МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных систем»

БАЗЫ ДАННЫХ

Методические рекомендации к лабораторным работам
для студентов специальности
1-40 05 01 «Информационные системы и технологии»
(по направлениям)
очной и заочной форм обучения

Могилев 2021

УДК 004.65
ББК 32.973.26-0.18.2
Б17

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «15» декабря 2020 г., протокол № 5

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 двенадцати лабораторных работ по дисциплине «Базы данных» для студентов специальности 1-40 05 01 «Информационные системы и технологии» (по направлениям) очной и заочной форм обучения. Рассматриваются методологии IDEF1X, IE и язык Transact-SQL в среде Microsoft SQL Server.

Учебно-методическое издание

БАЗЫ ДАННЫХ

Ответственный за выпуск В. В. Кутузов

Корректор Т. А. Рыжикова

Компьютерная верстка Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

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 «Базы данных» (БД) является формирование профессиональных компетенций для работы с современными технологиями моделирования, проектирования, разработки на языке SQL и эксплуатации баз данных, используемых в различных областях науки, техники и экономики.

В результате освоения учебной дисциплины студент

ознакомится с:

– основными понятиями БД, основами построения и функционирования БД, технологиями организации БД;

– языком создания и манипулирования данными SQL;

– способами защиты данных;

– приемами работы в распределенных и многопользовательских БД;

научится:

– строить информационную модель предметной области;

– создавать соответствующую модели базу данных в используемой СУБД;

– организовывать ввод информации в базу данных и вывод отчетов;

– формулировать запросы к БД;

– организовывать работу в многопользовательской БД;

овладеет:

– методами, средствами и технологиями разработки информационных моделей и их программной реализации в выбранной СУБД;

– теорией и стандартами языков описания и манипулирования данными, теоретическими и математическими основами построения выбранной модели данных;

– технологиями и техникой программной реализации баз данных, методами и языковыми средствами манипулирования данными, поддержания целостности, непротиворечивости и защиты информации;

– технологией организации распределенных баз данных, методами и средствами их реализации и использования в решениях профессиональных задач.

Методические рекомендации содержат описание двенадцати лабораторных работ, задания, контрольные вопросы, список литературы [1–8], которую необходимо использовать при подготовке к защите лабораторных работ.

1 Разработка технического задания на проектирование базы данных

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

Теоретические положения

База данных является основным компонентом любой информационной системы (ИС), поэтому проект технического задания (ТЗ) 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

Здесь 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 би-

лета 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;

– добавление нового читателя в библиотеку (в БД), проверка его данных;

– запрос информации о наличии книг в книжном фонде;

– формирование отчета о заказе книг по заявкам преподавателей;
– формирование отчета о списании устаревшей литературы;
– добавление новых книг в каталог библиотеки.

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 2019.

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 трех 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 Разработка логической и физической моделей базы данных с помощью CASE-средств разработки информационных систем

Цель: изучить основные понятия баз данных; изучить основы нотаций Integration DEFinition (IDEF1X) и Information Engineering (IE); разработать логическую и физическую модели БД.

Теоретические положения

Взаимосвязи между основными понятиями БД представлены в таблице 2.1.

Таблица 2.1 – Основные понятия баз данных

Точка зрения	Предметная область	Реляционная теория	Физическое хранение	Стандарт SQL, SQL Server	Объектная модель
Взаимное соответствие терминов	Сущность	Отношение (Relation)	Файл	Таблица (Table)	Класс
	Свойство	Атрибут (Attribute)	Поле (Field)	Столбец (Column)	Свойство
	Экземпляр	Кортеж (Tuple)	Запись (Record)	Строка (Row)	Объект

Существуют три вида связей между таблицами [2–8].

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) 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.

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

Атрибут (атрибуты) первичного ключа должен:

– уникально идентифицировать значение сущности;

– не содержать неопределенное значение NULL;

– сохранять уникальность с течением времени;

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

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.

Составной ключ – первичный ключ, состоящий из нескольких атрибутов.

При формировании составного ключа необходимо руководствоваться следующими положениями:

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

– нельзя включать в состав ключа неуникальное поле, т. е. поле, значения которого могут повторяться 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.

Для реализации связей 1:1 и 1:М необходимо, чтобы дочерняя таблица содержала ссылку (внешний ключ) на родительскую таблицу.

Связь М:М может быть создана только на уровне логической модели. Для разрешения связи М:М на физическом уровне создается новая промежуточная таблица и две новые связи 1:М от имеющихся двух таблиц к новой таблице.

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 сущности 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. Внешний ключ в составном первичном ключе лучше поставить на первое место, что позволит не создавать на нём отдельный индекс.

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 определён без использования значения первичного ключа соответствующего кортежа родительского отношения [7]. 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.

На рисунке 2.1 приведены примеры обозначений различных связей в нотациях IDEF1X и IE.

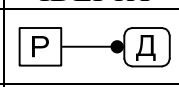
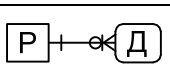
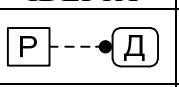
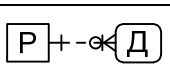
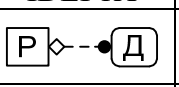
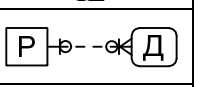
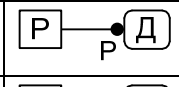
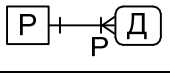
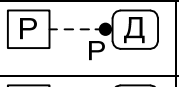
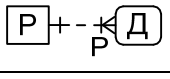
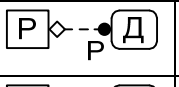
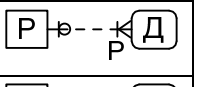
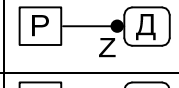
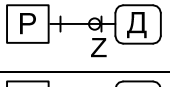
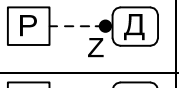
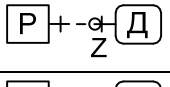
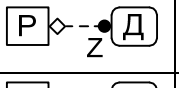
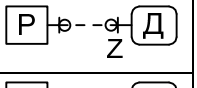
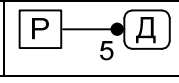
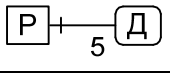
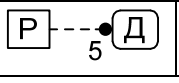
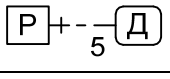
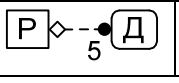
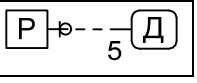
Мощность связи	Тип связи					
	Идентифицирующая		Неидентифицирующая обязательная		Неидентифицирующая необязательная	
	IDEF1X	IE	IDEF1X	IE	IDEF1X	IE
1 к 0, 1 или более						
1 к 1 или более						
1 к 0 или 1						
1 к точно						

