МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

БАЗЫ ДАННЫХ

Методические рекомендации
к лабораторным работам
для студентов направления подготовки
27.03.05 «Инноватика»
очной формы обучения

Могилев 2024

УДК 004.65
ББК 32.973.26-0.18.2
Б 17

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» « 26 » апреля 2024 г., протокол № 10

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

Рецензент канд. техн. наук, доц. А. Е. Науменко

Методические рекомендации содержат описание десяти лабораторных работ по дисциплине «Базы данных» для студентов направления подготовки 27.03.05 «Инноватика» очной формы обучения. Рассматриваются методологии IDEF1X, IE, UML и язык Transact-SQL в среде Microsoft SQL Server.

Учебное издание

БАЗЫ ДАННЫХ

Ответственный за выпуск	В. В. Кутузов
Корректор	И. В. Голубцова
Компьютерная верстка	М. М. Дударева

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 16 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» является формирование профессиональных компетенций для работы с современными технологиями моделирования, проектирования, разработки на языке SQL, индексирования и защиты данных баз данных, используемых в различных областях науки, техники и экономики.

В результате освоения учебной дисциплины студент

познает:

– общую классификацию моделей данных;

– основные функции современных систем управления базами данных (СУБД);

– внутреннюю организацию реляционной СУБД;

– принципы построения СУБД в архитектуре «клиент-сервер»;

– разновидности и способы организации распределенных систем;

– современные промышленно-сопровождаемые СУБД;

научится:

– создавать и модифицировать таблицы базы данных;

– добавлять, удалять, выбирать, изменять данные в таблицах средствами СУБД и языков баз данных;

– реализовывать в базе данных хранимые процедуры, триггеры и представления;

овладеет:

– средствами и технологиями создания и изменения объектов базы данных;

– средствами разработки локальных баз данных.

Методические рекомендации содержат описание десяти лабораторных работ, задания, контрольные вопросы, список литературы [1–7], которую необходимо использовать при подготовке к защите лабораторных работ.

1 Разработка технического задания на проектирование базы данных

Цель работы: разработать проект технического задания на проектирование базы данных информационной системы для выбранной предметной области.

Теоретические положения

База данных (БД) является основным компонентом любой информационной системы (ИС), поэтому проект технического задания (ТЗ) разрабатывается на основе [1] и имеет структуру, представленную 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 билета соответствует номеру формуляра, который содержит информацию о выданных книгах.

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

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

Каждый отдел библиотеки выполняет свои функции.

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

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

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

Директор и заместитель библиотеки имеют доступ ко всей информации.

Библиотекари имеют 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, по его фамилии;

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

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

4.3 Требования к видам обеспечения.

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

Задание

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

Содержание отчета: тема и цель работы; техническое задание объемом не менее четырех 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 CASE-средства концептуального проектирования функциональных моделей информационных систем

Цель работы: разработать функциональную модель информационной системы для выбранной предметной области с использованием методологии BPMN и CASE-средства концептуального проектирования функциональных моделей информационных систем Sparx Systems Enterprise Architect.

Теоретические положения

Функциональная модель ИС отображает функциональную структуру системы, т. е. выполняемые системой действия по преобразованию входов системы в выходы и связи между этими действиями.

BPMN (Business Process Modeling Notation, нотация моделирования бизнес-процессов) – это нотация (система условных обозначений и правил по их использованию), предназначенная для описания предметной области реального бизнеса, т. е. для моделирования бизнес-процессов.

Бизнес-процесс (ГОСТ Р ИСО 19440) представляет собой набор видов деятельности предприятия или набор действий людей и (или) машин), направлен-

ных на реализацию одной или более задач предприятия для достижения цели, создание определённого продукта или услуги для потребителей.

В соответствии с глоссарием стандарта BPMN 2.0 бизнес-процесс (Process) представляется «графом Flow-элементов (набором активностей, событий, шлюзов) и отношений Sequence Flow, связывающих их в исполняемый поток» (https://www.omg.org/spec/BPMN/2.0).

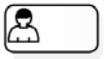
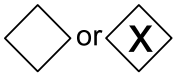
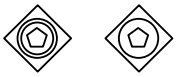
При создании новой функциональной модели в Sparx Systems Enterprise Architect необходимо выбрать File → New project, задать имя файла и тип проекта Enterprise Architect Project (*.eap).

Далее в окне Model Wizard в разделе BPMN следует выбрать тип модели BPMN 2.0 Business Process, добавляемой в основной пакет Model. В таблице 2.1 приведен перечень основных элементов диаграммы BPMN Business Process.

Таблица 2.1 – Элементы диаграммы BPMN Business Process в Enterprise Architect

Изображение	Наименование элемента
1	2
Элементы управления (события, действия, шлюзы)	
 Start Event	Стартовое событие, произошедшее в описании процесса
 Intermediate Event	Промежуточное событие, возникающее между стартовым и конечным событиями
 End Event	Конечное событие, указывающее, в какой точке завершен процесс
 Business Process	Бизнес-процесс
 Activity	Деятельность, задача, подпроцесс – действия (обозначаемые глаголами), которые должны быть выполнены на определенном этапе бизнес-процесса. Действия бывают следующих видов: процесс – сложное действие, подлежащее дальнейшей декомпозиции при моделировании; задача – элементарное действие, которое уже не может быть дальше декомпозировано. Подпроцесс – это деятельность верхнего уровня (помеченная графическим элементом ⊞), которая может быть смоделирована с использованием деятельностей, шлюзов, событий и потоков последовательности. Для того, чтобы создать блок действия подпроцесса, нужно для блока данного действия выбрать Properties → Type: subProcess. Чтобы привязать к блоку действия подпроцесса диаграмму декомпозиции, нужно в контекстном меню блока выбрать New Child Diagram → Add diagram → BPMN 2.0 → Business Process На одном уровне декомпозиции размещают не более 9 действий Типы задач в нотации BPMN 2.0 (Properties → Type процесса):  abstract task (задача общего типа);  manual task (задача, выполняемая вручную, исключающая применение автоматизированных механизмов или приложений, выполняемая за рамками автоматизированной ИС (например, совещание));  business rule task (бизнес-правило – задача, обеспечивающая доступ к механизму бизнес-правил и получение на выходе предоставленной информации об изменениях в бизнес-процессе);

