для определения новых значений	UPDATE Book_copy SET b_publisher = CONCAT('издательство ', b_publisher)
Обновление в таблице Book значений столбца с указанием условия отбора строк для обновления	UPDATE Book SET b_publisher = 'AZ' WHERE b_publisher = 'Лира';
Обновление в таблице Book значений нескольких столбцов с указанием условия отбора строк для обновления	UPDATE Book SET b_publisher = 'AZ', b_title = 'Букварь' WHERE b_publisher = 'Лира';
Обновление в столбце b_publisher таблицы Book значения 'ИНФРА-М' на значение 'Y' во всех строках, где b_publisher='ИНФРА-М'	UPDATE Book SET Book.b_publisher = 'Y' FROM (SELECT * FROM Book WHERE b_publisher='ИНФРА-М') AS BS WHERE Book.b_id_book = BS.b_id_book

В таблице 6.3 приведены примеры использования **оператора DELETE**.

Примечание – Во избежание нежелательных последствий с помощью инструкции SELECT INTO создается копия таблицы «Book» и выполняется работа уже с копией:

```
SELECT * INTO Book_copy FROM Book
DELETE FROM Book_copy /* удаление всех строк из таблицы Book_copy */
```

```
DROP  TABLE  Book_copy     /* удаление таблицы Book_copy  */
```

Таблица 6.3 – Примеры инструкций DELETE

Задача	SQL-код решения
Удаление из таблицы Book строк с указанием условия отбора удаляемых строк	DELETE FROM Book_copy WHERE b_publisher LIKE 'A%';
Удаление первых двух строк таблицы Book_copy. Подзапрос (инструкция SELECT в круглых скобках) возвращает базовый набор строк для инструкции DELETE. Результату этого подзапроса присваивается псевдоним bc, а директива WHERE задает параметры сравнения строк из bc с базовой таблицей. Затем директива DELETE автоматически удаляет все совпавшие строки	SELECT * INTO Book_copy FROM Book DELETE Book_copy FROM (SELECT TOP 2 * FROM Book_copy) AS bc WHERE Book_copy.b_id_book = bc.b_id_book
Удаление тех строк из таблицы Book_copy, для которых нет соответствующих строк в таблице Book, с использованием стандартного синтаксиса оператора DELETE, при этом для определения удаляемых строк используется подзапрос	DELETE FROM Book_copy WHERE b_publisher = 'Лира' AND b_id_book NOT IN (SELECT b_id_book FROM Book);
Удаление тех строк из таблицы Book_copy, для которых нет соответствующих строк в таблице Book, с использованием дополнительного предложения FROM, которое вводит источник табличного типа, конкретизирующий данные, удаляемые из таблицы в первом предложении FROM	DELETE FROM Book_copy FROM Book_copy AS bc LEFT JOIN Book ON bc.b_id_book = Book.b_id_book WHERE bc.b_publisher = 'Лира' AND Book.b_id_book IS NULL;

Инструкция TRUNCATE TABLE ИмяЦелевойТаблицы удаляет все строки в целевой таблице, не записывая в журнал транзакций удаление отдельных строк, за счет чего выполняется быстрее и требует меньших ресурсов системы.

В таблице 6.4 представлены допустимые подстановочные символы для шаблонов LIKE (см. таблицу 6.3), их значения и примеры использования.

Таблица 6.4 – Подстановочные символы, используемые в шаблонах LIKE

Подстановочный знак	Значение	Пример	Результат
%	Символ-шаблон, заменяющий любую последовательность символов	WHERE [title] LIKE 's%'	Строка, начинающаяся с s: Samantha или sven
_ (подчеркивание)	Символ-шаблон, заменяющий любой одиночный символ	WHERE [titleofcourtesy] LIKE 'm_.'	Ms. Mr.
[<список символов>]	Один символ из списка	WHERE [firstname] LIKE '[sp]%'	Строка, в которой первый символ s или p: Sara или Paul
[<диапазон символов>]	Один символ из диапазона. При этом можно перечислить сразу несколько диапазонов (например, [0-9a-z])	WHERE [freight] LIKE '6[5-7]%'	Строка, в которой второй символ – цифра 5, 6 или 7: 65,83 или 677,54