Рисунок 2.1 – Правила изображения связей в нотациях IDEF1X и IE для родительского (Р) и дочернего (Д) отношений

Условные обозначения, используемые в нотации UML, приведены в [7, с. 286–293].

Выбор той или иной нотации может определяться как корпоративными стандартами разработчика, так и требованиями заказчика. Поэтому разработчику БД полезно знать несколько нотаций. Нотации IDEF1X и IE во многом схожи. Отличия проявляются в отображении мощности связи между сущностями, а также в отображении и смысле иерархии категории.

Мощность (кардинальность, cardinality) связи – свойство связи, которое служит для указания количества взаимосвязанных строк в таблицах, объединённых связью (например, «один ко многим», «один к пяти» и т. д.).

В нотации IE для числа экземпляров «ноль» используется обозначение ⟜, для числа экземпляров «один» используется обозначение ⊣, для числа экземпляров «много» используется обозначение ≺.

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

Обязательным является применение имен ролей в том случае, когда два или более атрибута одной сущности имеют одинаковую область значений, но разный смысл. На рисунке 2.2 сущность «Продажа валюты» содержит информацию об акте обмена валюты, в котором участвуют две валюты – проданная и купленная. Информация о валютах содержится в сущности «Валюта». Следовательно, сущности «Продажа валюты» и «Валюта» должны быть связаны дважды и первичный ключ «Номер валюты» должен дважды мигрировать в сущность «Продажа валюты» в качестве внешнего ключа. Требуется различать два атрибута, мигрировавших из сущности «Валюта»: один содержит номер проданной валюты, а другой содержит номер купленной валюты, они имеют общую область значений и ссылаются на одну и ту же сущность «Валюта». На рисунке 2.2 атрибутам присвоены разные имена ролей: «Проданная» и «Купленная».

Рисунок 2.2 – Пример обязательного использования имен ролей

Возможны следующие **стратегии обеспечения ссылочной целостности**:

– C – CASCADE (каскадировать) – разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других сущностях 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);

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

– SD – SET DEFAULT (установить по умолчанию) – разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию;

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

– NOA – NO ACTION (игнорировать) – выполнять операции, не обращая внимания на нарушения ссылочной целостности;

– NONE (условного обозначения нет) – не требуется обеспечение ссылочной целостности. При генерации скрипта базы будут созданы только таблицы и связи между ними. Триггеры по поддержанию ссылочной целостности создаваться не будут.

Выбор стратегии контроля ссылочной целостности определяется бизнес-правилами, действующими в моделируемой предметной области, и типом операции: I – вставка (Insert); D – удаление (Delete); U – обновление (Update).

В современных CASE-средствах (Computer Associates AllFusion ERwin Data Modeler, Sparx Enterprise Architect и т. п.) на **логическом уровне** представления модели ранее выявленные сущности, атрибуты и связи размещаются на логической схеме согласно правилам какой-либо нотации (Чена, IDEF1X, IE, UML), как правило, без учета конкретной СУБД.

На **физическом уровне** представления модели максимально учитываются технические особенности работы конкретной СУБД и её возможности по организации и управлению структурами разрабатываемой БД и данными в ней. Одной и той же логической модели может соответствовать несколько разных физических. На физическом уровне составные части БД описываются таким образом, чтобы на основе этого описания в каком-либо CASE-средстве можно было автоматически сгенерировать SQL-код для создания базы данных.

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

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

Логическая модель БД отображена на рисунке 2.3.

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.

Связь между сущностями Lecturer и Order 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 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.