Продолжение таблицы 2.1

1	2
	user task (задача, типичная для технологического процесса (упорядоченной последовательности взаимосвязанных действий), где пользователь выступает в роли исполнителя и выполняет задачи с помощью других людей или программного обеспечения);
	script task (сценарий, скрипт – задача, выполняемая автоматизированной системой без участия человека);
	receive task (получение сообщения);
	send task (отправка сообщения);
	service task (сервисная задача, предназначенная для оказания услуги, которая может, например, являться веб-сервисом или автоматизированным приложением)
 Gateway	Шлюз (развилка) – логический оператор, с помощью которого организуется ветвление и синхронизация потоков управления в модели процесса. Отображает точки принятия решений в процессе. Типы логических операторов: Exclusive – оператор исключающего ИЛИ. При ветвлении потоков исполняется одна ветвь, для которой истинно условие. При слиянии потоков после завершения одной входящей ветви активирует исходящий поток; Event-Based – событийный оператор исключающего ИЛИ. Поток управления направляется по той ветви, где событие произошло раньше. Для слияния потоков не используется; Parallel Event-Based – параллельный событийный оператор. В случае запуска одной ветви процесса по событию ожидаются также другие события, и при их наступлении запускаются соответствующие ветви; Inclusive – включающий оператор И/ИЛИ, используется для разделения потока операций на несколько альтернативных и параллельных маршрутов. При ветвлении активируется одна или более ветвей. При слиянии все выполняющиеся входные ветви завершаются; Complex – комплексный оператор. Используется для моделирования сложных условий ветвления и слияния; Parallel – параллельный оператор И, который используется для создания параллельных маршрутов (без необходимости проверки каких-либо условий) и их объединения. При разделении на параллельные потоки все ветви активируются одновременно. При синхронизации (слиянии) параллельных ветвей оператор ждет завершения всех входящих ветвей, и затем активирует исходящий поток
	
	
	
	
	
	
Зоны ответственности	
 Pool	Пул (область, набор) – это объект, использующийся для отображения области процесса (совокупности всех действий и ответственных за их выполнение лиц). Пул не отображает конкретные внутренние процессы участников, он показывает глобальные взаимодействия и зависимости между участниками процесса. Пул может быть не изображен на диаграмме, но он всегда есть. На одной диаграмме может быть несколько пулов. Пул может содержать несколько дорожек

Окончание таблицы 2.1

1	2
Lane	Дорожка – отображает зону ответственности (внутреннюю роль) участника процесса (директора, менеджера и т. п.). В дорожке описываются все действия лица, ответственного за выполнение задач. Дорожки могут располагаться как вертикально, так и горизонтально
Данные	
Data Object	Объект данных – это элемент, который показывает, какие данные и документы нужны для того, чтобы какое-то действие запустилось, или являются результатом выполненного действия. Типы объектов данных: входные данные (внешний вход, который может использоваться для процесса, задачи); выходные данные (результат выполнения процесса, задачи); коллекция объектов данных, представляющая группу объектов, несущих информацию, например, список заказанных товаров (Properties DataObject: isCollection=true)
Data Store	Хранилище данных – объект, который процесс может использовать для записи и извлечения данных, например, БД или таблица
Message	Сообщение – элемент, отображающий коммуникацию между двумя участниками процесса (e-mail, sms-сообщения, переписка в мессенджере, которыми пользуются участники процесса, коммуникации на сайте компании и т. д.). Типы сообщений: инициирующее (полученное) сообщение (Properties: isInitiating=true); сообщение-ответ (отправленное) (Properties: isInitiating=false)
Соединительные элементы	
Sequence Flow	Последовательный поток – стрелка, показывающая последовательность действий
Association	Ассоциация – позволяет установить соответствие между артефактом и элементом потока (событие, действие, шлюз)
Message Flow	Поток сообщений – показывает сообщения, которыми обмениваются участники бизнес-процесса. Также Message Flows может связывать два отдельных пула в диаграмме
Data Association	Ассоциация данных – соединяет элемент данных (объект данных, хранилище) с элементом потока (событие, действие, шлюз)
Conversation Link	Связь диалога
Артефакты (для добавления дополнительной информации о процессе)	
Group	Группа объектов – прямоугольник, объединяющий несколько действия, принадлежащих к одной категории, но не являющийся действием (задачей, подпроцессом), не влияющий на поток управления
Text Annotation	Текстовая аннотация – комментарий, присоединяемый к какому-либо элементу диаграммы, пояснения, улучшающие читабельность диаграммы

Пример разработки функциональной модели ИС «Библиотека».

Вначале разрабатывают модель верхнего уровня процессов, отображающую подпроцессы в ИС (рисунок 2.1), затем проводят декомпозицию подпроцессов – строят модели, отображающие детали подпроцессов (рисунки 2.2 и 2.3).