Окончание таблицы 6.4

Подстановочный знак	Значение	Пример	Результат
[^<список или диапазон символов>]	В сочетании с квадратными скобками исключает из поискового образца символы из списка или диапазона	WHERE [shipaddress] LIKE '[^0-9]%'	Строка, в которой первый символ не цифра
ESCAPE ' '	Для поиска символа, который является подстановочным знаком, его указывают после Escape-символа с помощью ключевого слова ESCAPE	WHERE col1 LIKE '!_%' ESCAPE '!'	Поиск строки, которая начинается со знака подчеркивания (_), используя в качестве Escape-символа (!)
'ymd'	Для поиска даты используется форма без разделителей, которая не зависит от языка входа в систему для всех типов данных даты и времени	WHERE [birthdate] = '19581208'	1958-12-08 00:00:00.000

Агрегатные функции выполняют вычисления над значениями в наборе строк. Все агрегатные функции за исключением COUNT(*) игнорируют значения NULL (таблица 6.5).

Таблица 6.5 – Агрегатные функции

Синтаксис	Назначение
AVG()	Вычисляет среднее значение для указанного столбца
SUM()	Суммирует все значения в указанном столбце
MIN()	Находит минимальное значение в указанном столбце
MAX()	Находит максимальное значение в указанном столбце
COUNT()	Подсчитывает количество строк с непустым (не NULL) значением указанного столбца
COUNT(*)	Подсчитывает общее количество строк, удовлетворяющих условию, включая пустые (NULL)

Выражения в функциях AVG и SUM должно представлять числовое значение.

Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату.

Агрегатные функции могут использоваться только в списке предложения SELECT и в составе предложения HAVING. Во всех других случаях это недопустимо.

Задание

Необходимо для разрабатываемой БД написать по три команды INSERT, UPDATE, DELETE для каждой таблицы с использованием различных функций и предикатов в условии отбора.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Объяснить синтаксис команд INSERT, SELECT INTO, UPDATE, DELETE и указать ограничения для данных команд.
2 Привести примеры использования команд INSERT, UPDATE, DELETE.
3 Указать назначение и ограничения инструкции TRUNCATE TABLE.

7 Язык SQL. Работа с представлениями

Цель: научиться создавать представления в СУБД MS SQL Server.

Теоретические положения

Представление – это именованный запрос на выборку, сохраненный в БД, который выглядит и работает как таблица, при обращении по имени создает виртуальную таблицу, наполняя ее актуальными данными из БД, с которой можно работать так же, как с реально существующей на диске таблицей. Физически представление реализовано в виде SQL-запроса, на основе которого производится выборка данных из одной или нескольких таблиц или представлений. Представление часто применяется для ограничения доступа пользователей к конфиденциальным данным в таблице [3, 4, 11].

Для создания представлений средствами Transact-SQL используется следующая конструкция:

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] )
[ WITH { ENCRYPTION | SCHEMABINDING } ]
AS
select_statement
[WITH CHECK OPTION]
```

Рассмотрим составляющие данной конструкции.

view_name – имя представления. При указании имени необходимо придерживаться тех же правил и ограничений, что и при создании таблицы.

column – имя столбца, которое будет использоваться в представлении (длина имени до 128 символов). Имена столбцов перечисляются через запятую в соответствии с их порядком в представлении. Имена столбцов можно указывать в команде SELECT, определяющей представление.

WITH ENCRYPTION – данный параметр предписывает серверу шифровать код SQL-запроса. Это гарантирует, что пользователи не смогут просмотреть код запроса и использовать его. Если при определении представления

необходимо скрыть имена исходных таблиц и колонок, а также алгоритм объединения данных, то следует использовать эту опцию.

WITH SCHEMABINDING – привязывает представление к схеме базовой таблицы. Нельзя будет изменить описание базовых таблиц, если это повлияет на представление. Сначала нужно будет изменить или удалить само представление для сброса зависимостей от таблицы, которую требуется изменить. При указании этого атрибута базовые таблицы в представлении должны быть записаны в виде <схема>.<таблица>.

select_statement – код запроса SELECT, выполняющий выборку, объединение и фильтрацию строк из исходных таблиц и представлений. Можно использовать команду SELECT любой сложности со следующими ограничениями:

1) нельзя создавать новую таблицу на основе результатов, полученных в ходе выполнения запроса, т. е. запрещается использование параметра INTO;

2) нельзя проводить выборку данных из временных таблиц, т. е. нельзя использовать имена таблиц, начинающихся на # или ##;

3) в представление нельзя включать предложение ORDER BY, если только в списке выбора инструкции SELECT нет также предложения TOP.

Предложение WITH CHECK OPTION применяется только к обновлениям, выполненным через представление. Оно неприменимо к обновлениям, выполненным непосредственно в базовых таблицах представления. Если имеется представление с ограничением фильтра в предложении WHERE инструкции SELECT, а затем с помощью представления были изменены строки таблицы, то можно изменить некоторое значение так, что задействованная строка уже не будет удовлетворять фильтру предложения WHERE. Возможно даже обновление строк, которые выходят за пределы области фильтра. Предложение WITH CHECK OPTION препятствует подобному исчезновению строк при обновлении через представление, а также ограничивает модификации только строками, которые удовлетворяют критериям фильтра.

Чтобы выполнить представление, т. е. получить данные в виде виртуальной таблицы, необходимо выполнить запрос SELECT к представлению так же, как и к обычной таблице: SELECT * FROM view_name.

Для удаления представления используется команда T-SQL DROP VIEW{view […n]}. За один раз можно удалить несколько представлений.

Обновляемые представления не содержат функции агрегирования или вычисляемых столбцов. Кроме того, представление, указанное в предложении FROM инструкции DELETE, должно содержать ровно одну таблицу (имеется в виду предложение FROM, используемое для создания представления, а не директива FROM в инструкции DELETE) [4].

В качестве примера приведено представление, предоставляющее информацию об экземплярах книг, которые были изданы за последние пять лет.

```
CREATE VIEW Get_full_info_copy_of_book
AS
SELECT        /*Указываем, какие поля будут выбраны*/
Copy_of_book.cb_code_of_copy,
```

```
Book.b_author_last_name, Book.b_title, Book.b_year_of_publication, Book.b_publisher,
Copy_of_book.cb_d_number,
Copy_of_book.cb_mark_of_write_off, Copy_of_book.cb_mark_of_replacement
FROM /*Указываем таблицу и связанные с ней таблицы, из которых выбираются связанные данные*/
Book
   INNER JOIN Copy_of_book
   ON Book.b_id_book = Copy_of_book.cb_b_id_book
WHERE YEAR(Book.b_year_of_publication) BETWEEN YEAR(GETDATE()-5) AND YEAR(GETDATE())
/* GETDATE() возвращает текущую дату, YEAR(<дата>) – год <даты> */
```

Задание

В разрабатываемой базе данных необходимо реализовать 10 представлений с использованием стандартных функций T-SQL. При этом должно быть использовано не менее 10 различных функций.

Содержание отчета: тема и цель работы; SQL-код 10 представлений.

Контрольные вопросы

1 Что такое представление и в каких случаях целесообразно его использовать? Перечислить способы создания представлений.

2 Какие виды представлений различают? Что такое обновляемые представления? Что такое кеширующие (материализованные, индексированные) представления?

3 Перечислить операторы SQL, с помощью которых представления создаются, удаляются и изменяются.

4 Перечислить ограничения при создании представлений.

8 Язык SQL. Создание хранимых процедур и пользовательских функций

Цель: научиться создавать хранимые процедуры для вставки, удаления, изменения данных и пользовательские функции с использованием T-SQL.

Теоретические положения

Хранимая процедура – это скомпилированный набор SQL-предложений, сохраненный на сервере баз данных как именованный объект и выполняющийся как единый фрагмент кода. Хранимые процедуры могут принимать и возвращать параметры. При этом клиент осуществляет только вызов процедуры по

ее имени, затем сервер базы данных выполняет блок команд, составляющих тело вызванной процедуры, и возвращает клиенту результат [3, 4, 11].

Хранимые процедуры обычно используются для поддержки ссылочной целостности данных и реализации бизнес-правил. В последнем случае повышается скорость разработки приложений, поскольку, если бизнес-правила изменяются, можно изменить только текст хранимой процедуры, не изменяя клиентские приложения. По сравнению с обычными SQL-запросами, посылаемыми из клиентского приложения, они требуют меньше времени для подготовки к выполнению, поскольку скомпилированы и сохранены.

Хранимые процедуры имеют следующее определение:

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [;number]
[ {@parameter data_type} [= default] [ OUT | OUTPUT | [READONLY]] [,...n]
AS sql_statement [...n]
```