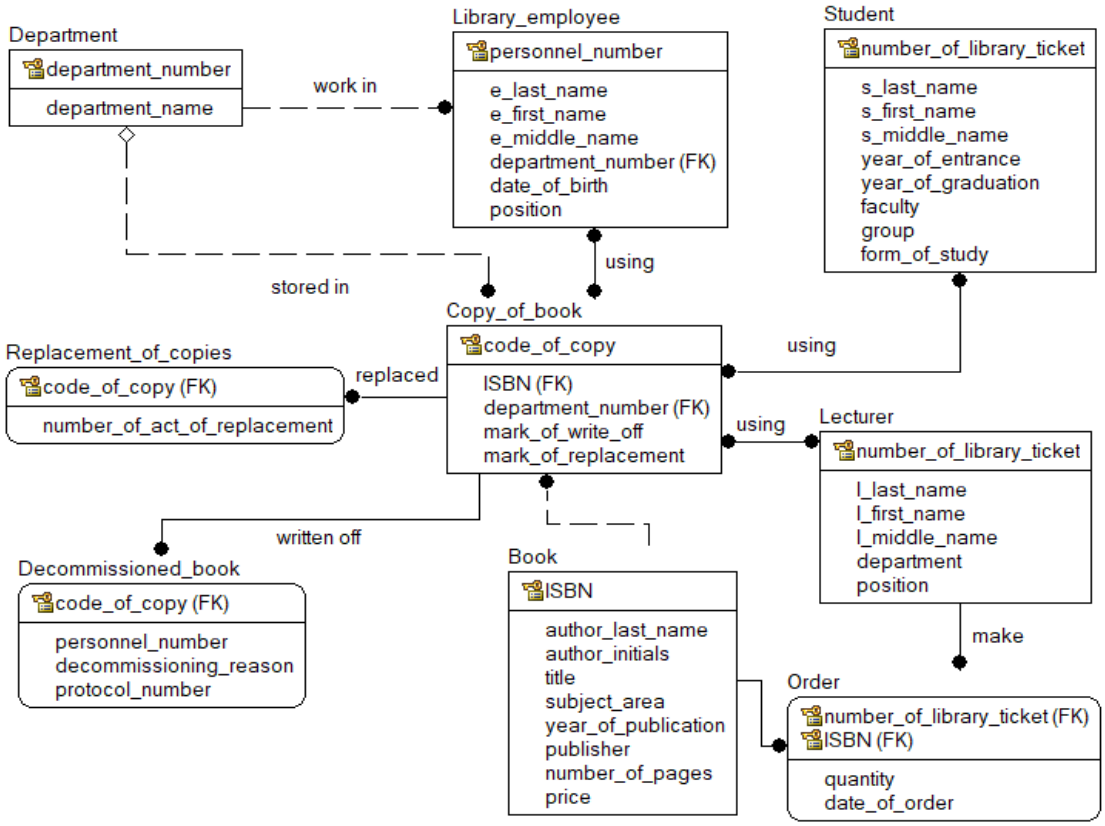


Рисунок 2.3 – Логическая модель БД

Разрешение связей M:M осуществлено на физическом уровне (рисунок 2.4). Были введены дополнительные зависимые сущности (association tables) Issuing_books, Use_of_library_student, Use_of_library_lecturer.

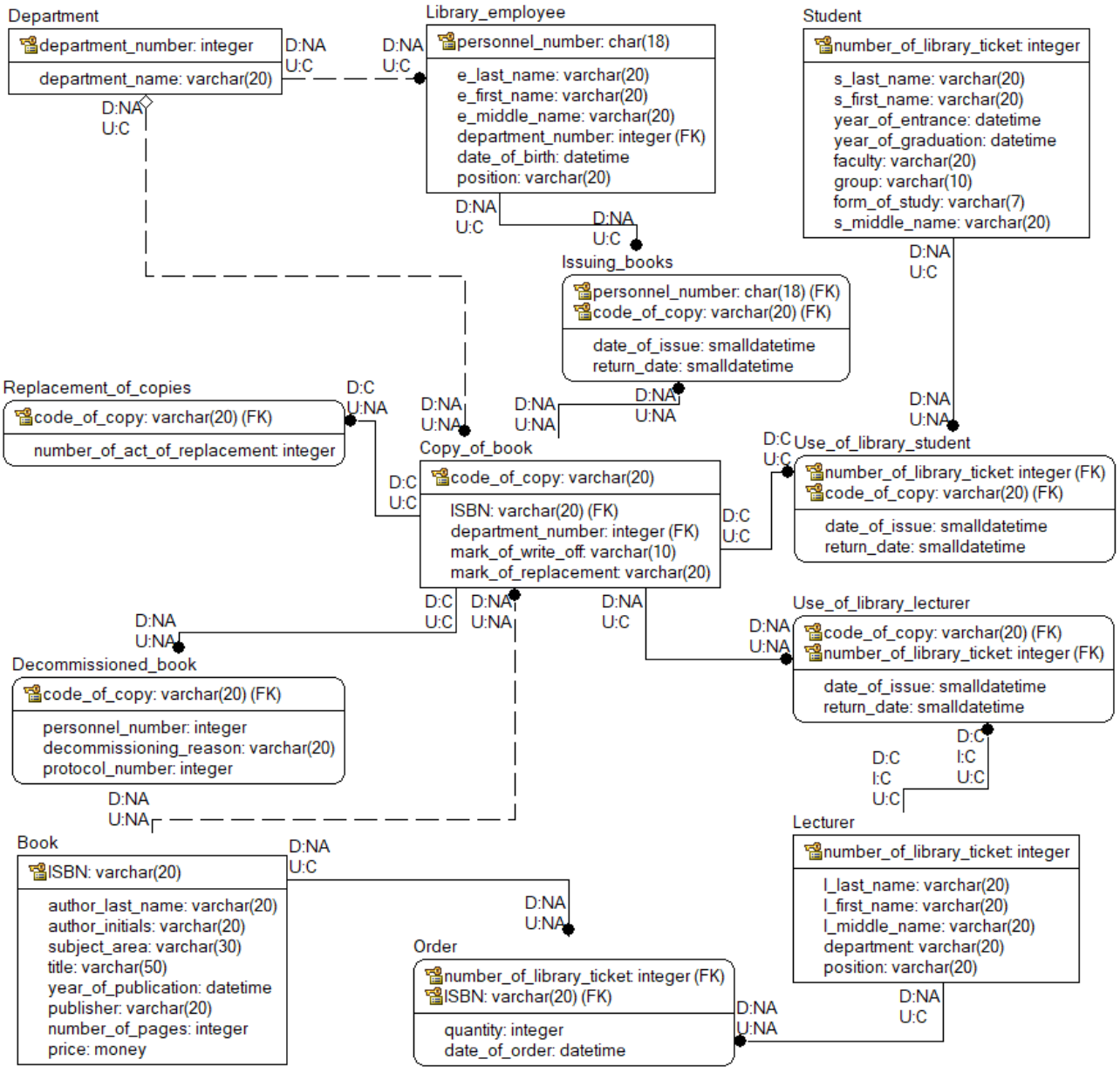


Рисунок 2.4 – Физическая модель БД

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_library_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 разрешается обновление.

Можно сформулировать следующие общие рекомендации по выбору типов данных (применительно к MS SQL Server).

Для коротких символьных значений и символьных строк фиксированной длины следует выбирать тип CHAR. Например, для поля «единица измерения» со значениями «кг», «шт.», «уп.» – CHAR(3), для поля «пол» – CHAR(l) и т. п.