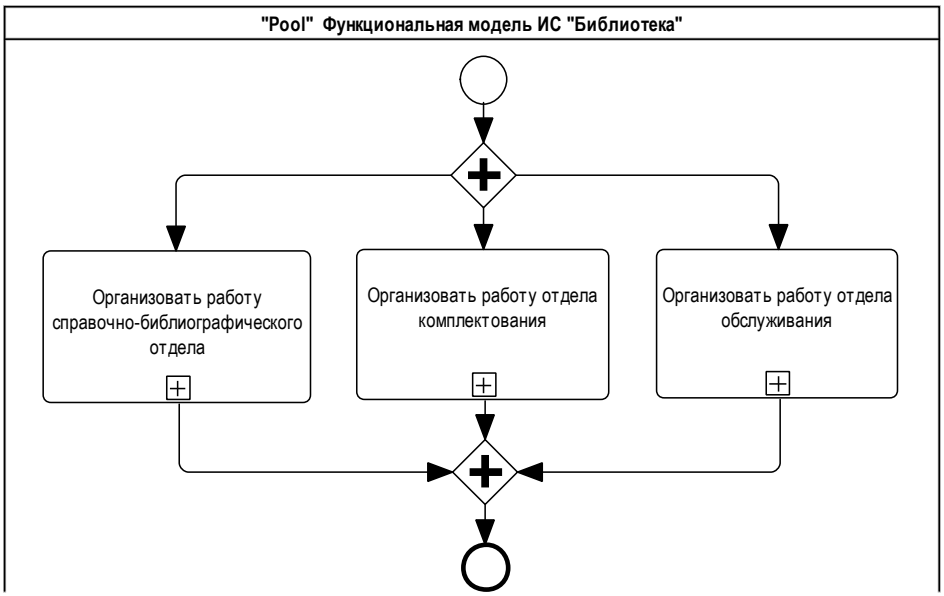


Рисунок 2.1 – Диаграмма подпроцессов функциональной модели (первый уровень)

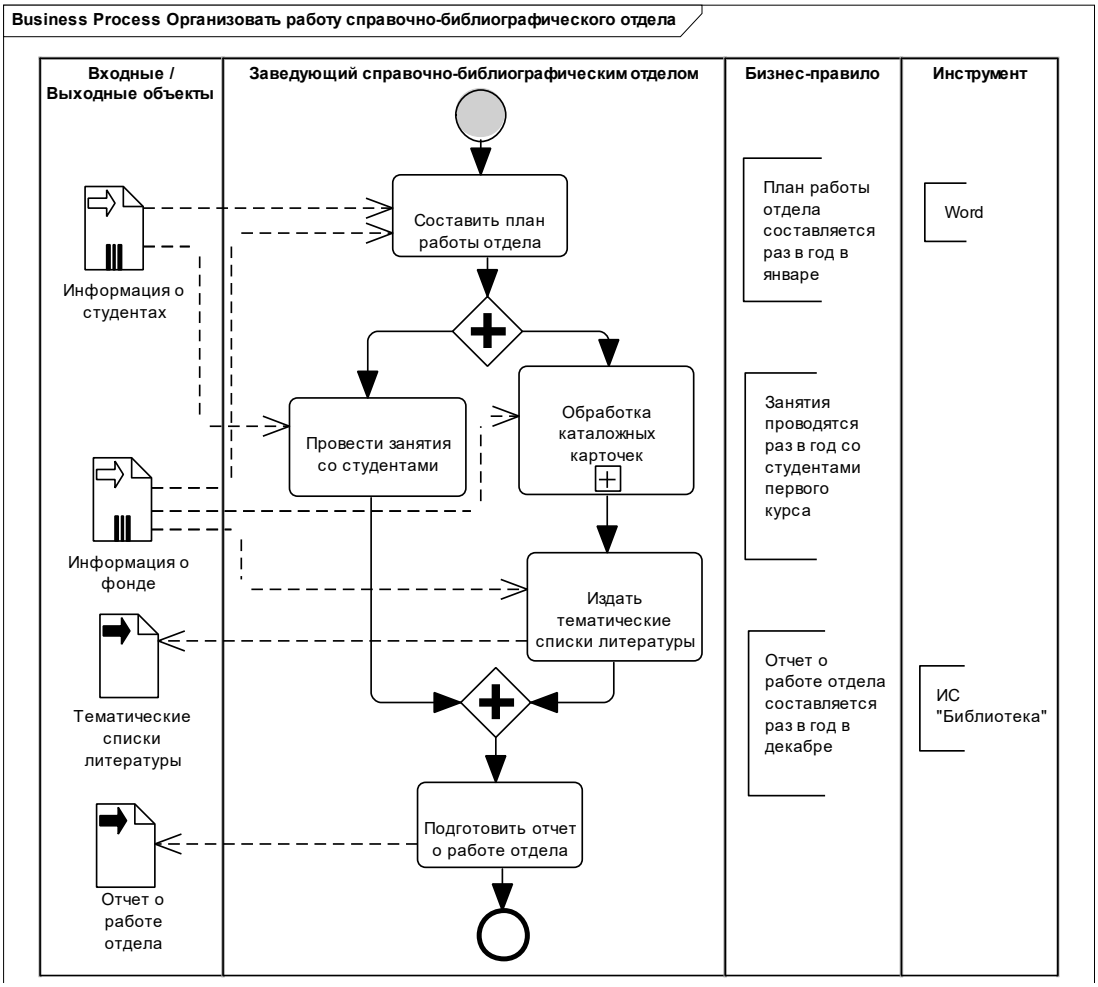


Рисунок 2.2 – Диаграмма декомпозиции подпроцесса «Организовать работу справочно-библиографического отдела» (второй уровень)

Процесс начинается с определенного (начального) события, состоит из всех действий, которые следует выполнить для достижения цели, и завершается

достижением цели или запуском других процессов. Процесс следует понимать, как сквозной набор работ (например, «от поступления заявки до оплаты»), пересекающий всю систему, чтобы выполнить поставленную цель и доставить ценность потребителю.

Для построения функциональной модели выполняют следующие действия.

1 На основе технического задания составляется список действий в моделируемой системе (вначале описывается линейная последовательность действий от начала к результату, далее при необходимости добавляют ветвления).

2 Действия переводятся в задачи (описываемые глаголами в неопределенной форме), т. е. ответы на вопрос «что нужно сделать». Действие может быть сложным и комплексным (например, автоматизировать деятельность отдела), а задача – это простое конкретное действие, выполняемое непосредственно исполнителем (автоматизация деятельности отдела делится на части и выстраивается последовательность). Не рекомендуется использовать в наименовании задачи союз «и», т. к. он объединяет две разные задачи (например, подготовить отчет о деятельности отдела и табель рабочего времени).

3 Определяются исполнители – должностные лица, ответственные за выполнение действий и задач. При необходимости определяются внешние сущности, которые находятся за рамками анализируемой системы (например, клиенты, поставщики) и являются необходимыми для получения результата.