Рассмотрим составляющие данной конструкции.

procedure_name – имя создаваемой процедуры. Используя префиксы sp_, # и ##, можно определить создаваемую процедуру как системную или временную (локальную или глобальную). Имя процедуры должно характеризовать действие, выполняемое ею, и записываться в формате <глагол_объект>.

number – параметр определяет идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур.

@parameter – определяет имя параметра, который будет использоваться создаваемой хранимой процедурой для передачи входных или выходных данных. Параметры, определяемые при создании хранимой процедуры, являются локальными переменными, поэтому несколько хранимых процедур могут иметь абсолютно идентичные параметры. Можно объявить один или несколько параметров, максимум – 2100.

data_type – определяет, к какому типу данных должны относиться значения параметра описываемой процедуры.

default – позволяет определить для параметра значение по умолчанию, которое хранимая процедура будет использовать в случае, если при ее вызове указанный параметр был опущен.

OUT | OUTPUT – определяет указанный параметр как выходной. Его значение может быть использовано вызвавшей программой.

READONLY – означает, что значение параметра не может быть изменено внутри хранимой процедуры.

AS – ключевое слово, определяющее начало кода хранимой процедуры. После этого ключевого слова следуют команды T-SQL, которые и составляют непосредственно тело процедуры (sql_statement). Здесь можно использовать любые команды, включая вызов других хранимых процедур, за исключением команд, начинающихся с ключевого слова CREATE.

Более подробно вопросы создания хранимых процедур различных видов рассмотрены в [4, 11].

Далее приведен пример хранимой процедуры, возвращающей количество экземпляров какой-либо книги.

```
/* проверяется, существует ли хранимая процедура Count_number_of_copies, и при необходимости она удаляется */
DROP PROCEDURE IF EXISTS Count_number_of_copies;
GO

/* проверяется, существует ли временная таблица Temp1, и при необходимости она удаляется */
DROP TABLE IF EXISTS TEMP1;
GO

CREATE PROCEDURE Count_number_of_copies
@b_id_book varchar(20)      /*Объявляем входную переменную*/
@number int  OUTPUT         /*Объявляем выходную переменную*/
AS
/* Следующая конструкция проверяет, существуют ли записи в таблице «Book» с заданным b_id_book*/
IF NOT EXISTS (SELECT * FROM Book WHERE b_id_book = @b_id_book)
RETURN 0      /* Вызывает конец процедуры Count_number_of_copies */
SELECT Copy_of_book.cb_b_id_book
INTO  TEMP1      /*Сохраняет выбранные поля во временной таблице Temp1*/
FROM Copy_of_book
WHERE cb_b_id_book = @b_id_book
SELECT @number = COUNT(cb_b_id_book)      /* COUNT подсчитывает количество неповторяющихся записей поля b_id_book */
FROM TEMP1
```