Для символьных строк переменной длины нужно выбирать тип VARCHAR() с указанием максимально возможной длины хранимого значения. Если при добавлении данных длина строки превысит указанное ограничение, система не сможет добавить данные и вернёт сообщение об ошибке.

Для числовых атрибутов, имеющих ведущие нули, следует выбирать тип CHAR, а не числовой тип, иначе ведущие нули будут потеряны. Например, для серии и номера паспорта можно использовать CHAR(10).

Для хранения дат нужно выбирать тип DATE или DATETIME.

Для хранения денежных величин чаще всего используют тип MONEY.

Не рекомендуется использовать столбцы с типами FLOAT и REAL в предложении WHERE инструкции SELECT, т. к. данные типы не хранят точных значений. Также не рекомендуется использовать FLOAT и REAL в финансовых приложениях, в операциях, связанных с округлением. Для этого лучше использовать DECIMAL, MONEY или SMALLMONEY.

Задание

Разработать логическую и физическую модели БД.

Логическая модель должна быть представлена ER-диаграммой логического уровня, разработанной с использованием методологии IDEF1X, или IE, или UML. ER-диаграмма логического уровня должна отображать сущности (не менее пяти), связи между сущностями (в том числе не менее одной связи M:M), имена связей (verb phrase) и атрибуты.

ER-диаграмма физического уровня должна отображать сущности, связи между сущностями и атрибуты, типы данных атрибутов, NULL-значения атрибутов, мощность связей, ограничения ссылочной целостности. Здесь же должны быть разрешены связи M:M.

Содержание отчета: тема и цель работы; ER-диаграммы логического уровня и физического уровней; 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 В чем сущность методологии IDEF1X?

2 Охарактеризовать основные понятия модели «сущность – связь»: сущности, атрибуты, виды ключей (первичный, составной, естественный, суррогатный, альтернативный, инверсный, 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

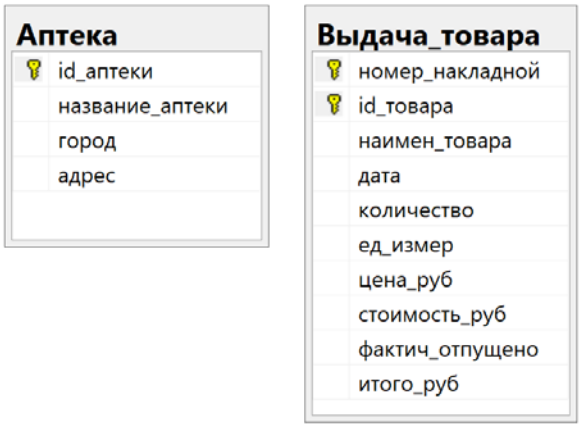


Рисунок 3.2 – База данных в 1НФ

Далее проверяется выполнение требований к 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.

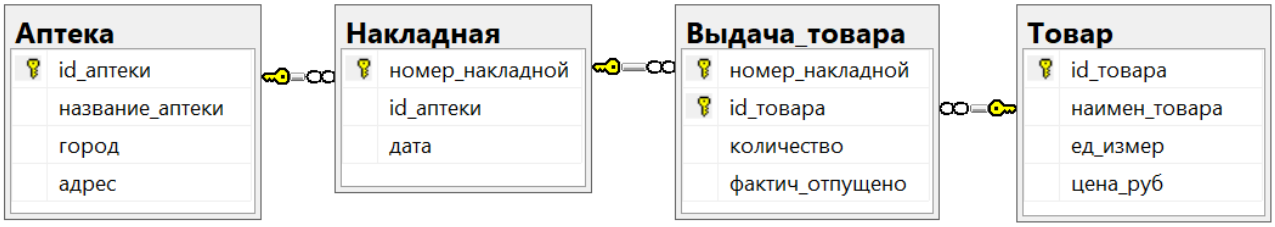


Рисунок 3.3 – Схема базы данных в 3НФ

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 её 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 используются 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 кла-

виши 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, если 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).

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 могут 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 999 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, обеспечивающие возможность сохранения копий строк из других таблиц при удалении данных, определить связи между ними.

Для таблиц, созданных в лабораторной работе № 3, 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 (назвать не менее восьми рекомендаций).

6 Что такое ограничения целостности базы данных?

7 Как создаются связи 1:1, 1:M, M:M?

5 Создание sql-скрипта (сценария) 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 используется **команда INSERT**, имеющая следующий синтаксис [2, 4, 8]:

```
INSERT [INTO] { имя_таблицы | имя_представления }
{[ (column1, …, column_n) ]
{ VALUES ( { DEFAULT | NULL
| выражение }[,... n ] ) | временная_таблица  | инструкция_выполнения }}
|   DEFAULT VALUES
```

Ключевое слово INSERT и необязательное ключевое слово INTO вводят инструкцию. Аргумент «имя_таблицы» задает целевую таблицу, в которую необходимо вставить данные. Можно в качестве цели вставки задать имя представления. Далее может указываться список столбцов, разделенных запятыми, который будет получать вставляемые данные. Вставляемые значения можно задать ключевыми словами DEFAULT, NULL или выражениями. В качестве альтернативы можно использовать инструкцию SELECT для создания временной таблицы, которая станет поставщиком данных для вставки.

Для создания набора вставляемых данных можно использовать инструкцию выполнения вместе с хранимой процедурой или пакетом SQL. Предложение DEFAULT VALUES использует значения таблицы по умолчанию для каждого столбца в новой строке. Инструкция INSERT может вставлять множество строк данных, если в качестве поставщика данных используется временная таблица или результаты инструкции выполнения.

Если в явном виде не указан список столбцов таблицы, то инструкция INSERT будет пытаться вставить значения в каждый столбец таблицы в порядке предоставления значений.