4 Исполнителям назначаются действия.

5 Описываются условия выполнения процессов и задач (шлюзы).

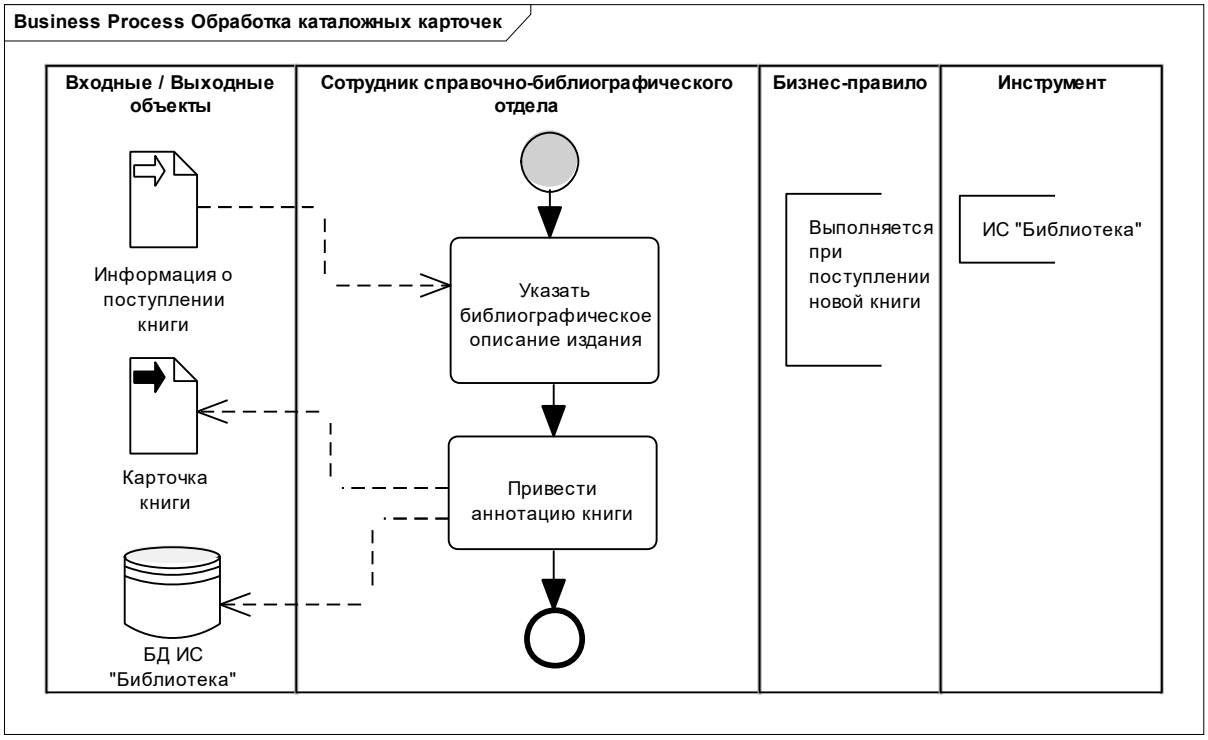


Рисунок 2.3 – Диаграмма декомпозиции подпроцесса «Обработка каталожных карточек» (третий уровень)

Задание

Разработать функциональную модель информационной системы для выбранной предметной области с использованием Enterprise Architect.

Модель должна содержать не менее трех уровней декомпозиции, на последнем уровне должно быть не менее 10 блоков действий (минимум).

Содержание отчета: тема и цель работы; диаграммы декомпозиции.

Контрольные вопросы

1 Перечислить основные этапы моделирования с использованием методологии BPMN.

2 Охарактеризовать элементы нотации BPMN.

3 Чем различаются модели системы AS-IS, TO-BE и SHOULD-BE?

3 Основы использования CASE-средств концептуального проектирования информационной модели системы

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

Теоретические положения

Взаимосвязи между основными понятиями БД представлены в таблице 3.1.

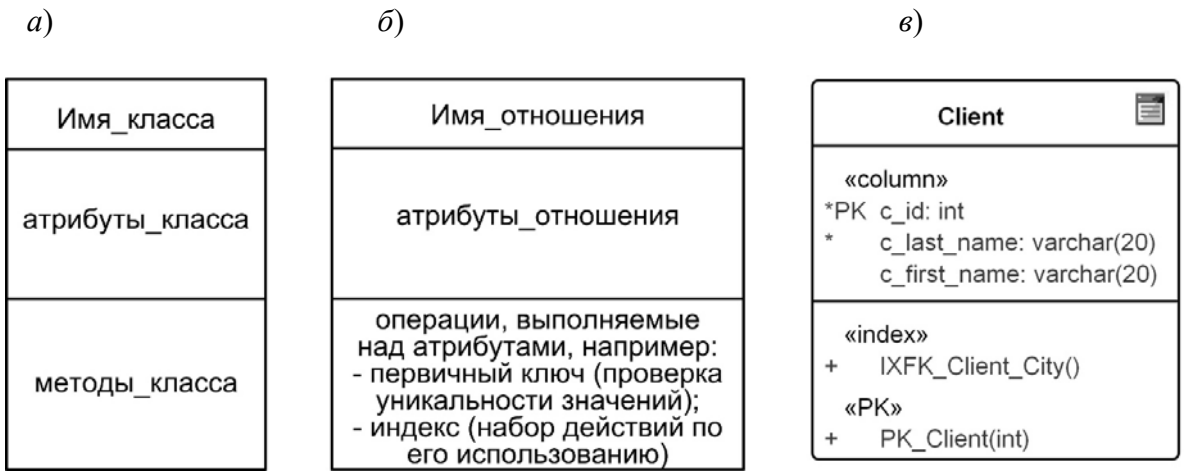
Таблица 3.1 – Основные понятия баз данных

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

На сегодняшний день существует большое количество инструментов для разработки схем БД, например, AllFusion ERwin Data Modeler, DbSchema (https://dbschema.com), DbDiagram.io (https://dbdiagram.io) и т. д. В данной лабораторной работе разработка схемы БД будет вестись с использованием CASE-средства Sparx Systems Enterprise Architect [6, с. 286–293].