Пример хранимой процедуры на удаление из таблицы «Student». Допустимо, если в таблице «Use_of_libr_student» нет ссылающихся записей.

```
CREATE  PROCEDURE   Delete_student
@num int            /* Объявляем входные переменные */
AS                  /* Проверяем, есть ли ссылающиеся записи, если записей нет, разрешается удаление */
IF NOT EXISTS  (SELECT * FROM Use_of_libr_student
                    WHERE  us_s_number_of_library_ticket =@num)
DELETE              /* Оператор удаления */
FROM Student        /* Имя таблицы, откуда нужно удалить */
WHERE               /* Условие удаления – удаляем строку, для которой значе-ние поля us_s_number_of_library_ticket совпадает с нужным */
us_s_number_of_library_ticket = @num
```

Пример хранимой процедуры на вставку в таблицу «Order_of_book». Разрешена, если в таблицах «Book» и «Lecturer» есть записи, на которые будет ссылаться новая запись.

```
PROCEDURE  Create_new_order
@quantity int,
@order_date datetime,
@number int,
@b_id_book varchar(20)
AS   /* Проверяем, есть ли запись в таблице «Order_of_book» с такими же значениями ключевых полей, как у новой записи */
IF EXISTS (SELECT * FROM Order_of_book
            WHERE ob_b_id_book = @b_id_book
            AND ob_l_number_of_library_ticket = @number)
RETURN 0          /* Если есть, завершаем выполнение процедуры */
IF EXISTS (SELECT * FROM Lecturer
            WHERE l_number_of_library_ticket = @number)
/*Проверяем, есть ли в «Lecturer» соответствующая запись */
IF EXISTS (SELECT * FROM Book
             WHERE b_id_book = @b_id_book)
/* Проверяем, есть ли в «Book» соответствующая запись */
INSERT INTO   Order_of_book          /* Указываем таблицу, в которую вставляем запись  */
VALUES  (@quantity, @order_date, @number, @b_id_book)    /* Указываем, какие значения  */
```

Пример хранимой процедуры на обновление таблицы «Student» (изменение фамилии студента).

```
CREATE PROCEDURE Update_student
@number int,          /* Объявляем входные переменные */
@lname varchar(20)
AS
IF EXISTS (SELECT * FROM Student        /* Проверяем, существуют ли студенты */
   WHERE s_number_of_library_ticket = @number)       /* номер читательского билета которых равен искомому */
UPDATE Student          /* Если такие есть, обновляем таблицу «Student» */
SET s_last_name=@lname    /* полю фамилия присваиваем новое значение */
    WHERE s_number_of_library_ticket = @number      /* если номер читательского билета записи равен искомому */
```

Пользовательские функции (*User Defined Functions, UDF*). Пользовательские функции – это процедуры T-SQL, которые инкапсулируют повторно ис-

пользуемый код T-SQL, могут принимать параметры и возвращать либо скалярные значения, либо таблицы [4].

В отличие от хранимых процедур, пользовательские функции встроены в инструкции T-SQL, исполняются как часть команды T-SQL и не могут выполняться с помощью команды EXECUTE. Пользовательские функции имеют доступ к данным SQL Server, но не могут выполнять инструкции DDL или изменять любые данные в постоянных таблицах с помощью инструкций DML.

Функции, определяемые пользователем, могут быть скалярными или табличными. Скалярная функция возвращает скалярное значение (число). Это означает, что в предложении RETURNS скалярной функции задается один из стандартных типов данных. Функции являются табличными, если предложение RETURNS возвращает набор строк.