Если выполняется вставка данных в представление, а не в таблицу, то представление должно быть *обновляемым*. Кроме того, с помощью одной инструкции INSERT можно вставить данные только в одну из базовых таблиц, на которые ссылается представление.

Если таблица имеет триггер INSTEAD OF INSERT, то вместо любой инструкции INSERT, пытающейся поместить строки в эту таблицу, будет выполняться код в триггере [4].

При вставке значений в таблицу со столбцом идентификаторов можно задать значение для этого столбца обычным способом. Однако, используя директиву SET IDENTITY_INSERT, можно задать значение для столбца идентификаторов следующим образом:

```
SET  IDENTITY_INSERT  Продукт.Количество
ON  INSERT  INTO  Продукт.Количество (ПродуктID, Name, Value)
VALUES  (17285, 'Сахар', 10')
```

Еще один способ вставки данных во вновь создаваемую таблицу – с помощью **инструкции SELECT INTO**, которая является вариацией простой инструкции SELECT. Ее синтаксис выглядит следующим образом [4]:

```
SELECT Select_List
INTO имя_новой_таблицы
FROM исходник_таблицы
[WHERE условие]
[GROUP BY выражение]
[HAVING условие]
[ORDER BY выражение]
```

Инструкция SELECT INTO в целом идентична инструкции SELECT.

Новым ключевым элементом является предложение INTO. В нем можно задать имя таблицы (с помощью любого корректного идентификатора SQL Server), и инструкция SELECT INTO создаст эту таблицу. Таблица будет иметь по одному столбцу для каждого столбца результатов выполнения инструкции SELECT. Имена и типы данных этих столбцов будут такими же, как и у соответствующих столбцов в списке SELECT. Другими словами, инструкция SELECT INTO использует результаты выполнения инструкции SELECT и преобразует их в постоянную таблицу [4].

В таблицах, создаваемых с помощью инструкции SELECT INTO, не содержатся индексы, первичные ключи, внешние ключи, значения по умолчания и триггеры. Если для таблицы необходимо создать какой-либо из вышеперечисленных объектов, следует создать таблицу с помощью инструкции CREATE TABLE, а затем использовать инструкцию INSERT для заполнения таблицы данными. Обычно это сделать проще, чем создавать таблицу с помощью инструкции SELECT INTO, а затем фиксировать другие свойства с помощью инструкции ALTER TABLE [4].

Инструкцию SELECT INTO можно также использовать для создания временной таблицы. При этом первым символом в имени таблицы следует поставить знак решетки #. Временные таблицы полезны при работе с SQL в длинном триггере или хранимой процедуре, когда во время выполнения процедуры нужно отслеживать информацию. SQL Server автоматически удаляет временные таблицы после окончания работы с ними.

Инструкция UPDATE реализует один из способов изменения любых данных, содержащихся в таблице. Можно написать инструкцию UPDATE таким образом, чтобы она влияла только на отдельное поле в отдельной строке либо вычисляла изменения в столбце во всех строках таблицы. Можно также написать инструкцию, которая будет изменять все строки из множества столбцов. Синтаксис инструкции UPDATE:

```
UPDATE { имя_таблицы | имя_представления }
SET { имя_столбца = {выражение | DEFAULT | NULL}
| @ переменная = выражение | @переменная-столбец = выражение
{ [ FROM {исходная_таблица} [,... n] ]
[ WHERE условие_поиска ] }
```

Для обновляемых строк нужно задать имя таблицы или представления. Ключевое слово SET представляет вносимые изменения. Можно задать значение столбца равным выражению, значению по умолчанию или пустому значению NULL. Можно присвоить выражение локальной переменной. Можно объединить присвоение значения локальной переменной и столбцу в одном и том же выражении. В одной директиве SET можно задать множество столбцов.

Самая простая инструкция UPDATE выполняет одно изменение, которое влияет на все строки таблицы. Например, можно изменить стоимость всех продуктов, перечисленных в таблице «Продукция», на 20 денежных единиц с помощью следующей инструкции:

```
UPDATE Продукция SET Стоимость =20.00
```

Инструкцию UPDATE можно также использовать для присвоения значений локальным переменным. Например, следующий пакет создает локальную переменную, присваивает ей значение и выводит результат на печать:

```
DECLARE @Name nvarchar(50) UPDATE Продукция
SET @Наименование= Крем  PRINT @Наименование
```

В данном случае SQL Server обработал все строки в таблице, хотя инструкция UPDATE не изменяет в ней никаких данных. Чтобы обновление было более эффективным, можно добавить директиву WHERE, которая отбирает отдельную запись:

```
DECLARE @Name nvarchar(50) UPDATE Продукция
SET @Наименование= Крем
WHERE ПродукцияID = 418 PRINT @Наименование
```

Если инструкция UPDATE не отберет никаких строк, она не присвоит значение локальной переменной. Чтобы присвоить значение локальной переменной, не ссылаясь на таблицу, используется инструкция SET.

Для удаления записей при помощи запросов из существующей таблицы можно использовать **инструкцию DELETE**. Ее базовый синтаксис следующий:

```
DELETE [FROM]
{ имя_таблицы | имя_представления }
[ FROM исходная_таблица ]  [ WHERE условия_поиска ]
```

Необязательное ключевое слово FROM можно использовать для удобочитаемости инструкции SQL. Если не включить предложение WHERE в инструкцию DELETE, то инструкция удалит все строки из целевой таблицы [4].

Простейшая инструкция DELETE удаляет все строки из целевой таблицы. Во избежание нежелательных последствий используется инструкция SELECT INTO для создания копии таблицы Клиенты и выполняется работа уже с этой копией:

```
SELECT * INTO КлиентыCopy FROM Клиенты
DELETE FROM КлиентыCopy
DROP TABLE КлиентыCopy
```

Инструкция DROP TABLE удаляет временную копию таблицы после выполнения первых двух инструкций, так что для следующего примера необходимо создать новую копию.