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

На рисунке 3.1 показано, как основные понятия языка UML (Unified Modeling Language – унифицированный язык моделирования, язык графического описания для объектного моделирования в области разработки программного обеспечения) связаны с разработкой баз данных. Прямоугольник обозначает класс или отношение, атрибуты класса – атрибуты отношения, к методам отношения относят первичный ключ, внешний ключ, индексы, т. к., например, первичный ключ – это не просто поле, обладающее определенными свойствами, но и наличие необходимости выполнять проверку уникальности значений.



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

Рисунок 3.1 – Структура отношения в Enterprise Architect

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

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

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

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

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

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

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

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

Связи.

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

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

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

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

1) направленность. Большинство средств проектирования схем баз данных (не только Enterprise Architect) при создании связи придерживается правила «от дочерней к родительской» таблице. В Enterprise Architect в нотациях UML, IDEF1X, IE стрелки связей на схемах изображаются просто от дочернего отношения к родительскому, а не от конкретного поля дочерней таблицы к конкретному полю родительской таблицы.

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

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

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

Значения мощности связи определяются требованиями предметной области, которую описывает база данных. Например: в системе обязательно должен быть хотя бы один модератор, но их может быть не более двух → «1 к 1..2»;

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

4) тип. Существует три классических вида связей между таблицами [2–11]: 1:1, 1:M, M:M.

Связь между сущностями также может быть одного из следующих типов:

а) идентифицирующая связь;

б) неидентифицирующая (обязательная или необязательная);

в) категоризации (типизирующая);

г) рекурсивная (связь сущности самой с собой).

Создание идентифицирующей связи.

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

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

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

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

В появившемся окне Foreign Key Constraint будет предложено создать в дочернем отношении Order столбец внешнего ключа c_id (*) и задать имя для ограничения внешнего ключа FK_Order_Client. В соответствии с определением идентифицирующей связи внешний ключ нужно сделать частью составного первичного ключа. Открыв из контекстного меню таблицы Order окно Columns, нужно для столбца o_c_id поставить галочку в чекбоксе PK. В результате будет создана идентифицирующая связь (типа 1:M). Столбец внешнего ключа o_c_id в составе первичного ключа будет помечен обозначением pfK (рисунок 3.2).

Если выделить созданную таблицу Client, выбрать в контекстном меню пункт Code Engineering → Generate DDL, в появившемся окне Generate DDL задать Generate To DDL Execution Engine и нажать кнопку Generate, то будет автоматически сгенерирован код SQL, описывающий таблицу Client. Указанный код будет размещен на вкладке Execute DDL инструмента Database Builder создания физической модели базы данных.

Приведем пример идентифицирующей связи типа 1:1.

На рисунке у каждого клиента может быть только один паспорт, и у каждого паспорта может быть только один владелец, поэтому между сущностями

Client и Passport связь 1:1 (рисунок 3.3). Так как каждый паспорт обязан быть строго привязан к соответствующему клиенту и может быть определён только через связь с данным клиентом (и не имеет смысла без конкретного клиента), связь будет идентифицирующей.