Виды пользовательских функций:

– *скалярная функция* – возвращает вызывающей стороне одно значение;

– *функция с табличным значением* – возвращает таблицу и может появляться в предложении FROM запроса T-SQL. Функция с табличным значением, состоящая из одной строки кода, называется *встроенной пользовательской функцией с табличным значением*. Функция с табличным значением, состоящая из нескольких строк кода, называется *многооператорной возвращающей табличное значение пользовательской функцией*.

Пользовательские скалярные функции могут появляться в любом месте запроса, где может появляться выражение, возвращающее одно значение (например, в списке столбца SELECT). Весь код внутри пользовательской скалярной функции должен быть заключен в блок BEGIN/END.

В SSMS, если щелкнуть правой кнопкой мыши в окне запроса, выбрать команду Insert Snippet (Вставить фрагмент) и вставить фрагмент для скалярной функции, то можно увидеть следующие выходные данные:

```
CREATE FUNCTION [dbo].[FunctionName]
(  @param1 int,
   @param2 int  )
RETURNS INT
AS
BEGIN
   RETURN @param1 + @param2
END
```

На основе данного фрагмента можно, например, создать простую скалярную функцию для вычисления стоимости как цены, умноженной на количество, в таблице Sales.

```
IF OBJECT_ID('fn_extension', 'FN') IS NOT NULL
DROP FUNCTION fn_extension
GO
CREATE FUNCTION fn_extension
```

```
( @unitprice AS MONEY,
  @qty AS INT )
RETURNS MONEY
AS
BEGIN
RETURN @unitprice * @qty
END;
GO
```

Для вызова функции можно просто вызвать ее внутри запроса T-SQL, например в инструкции SELECT:

```
SELECT Orderid, unitprice, qty, fn_extension(unitprice, qty) AS extension
FROM Book;
```

Встроенная пользовательская функция с табличным значением содержит одну инструкцию SELECT, которая возвращает таблицу.

Рассмотрим пример функции, возвращающей только те строки таблицы Book, у которых количество находится в промежутке между двумя величинами.

```
CREATE FUNCTION fn_FilteredExtension
(  @lowqty AS SMALLINT,
   @highqty AS SMALLINT )
RETURNS TABLE AS RETURN
( SELECT orderid, unitprice, qty   FROM Sales.OrderDetails
  WHERE qty BETWEEN @lowqty AND @highqty );
```

Пример вызова данной функции:

```
SELECT orderid, unitprice, qty
FROM fn_FilteredExtension (3,5);
```

Поскольку встроенная функция с табличным значением не выполняет никаких других операций, оптимизатор обрабатывает ее точно так же, как представление. Можно даже применять к ней инструкции INSERT, UPDATE и DELETE, как для представления.

Задание

Необходимо реализовать 10 хранимых процедур, в том числе для вставки, удаления, изменения данных с использованием стандартных функций SQL.

Необходимо реализовать также одну пользовательскую функцию.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Что такое хранимая процедура? Для чего используется?

2 Какие виды параметров могут использоваться в процедуре?

3 Как производится средствами T-SQL создание, модификация и удаление хранимых процедур? Как создаются хранимые процедуры на вставку, изменение и удаление данных?

4 Описать управление процессом компиляции хранимой процедуры.

5 Как создать и вызвать пользовательскую функцию?

9 Язык SQL. Работа с триггерами

Цель: научиться создавать триггеры в MS SQL Server Management Studio.

Теоретические положения

Триггер – это специальный тип хранимых процедур, который запускается автоматически при выполнении тех или иных действий с данными таблицы. Каждый триггер привязывается к конкретной таблице.

Существует три типа триггеров в зависимости от команд, на которые они реагируют: триггеры на вставку, на обновление и на удаление. Для одной таблицы допускается создание нескольких однотипных триггеров. Более подробно триггеры описаны в конспекте лекций и в [4].