С помощью ограничивающего предложения WHERE можно удалить множество строк, но не целую таблицу:

```
SELECT * INTO КлиентыCopy FROM Клиенты
DELETE КлиентыCopy WHERE Фамилия LIKE 'A%'
DROP TABLE КлиентыCopy
```

Инструкция DELETE не может удалять из таблицы строки со значениями NULL на пассивной стороне внешнего объединения. Рассмотрим, к примеру, инструкцию DELETE со следующей директивой FROM [4].

```
FROM Клиенты
LEFT JOIN Заказ ON Клиенты.КонтактID = Заказ.КонтактID
```

В данном случае таблица «Заказ» содержит значения NULL. Это означает, что столбцы из этой таблицы будут содержать значения NULL для строк, соответствующих контактам, для которых не размещены заказы. В таком случае можно использовать инструкцию DELETE только для удаления строк из таблицы «Клиенты», но не для удаления строк из таблицы «Заказы».

Если инструкция DELETE пытается нарушить работу триггера или ограничения, поддерживающего целостность ссылок, она не будет выполнена. Даже если только одна удаляемая строка из набора нарушает ограничивающие усло-

вия, то выполнение инструкции будет отменено, а SQL Server вернет сообщение об ошибке и не станет удалять строки [4].

При выполнении инструкции DELETE для таблицы, в которой определен триггер INSTEAD OF DELETE, сама инструкция DELETE не будет выполнена. Вместо этого будут выполняться действия триггера для каждой удаляемой строки в таблице [4].

Для удаления первых трех записей в таблице, отсортированной в алфавитном порядке, можно использовать следующую инструкцию:

```
SELECT * INTO КлиентыCopy FROM Клиенты DELETE КлиентыCopy
FROM (SELECT TOP 3 * FROM КлиентыCopy ORDER BY Фамилия)
AS t
WHERE КлиентыCopy.КонтактID = t.КлиентID
DROP TABLE КлиентыCopy
```

Здесь инструкция SELECT в круглых скобках является подчиненным запросом, возвращающим базовый набор строк для инструкции DELETE. Результату этого подчиненного запроса присваивается псевдоним t, а директива WHERE задает параметры сравнения строк из t с перманентной таблицей. Затем директива DELETE автоматически удаляет все совпавшие строки.

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.

Далее будут рассмотрены некоторые способы отбора записей в запросах.

Оператор IN позволяет определить набор значений, которые должны иметь столбцы: WHERE выражение [NOT] IN (выражение). Выражение в скобках после IN определяет набор значений. Этот набор может вычисляться динамически на основе, например, еще одного запроса, либо это могут быть константы.

Например, выберем книги издательств Amedia, либо Aquarius, либо BHV:

```
SELECT * FROM Book
WHERE publisher IN ('Amedia', 'Aquarius', 'BHV')
```

Оператор BETWEEN определяет диапазон значений с помощью начального и конечного значений, которым должно соответствовать выражение: WHERE выражение [NOT] BETWEEN начальное_значение AND конечное_значение.

В таблице 6.1 представлены допустимые подстановочные символы, их значения и примеры использования.

Агрегатные функции выполняют вычисления над значениями в наборе строк. Все агрегатные функции за исключением COUNT(*) игнорируют значения NULL (таблица 6.2).

Таблица 6.1 – 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
[^<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

Таблица 6.2 – 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)

Выражение 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 и 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.

Для создания представлений средствами Transact-SQL используется следующая конструкция:

```
CREATE VIEW view_name [(column [,...n])]
[WITH ENCRYPTION]
AS
select_statement
```

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

WITH ENCRYPTION – использование этого параметра предписывает серверу шифровать код SQL-запроса. Это гарантирует, что пользователи не смогут просмотреть код запроса и использовать его.

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:

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

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

– в представление нельзя включать операции вычисления и группировки, т. е. запрещено указание параметров ORDER BY, COMPUTE и COMPUTE BY.

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.

В качестве примера приведено представление, предоставляющее информацию об экземплярах книг, которые были изданы с 2000 до текущего года.

```
CREATE VIEW InfoExample
AS
SELECT   /* Указываем какие поля будут выбраны */
Copy_of_book.code_of_copy, Book.author_last_name, Book.title, Book.year_of_publication, Book.publisher, Copy_of_book.department_number, Copy_of_book.mark_of_write_off, Copy_of_book.mark_of_replacement
FROM   /* Указываем таблицу и связанные с ней таблицы, из которых выбираются связанные данные */
Book INNER JOIN Copy_of_book ON Book.ISBN = Copy_of_book.ISBN
WHERE Book.year_of_publication BETWEEN 2000 AND YEAR (GETDATE())   /* GETDATE()  возвращает текущую дату, YEAR – год из даты */
```

Задание

Реализовать в разрабатываемой базе данных пять представлений с использованием стандартных функций SQL.

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

Контрольные вопросы

1 Что такое представление и когда целесообразно его использовать?

2 Перечислить команды SQL, с помощью которых представления создаются, удаляются и изменяются.

3 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 хранимой процедуры по ее имени, затем сервер базы данных выполняет блок команд, составляющих тело вызванной процедуры, и возвращает клиенту результат.

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 клиентских приложений. 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 PROCEDURE] procedure_name [;number]
[ {@parameter data_type} [= default] [OUTPUT] ] [,...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).

number – параметр задает 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.

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

OUTPUT – определяет указанный параметр как выходной.

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

AS – ключевое слово, определяющее начало кода хранимой процедуры. После этого ключевого слова следуют команды Transact-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 конспекте лекций и в [4, 8].