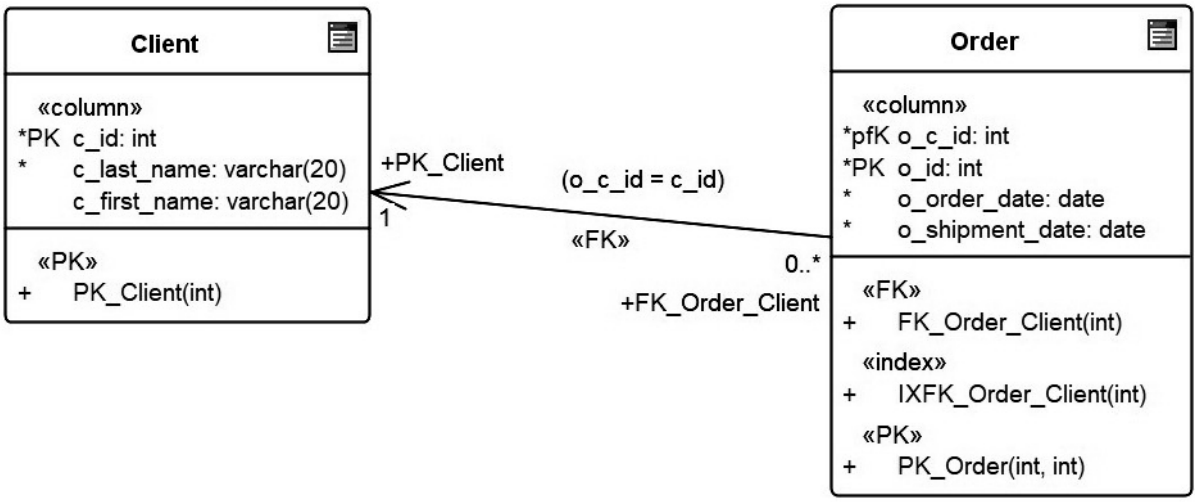


Рисунок 3.2 – Идентифицирующая связь 1:M

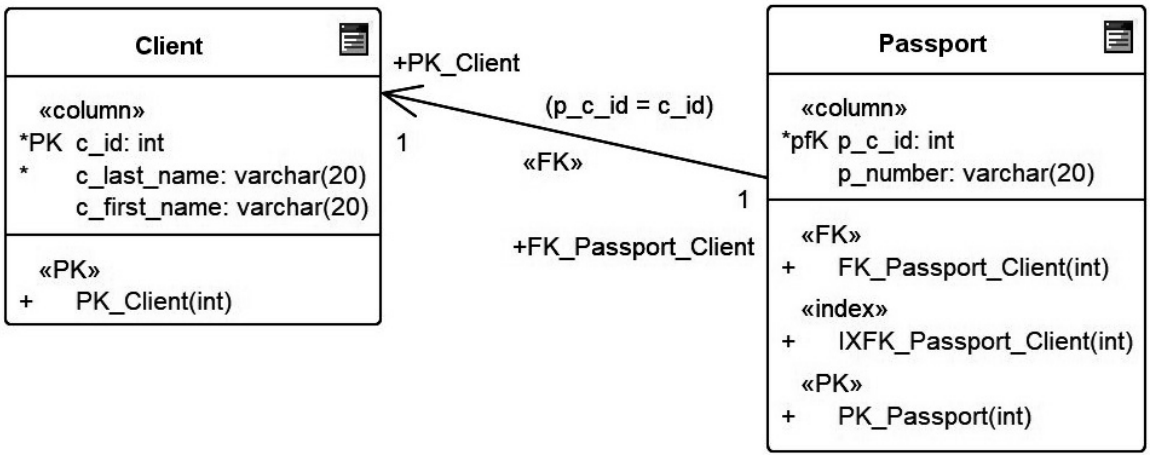


Рисунок 3.3 – Идентифицирующая связь 1:1

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

Создание неидентифицирующей связи.

Неидентифицирующей называется связь между двумя сущностями, в которой каждый экземпляр подчиненной сущности не зависит от значений атрибутов родительской сущности и может существовать без экземпляра родительской сущности. Кортежу дочернего отношения может не соответствовать ни одного кортежа родительского отношения, а сам кортеж дочернего отношения может быть полностью определён без использования значения первичного ключа соответствующего кортежа родительского отношения [9]. Атрибуты

первичного ключа родительской сущности мигрируют в подчиненную, чтобы стать там внешним ключом (неключевыми атрибутами).

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

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 необязательная (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 3.4 показаны сущности Client и City. Поскольку эти сущности могут идентифицироваться независимо друг от друга, между ними имеется неидентифицирующая связь. Если существует бизнес-правило, в соответствии с которым клиент в обязательном порядке должен соотноситься с городом, то связь между отношениями Client и City будет обязательной. Поскольку один клиент относится к одному городу, а из одного города может быть множество клиентов, то устанавливаемая связь будет иметь тип 1:M.

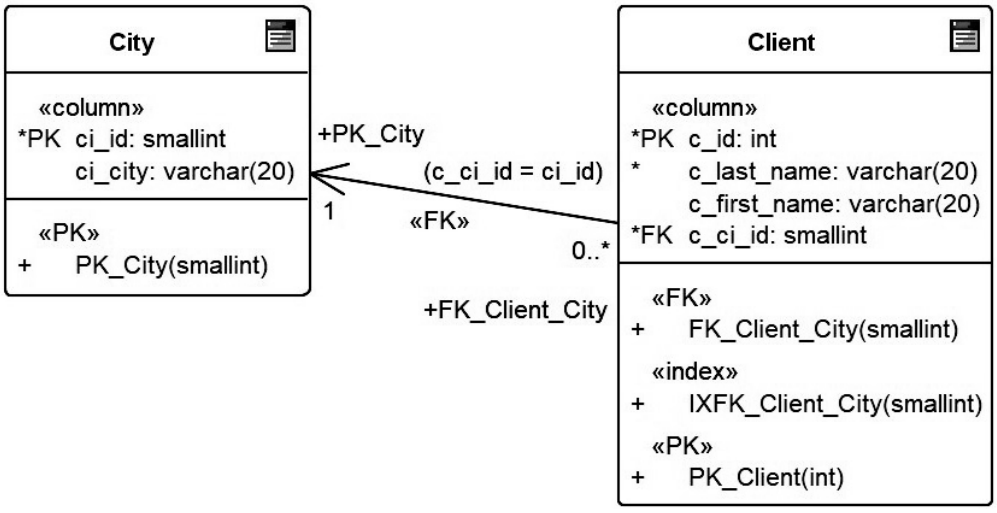


Рисунок 3.4 – Неидентифицирующая обязательная связь

Для того чтобы создаваемую неидентифицирующую связь определить как обязательную, для неключевого столбца внешнего ключа c_ci_id в отношении Client установлен признак Not Null (c_ci_id помечен звездочкой).

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

Создание связи M:M.

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

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

На рисунке 3.5 ассоциативная сущность m2m_seller_product отражает свойства связи M:M. Атрибут «дата поставки товара» не является ни свойством продавца, ни свойством товара, поэтому столбец m2m_delivery_date размещен в дополнительной ассоциативной сущности.