Для создания триггера используется следующая инструкция T-SQL:

```
CREATE TRIGGER Trigger_name
ON Table_name
{FOR {[DELETE] [,] [INSERT] [,] [UPDATE]}
[WITH APPEND]
AS
sql_statement [...n] }
```

Trigger_name – задает имя триггера, с помощью которого он будет распознаваться хранимыми процедурами и командами Transact SQL. Имя триггера должно быть уникальным в пределах БД.

Table_name – имя таблицы БД, к которой будет привязан триггер.

[DELETE] [,] [INSERT] [,] [UPDATE] – эта конструкция определяет, на какие автоматы будет реагировать триггер. При создании триггера должно быть указано хотя бы одно из этих ключевых слов, и допускается создание триггера, реагирующего на две или три команды.

WITH APPEND – указание этого ключевого слова требуется для обеспечения совместимости с более ранними версиями SQL Server.

sql_statement – определяет набор команд, которые будут выполняться при запуске триггера.

Для изменения триггера используется команда ALTER TRIGGER.

Далее приведен пример триггера, который будет запрещать удаление записей таблицы «Issuing_books», если текущий пользователь не владелец базы данных и если поле «дата выдачи» содержит какое-либо значение.

```
CREATE TRIGGER ondelete_iss    /*Объявляем имя триггера*/
ON Issuing_books                    /* имя таблицы, с которой связан триггер*/
FOR DELETE  /*Указываем операцию, на которую будет срабатывать триггер (здесь на  удаление)*/
AS
IF ( SELECT count(*)          /*проверяет*/
FROM Issuing_books            /*записи из таблицы «Issuing_books»*/
WHERE Issuing_books.date_of_issue IS NOT NULL) > 0  /*условие проверяет наличие записи в поле «date_of_issue». Если count возвращает значение, отличное от нуля (т. е. запись есть), то первое условие IF не выполнено*/
AND (CURRENT_USER <> 'dbo')  /*вызывается функция определения имени текущего пользователя и проверяется, владелец ли он*/
BEGIN
PRINT 'у вас нет прав на удаление этой записи'  /*выдача сообщения о неудаче операции*/
ROLLBACK TRANSACTION      /*откат (отмена) транзакции*/
END
```

Задание

В разрабатываемой БД необходимо реализовать 5 триггеров.

Содержание отчета: тема и цель работы; SQL-код выполнения задания.

Контрольные вопросы

1 Что такое триггер? Какие типы триггеров различают?

2 Как создать триггер?

3 С помощью каких команд T-SQL можно изменить или удалить триггер?

10 Назначение прав доступа пользователям к объектам базы данных средствами SQL

Цель: изучить основы управления правами доступа к объектам базы данных в СУБД MS SQL Server.

Теоретические положения

Вопросы управления правами доступа к объектам БД подробно описаны в конспекте лекций и в [4, 7]. Здесь же рассмотрен SQL-код основных инструкций.

Для предоставления прав доступа используется инструкция GRANT:

```
GRANT
{ ALL | permissions [..n]}
{ [ (column [..n] )] ON { table | view }
| ON { table | view} [ (column[..n]) ]
| ON {stored_procedure}
TO security_account
[ WITH GRANT OPTION ]   [ AS {group | role} ]
```

Для запрещения пользователям доступа к объектам БД используется инструкция DENY:

```
DENY
{ ALL | permissions [...n]}
{ [ (column [..n] )] ON { table | view }
| ON { table | view }  [ (column[..n]) ]
| ON { stored_procedure }
TO security_account [...n]  [ CASCADE ]
```

Неявное отклонение подобно запрещению доступа с тем отличием, что оно действует только на том уровне, на котором определено. Если пользователю на определенном уровне неявно отклонен доступ, он все же может получить его на другом уровне иерархии через членство в роли, имеющей право просмотра. По умолчанию доступ пользователя к данным неявно отклонен.