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

```
CREATE PROCEDURE Number_of_copies
/*Объявляем необходимые переменные*/
@ISBN varchar(20)
AS
/* Следующая конструкция проверяет, существуют ли записи в таблице «Book» с заданным ISBN*/
IF not EXISTS (SELECT * FROM Book WHERE ISBN = @ISBN)
RETURN 0       /*Вызывает конец процедуры Number_of_copies */
SELECT Copy_of_book.ISBN
INTO TEMP1 /*Сохраняет выбранные поля во временной таблице Temp1*/
FROM Copy_of_book
WHERE ISBN = @ISBN
SELECT COUNT(ISBN)       /*Count подсчитывает количество неповторяющихся записей поля ISBN*/
FROM TEMP1
```

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

```
CREATE PROCEDURE DeleteStudent
@Chit_num int          /*Объявляем необходимые переменные*/
AS                     /*Проверяем, если ссылающиеся записи, если записей нет, разрешается удаление*/
IF not EXISTS (SELECT * FROM Use_of_library_student WHERE number_of_library_ticket =@Chit_num)
DELETE /*Оператор удаления*/
```

FROM Student /*Имя таблицы, откуда нужно удалить*/
WHERE /*Условие удаления – удаляем строку, для которой значение поля number_of_library_ticket совпадает с нужным*/
number_of_library_ticket = @Chit_num

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

```
PROCEDURE  NewOrder
@Kolvo int,
@order_date datetime,
@Chit_nomer int,
@ISBN varchar(20)
AS  /*Проверяем, есть ли запись в таблице «Order» с такими же значениями ключевых полей, как у новой записи*/
IF EXISTS (SELECT * FROM Order WHERE ISBN = @ISBN AND number_of_library_ticket = @Chit_nomer)
RETURN 0  /*Если есть, завершаем выполнение процедуры*/
IF EXISTS (SELECT * FROM Lecturer WHERE number_of_library_ticket = @Chit_number)
/*Проверили, есть ли в «Lecturer» соответствующая запись*/
IF EXISTS (SELECT * FROM Book WHERE ISBN = @ISBN)
/*Проверили, есть ли в «Book» соответствующая запись*/
INSERT INTO  Order  /*Указываем таблицу, куда вставляем запись*/
VALUES  ( @Kolvo, @order_date, @Chit_number, @ISBN)  /*Указываем, какие значения*/
```

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

```
CREATE PROCEDURE UpdateStudent
@Chit_num int,  /*Объявляем необходимые переменные*/
@Fam varchar(20)
AS
IF EXISTS (SELECT * FROM Student  /*Проверяем, существуют ли студенты,*/
WHERE number_of_library_ticket = @Chit_num)  /*читательский номер которых равен искомому*/
UPDATE Student  /*Если такие есть обновляем таблицу «Student»
SET s_last_name=@Fam  /*полю фамилия присваиваем новое значение*/
WHERE number_of_library_ticket = @Chit_num /*если читательский номер записи равен искомому*/
```

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

Цель: изучить курсоры и научиться создавать их.

Курсоры в SQL Server представляют собой механизм обмена данными между сервером и клиентом. Курсор позволяет клиентским приложениям работать не с полным набором данных, а только с одной или несколькими строками.

Существует четыре основных типа курсоров, различающихся по предоставляемым возможностям, – статические, динамические, последовательные и ключевые. Тип курсора определяется на стадии его создания и не может быть изменен. Более подробно курсоры описаны в конспекте лекций и в [4].

Далее рассмотрен пример создания курсора для просмотра информации о студентах и выдачи информации об их количестве.

```
DECLARE curs1 CURSOR
GLOBAL          /*Создается глобальный курсор, который
                будет существовать до закрытия данного соединения*/
SCROLL          /*Создает прокручиваемый курсор*/
KEYSET          /*Будет создан ключевой курсор*/
TYPE_WARNING
FOR
SELECT          /*Какие поля  будут показаны в курсоре*/
Student.number_of_library_ticket, Student.s_last_name, Student.s_first_name, Student.middle_name, Student.year_of_entrance, Student.year_of_graduation, Student.faculty, Student.group
FROM Student                    /*Из какой таблицы выбираются данные*/
FOR READ ONLY                   /*Только для чтения*/
OPEN GLOBAL curs1               /*открываем глобальный курсор*/
DECLARE @@Counter int           /*объявляем переменную*/
SET @@Counter =@@CURSOR_ROWS    /*присваиваем ей число рядов курсора*/
Select @@Counter            /*выводим результат на экран*/
CLOSE curs1                 /*закрываем курсор*/
DEALLOCATE curs1            /*освобождаем курсор*/
```

Задание

В разрабатываемой БД необходимо реализовать три курсора.

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

Контрольные вопросы

1 Что такое курсор? Какие типы курсоров различают?

2 Что такое полный и результирующий наборы строк?

3 Какие основные операции выделяют при работе с курсором? С помощью каких команд T-SQL реализуются основные операции?

10 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; 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 пять 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?

11 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-сервера, 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 дается 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?

12 Взаимодействие с базами данных посредством современных языков программирования

Цель: изучить основы технологии ADO.NET и компоненты доступа к данным в MS Visual Studio .Net.

Теоретические положения

ADO.NET – это набор средств Microsoft .NET Framework, позволяющих приложению легко управлять и взаимодействовать со своим файловым или серверным хранилищем данных.

Для работы с различными СУБД подключаются (using) соответствующие пространства имен: для MS Access – System.Data.OleDb; для MS SQL – System.Data.SqlClient.

В объектной модели ADO.NET можно выделить несколько уровней.

Уровень данных. Это, по сути, базовый уровень, на котором располагаются сами данные (например, таблицы БД MS SQL Server). На данном уровне обеспечиваются физическое хранение информации на магнитных носителях и манипуляция с данными на уровне исходных таблиц (выборка, сортировка, добавление, удаление, обновление и т. п.).

Уровень бизнес-логики. Это набор объектов, определяющих, с какой БД предстоит установить связь и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с БД используется объект DataConnection. Для хранения команд, выполняющих какие-либо действия над данными, используется объект DataAdapter. И, наконец, если выполнялся процесс выборки информации из БД, для хранения результатов выборки используется объект DataSet. Объект DataSet представляет собой набор данных, «вырезанных» из таблиц основного хранилища, который может быть передан любой программе-клиенту, способной отобразить эту информацию пользователю либо выполнить какие-либо манипуляции с полученными данными.