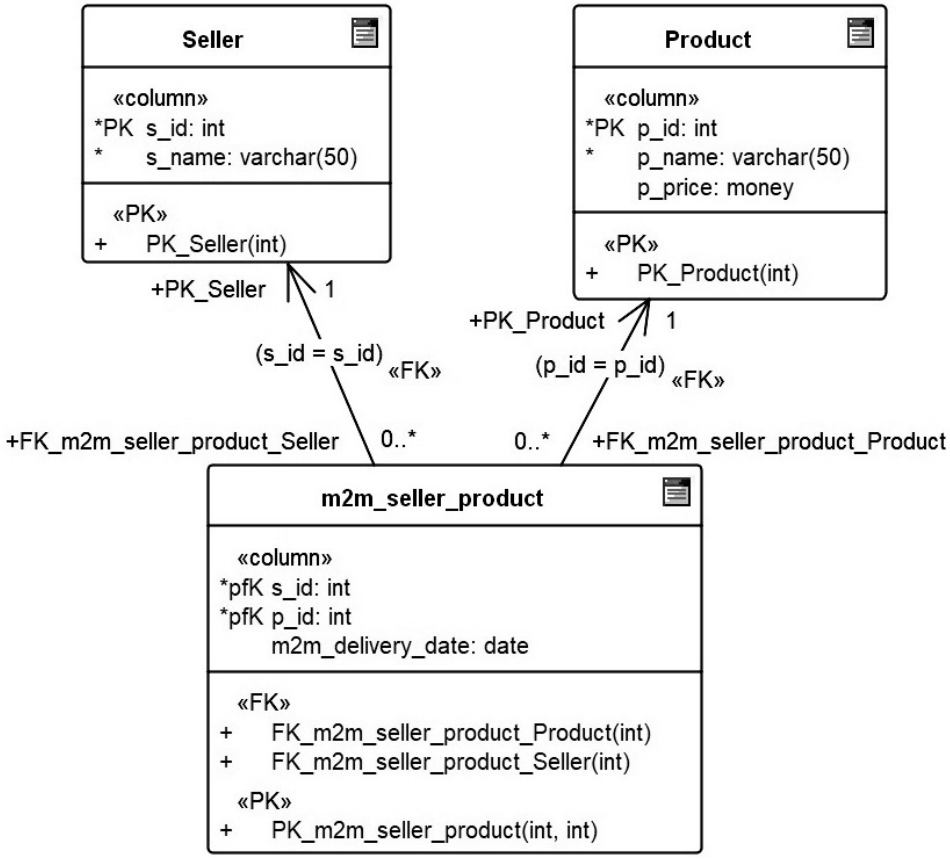


Рисунок 3.5 – Создание связи M:M

Рекурсивная связь – это неидентифицирующая необязательная связь сущности с самой собой (одна и та же сущность является и родительской, и дочерней одновременно), при которой экземпляр сущности может быть связан с другим экземпляром той же самой сущности.

Обеспечение ссылочной целостности базы данных.

Ссылочная целостность (referential integrity) – свойство реляционной базы данных, заключающееся в том, что для каждого значения внешнего ключа дочернего отношения должно существовать соответствующее значение первичного ключа в родительском отношении (т. е. запись в дочернем отношении не может ссылаться на несуществующую запись в родительском отношении). Либо значение внешнего ключа должно быть неопределенным (т. е. не ссылаться на кортеж родительского отношения) [6, с. 71–77].

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

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

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

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

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

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

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

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

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

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

Пример разработки информационной модели.

В информационной модели библиотеки используются следующие сущности:

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

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

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

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

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

– Replacement_of_copies – хранит номера актов замены книг;

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

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

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

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

Информационная модель БД ИС отображена на рисунке 3.6.

Далее приведены примеры описания связей между отношениями и принятых ограничений ссылочной целостности на рисунке 3.6.

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

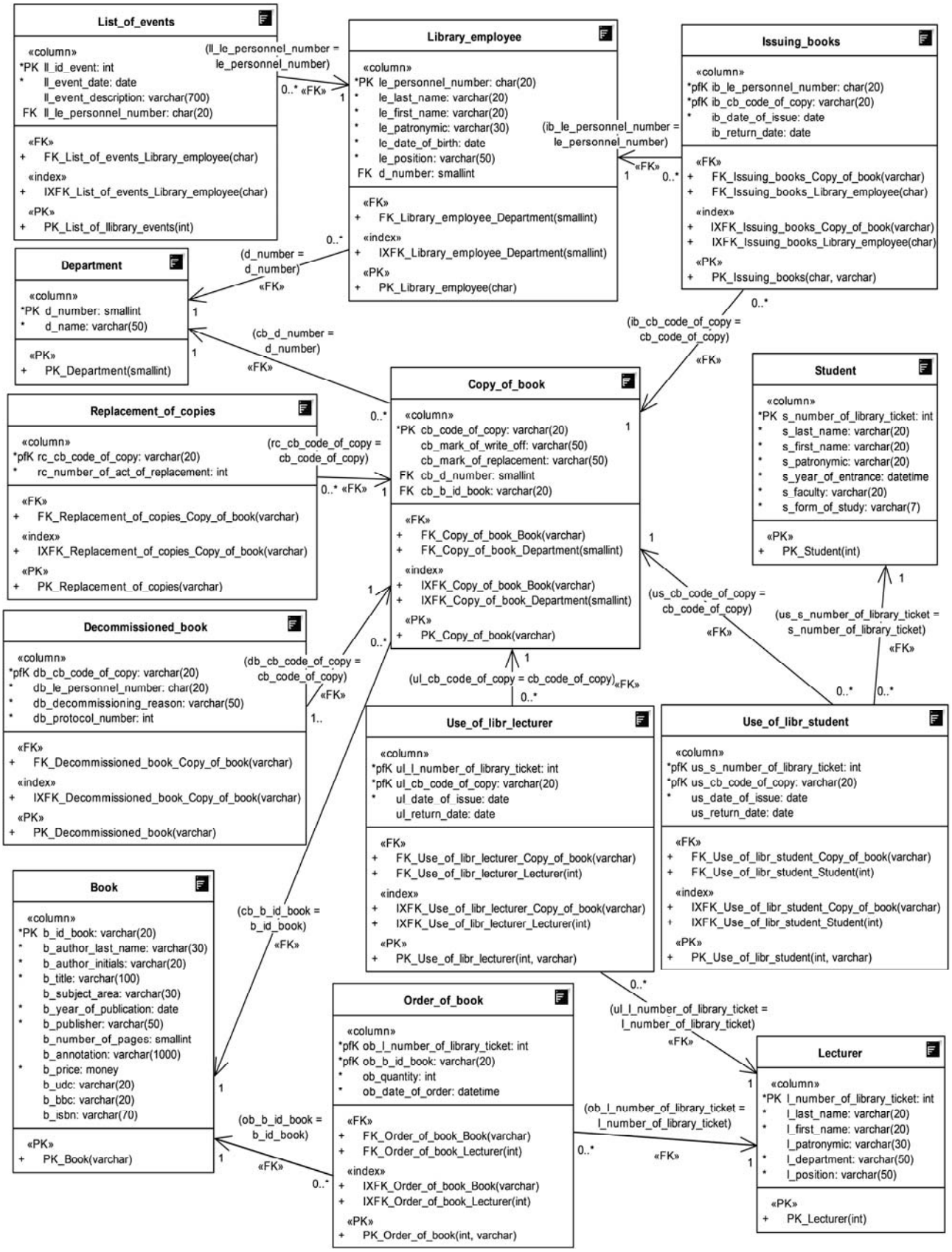


Рисунок 3.6 – Информационная модель

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 Student и Copy_of_book – M: M, т. к. один студент может пользоваться многими экземплярами, а один экземпляр может быть у многих студентов. Для разрешения связи M: M была введена дополнительная зависимая сущность Use_of_libr_student.