Для неявного отклонения доступа к объектам БД используется REVOKE:

```
REVOKE
{ ALL | permissions [...n] }
{ [ (column [..n])] ON { table | view }
| ON { table | view }  [ (column[..n]) ]
| ON { stored_procedure }
TO security_account [...n]  [ CASCADE ]  [ AS { role | group} ]
```

Значения параметров команд:

ALL – пользователю будут предоставлены все возможные разрешения;

Permissions – список доступных операций, которые предоставляются пользователю. Можно предоставлять одновременно несколько разрешений;

Statement – предоставление разрешений на выполнение команд T-SQL;

Security_account – указывается имя объекта системы безопасности, которую необходимо включить в роль. Это могут быть как учетные записи SQL-сервер, так и группы пользователей Windows;

WITH GRANT OPTION – позволяет пользователю, которому выдаются права на объект, самому назначать права на доступ к данному объекту другим пользователям;

As role | group – позволяет указать участие пользователя в роли, которому предоставляется возможность предоставлять права другим пользователям;

CASCADE – позволяет отзывать права не только у данного пользователя, но и у всех пользователей, которым он предоставил данные права.

Задание

Для базы данных реализовать систему безопасности: создать трех пользователей и назначить им права доступа к таблицам, представлениям и хранимым процедурам, созданным в предыдущих лабораторных работах.

Содержание отчета: тема и цель работы; прокомментированный SQL-код выполнения задания.

Контрольные вопросы

1 Для чего предназначены инструкции GRANT, DENY, REVOKE*?*
2 Что такое неявное отклонение доступа?
3 Для чего предназначен параметр WITH GRANT OPTION?

Список литературы

1 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы: ГОСТ 34.602–89. – URL: http://docs.cntd.ru/document/gost-34-602-89 (дата обращения: 05.09.2025).

2 **Агальцов, В. П.** Базы данных : учебник в 2 т. / В. П. Агальцов. – М. : ФОРУМ ; ИНФРА-М, 2024. – Т.2: Распределенные и удаленные базы данных. – 271 с.

3 **Агальцов, В. П.** Базы данных : учебник в 2 т. / В. П. Агальцов – М. : ФОРУМ ; ИНФРА-М, 2025. – Т. 1: Локальные базы данных. – 352 с. : ил.

4 **Бен-Ган, И.** Microsoft SQL Server 2012. Создание запросов : учебный курс Microsoft : пер. с англ. / И. Бен-Ган, Д. Сарка, Р. Талмейдж. – М. : Русская редакция, 2015. – 720 с. : ил.

5 **Бондарь, А. Г.** Microsoft SQL Server 2014 / А. Г. Бондарь. – СПб. : БХВ-Петербург, 2015. – 592 с. : ил.

6 **Гвоздева, Т. В.** Проектирование информационных систем: технология автоматизированного проектирования. Лабораторный практикум : учеб.-справ. пособие / Т. В. Гвоздева, Б. А. Баллод. – СПб. ; М. ; Краснодар : Лань, 2018. – 156 с. : ил.

7 **Голицына, О. Л.** Базы данных : учеб. пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. – 4-е изд., перераб. и доп. – М. : ФОРУМ ; ИНФРА-М, 2023. – 400 с.

8 **Кузин, А. В.** Разработка баз данных в системе Microsoft Access : учебник / А. В. Кузин, В. М. Демин. – 4-е изд. – М. : ФОРУМ ; ИНФРА-М, 2025. – 224 с.

9 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие / С. С. Куликов. – 2-е изд. – Мн. : БОФФ, 2021. – 600 с.

10 **Куликов, С. С.** Реляционные базы данных в примерах : практ. пособие для программистов и тестировщиков / С. С. Куликов. – Мн. : Четыре четверти, 2020. – 424 с.

11 **Полищук, Ю. В.** Базы данных и их безопасность : учеб. пособие / Ю. В. Полищук, А. С. Боровский. – М. : ИНФРА-М, 2025. – 210 с.

12 Шустова, Л. И. Базы данных : учебник / Л. И. Шустова, О. В. Тараканов. – М. : ИНФРА-М, 2023. – 304 с.