Каждый объект DataAdapter обеспечивает обмен данными между одной таблицей источника данных (базы данных) и одним объектом DataTable в наборе данных DataSet. Если DataSet сдержит несколько таблиц (объектов DataTable), то необходимо иметь и несколько адаптеров данных.

Используя объект DataAdapter, можно читать, добавлять, модифицировать и удалять записи в источнике данных. Чтобы определить, как каждая из этих операций должна произойти, DataAdapter поддерживает следующие свойства: SelectCommand – описание команды, которая обеспечивает выборку нужной информации из базы данных; InsertCommand – описание команды, которая обеспечивает добавление записей в базу данных; UpdateCommand – описание команды, которая обеспечивает обновление записей в базе данных; DeleteCommand – описание команды, которая обеспечивает удаление записей из базы данных. Каждая из этих команд реализована в виде SQL-запроса или хранимой процедуры. Эти свойства являются самостоятельными объектами и относятся к элементам класса OleDbCommand или SqlCommand. Данные объекты поддерживают свойство CommandText, содержащее описание SQL-запроса или хранимой процедуры.

Уровень приложения. Это набор объектов, позволяющих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется объект DataSet, а для отображения данных имеется набор элементов управления (DataGrid, TextBox, ComboBox, Label и т. д.).

Обмен данными между приложениями и уровнем бизнес-логики происходит с использованием формата XML, а средой передачи данных служат либо локальная сеть (Интранет), либо глобальная сеть (Интернет).

В ADO.NET для манипуляции с данными могут использоваться команды, реализованные в виде SQL-запросов или хранимых процедур (DataCommand).

Например, если нужно получить некий набор информации БД, формируется команда SELECT или вызывается хранимая процедура по ее имени.

Когда требуется получить набор строк из БД, необходимо выполнить указанную далее последовательность действий:

– открыть соединение (connection) с БД;

– вызвать на исполнение метод или команду, указав ей в качестве параметра текст SQL-запроса или имя хранимой процедуры;

– закрыть соединение с базой данных.

Связь с базой данных остается активной только на достаточно короткий срок – на период выполнения запроса или хранимой процедуры.

Когда команда вызывается на исполнение, она возвращает либо данные, либо код ошибки. Если в команде содержался SQL-запрос на выборку SELECT, то команда может вернуть набор данных. Можно выбрать из БД только определенные строки и столбцы, применяя объект DataReader, который работает достаточно быстро, т. к. использует курсоры read-only, forward-only.

Для считывания данных, хранящихся в таблице, применяется метод ExecuteReader(). Этот метод возвращает объект SqlDataReader, который используется для чтения данных.

Для определения параметров запросов применяется объект SqlParameter. Параметры, которые используются в командах, могут быть нескольких типов. Тип параметра задается с помощью свойства Direction объекта SqlParameter. Данное свойство принимает одно из значений перечисления ParameterDirection: Input, InputOutput, Output, ReturnValue.

По сравнению с запросами, которые посылаются из приложения базе данных, хранимые процедуры определяются на сервере, предоставляют большую производительность и являются более безопасными.

Объект SqlCommand имеет встроенную поддержку хранимых процедур. В частности, у него определено свойство CommandType, которое в качестве значения принимает значение из перечисления System.Data.CommandType. Значение System.Data.CommandType.StoredProcedure как раз указывает, что будет использоваться хранимая процедура.

При выполнении лабораторной работы рекомендуется также руководствоваться https://metanit.com/sharp/adonet/1.1.php.

Задание

Необходимо, используя технологию ADO.NET, разработать приложение в среде Visual Studio .Net для ввода, изменения и удаления данных в таблицах БД и вывода результатов работы запросов в пользовательском интерфейсе.

Содержание отчета: тема и цель работы; код подключения к БД.

Контрольные вопросы

1 Как осуществляется взаимодействие с БД через объект DataSet?

2 Для чего предназначены объекты SqlConnection, OleDbConnection, DataAdapter DataReader?

Список литературы

1 **ГОСТ 34.602–89**. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы [Электронный ресурс]. – Москва: Стандартинформ, 2009. – Режим доступа: http://docs.cntd.ru/document/gost-34-602-89. – Дата доступа: 29.12.2020.

2 **Агальцов, В. П.** Базы данных: учебник для вузов в 2 кн. Кн. 1: Локальные базы данных / В. П. Агальцов. – 2-е изд., перераб. и доп. – Москва: Форум: Инфра-М, 2012. – 352 с.

3 **Агальцов, В. П.** Базы данных: в 2 т. Т. 2. Распределенные и удаленные базы данных: учебник / В. П. Агальцов. – Москва: Форум: Инфра-М, 2015. – 272 с. : ил.

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

5 **Кузин, А. В.** Базы данных: учебное пособие для студентов высших учебных заведений / А. В. Кузин, С. В. Левонисова. – 6-е изд., стер. – Москва: Академия, 2016. – 320 с.

6 **Шустова, Л. И.** Базы данных [Электронный ресурс]: учебник / Л. И. Шустова, О. В. Тараканов. – Москва: Инфра-М, 2017. – 304 с. – Режим доступа: www.dx.doi.org/10.12737/11549. – Дата доступа: 29.12.2020.

7 **Куликов, С. С.** Реляционные базы данных в примерах [Электронный ресурс]: практическое пособие для программистов и тестировщиков / С. С. Куликов. – Минск: Четыре четверти, 2020. – 424 с. – Режим доступа: http://svyatoslav.biz/relational_databases_book/. – Дата доступа: 29.12.2020.

8 **Куликов, С. С.** Работа с MySQL, MS SQL Server и Oracle в примерах [Электронный ресурс]: практическое пособие / С. С. Куликов. – Минск: БОФФ, 2016. – 556 с. – Режим доступа: http://svyatoslav.biz/database_book/. – Дата доступа: 29.12.2020.