Ссылочная целостность реализована с учетом следующих ограничений предметной области:

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

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

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

– в таблице 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 Order_of_book разрешается обновление.

Задание

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

Содержание отчета: тема и цель работы; информационная модель; перечень всех сущностей, входящих в модель, с описанием их назначения и указанием атрибутов каждой сущности; подробное обоснование выбранных типов связей (идентифицирующих и неидентифицирующих (обязательных и необязательных)) между сущностями, принятых ограничений ссылочной целостности.

Контрольные вопросы

1 Каковы особенности применения нотации UML при проектировании БД?

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

3 Что такое ссылочная целостность?

4 Создание базы данных на основе реляционной СУБД

Цель работы: 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 [3, 6, 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 4.1), 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 4.2). В таблице «Выдача товара» однозначно идентифицировать каждую запись будет комбинация полей «номер_накладной» и «id_товара».

Далее проверяется выполнение требований ко 2НФ и 3НФ. Поле «дата» зависит только от номера накладной. Для приведения к 2НФ выделим указанные поля в отдельную таблицу «Накладная» (рисунок 4.3). Аналогично поля «ед_измер» и «цена_руб» зависят только от поля «наименование_товара», поэтому их выделим в таблицу «Товары».

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 4.1 – Образец накладной

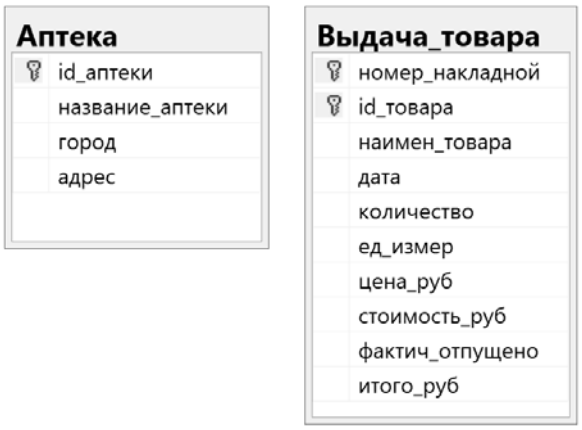


Рисунок 4.2 – База данных в 1НФ

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 4.3.

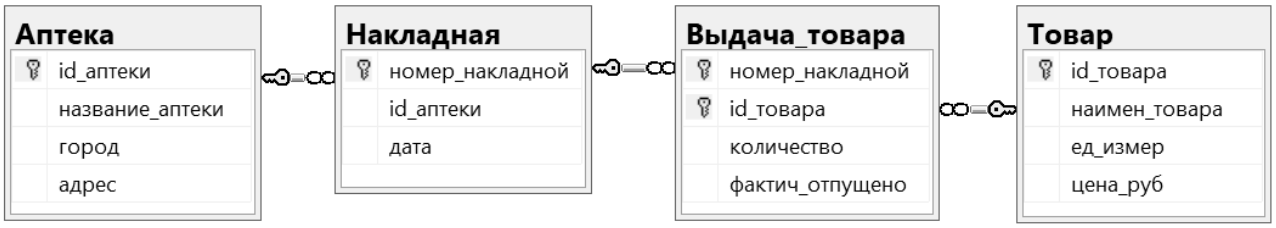


Рисунок 4.3 – Схема базы данных в 3НФ

Microsoft SQL Server – 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/).

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 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.

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

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

– Data Type – тип данных поля [3–7];

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

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.

5 Создание таблиц, связей между ними и индексов средствами SQL

Цель работы: изучить основы создания таблиц, связей и индексов средствами SQL, а также средствами SSMS.

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) ]]
[<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 можно указать начальное значение (seed) и приращение (increment).

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

Пример создания таблицы с ограничениями на значения столбцов:

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

Синтаксис ограничения внешнего ключа на уровне столбца имеет следующий 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 (или ON UPDATE) определяет, что должно произойти со строками дочерней таблицы (с FK) при удалении (или при обновлении) соответствующей строки родительской таблицы. Можно задать один из следующих вариантов: NO ACTION, CASCADE, SET NULL, SET DEFAULT.

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 следующий 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. В индексируемом столбце также желательно запретить хранение значений NULL. Не разрешено выполнение команд INSERT и UPDATE, приводящих к появлению дублирующихся значений.

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 скобках указываются имена столбцов, на основе которых будет построен индекс.

Задание

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

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

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

Контрольные вопросы

1 С помощью каких команд T-SQL можно создать, изменить или удалить таблицу? Какие ограничения на значения столбцов можно накладывать?

2 Что такое кластерный, некластерный и уникальный индексы? Перечислить общие рекомендации при планировании стратегии индексирования.

3 Что такое ограничения целостности базы данных? Как создаются связи 1:1, 1: M, M:M, идентифицирующая, неидентифицирующая обязательная?

6 Создание sql-скрипта (сценария) создания и заполнения базы данных

Цель работы: приобрести навыки создания sql-скриптов заполнения БД.

Теоретические положения

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

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. Для заполненной БД автоматически сгенерировать скрипт.

Пример команды INSERT:

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

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?

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

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

Теоретические положения

Для вставки данных в таблицу средствами T-SQL используются операторы INSERT, UPDATE, DELETE, синтаксис которых рассмотрен в [3–5, 7].

В таблице 7.1 приведены примеры **оператора INSERT** для вставки данных в таблицу, SQL-код определения которой приведен ниже:

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

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

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

В таблице 7.2 приведены примеры использования **оператора UPDATE**.

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

Таблица 7.1 – Примеры инструкций INSERT

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

Таблица 7.2 – Примеры инструкций UPDATE

Задача	SQL-код решения
Обновление в таблице Book_copy всех значений столбца b_publisher на основе выражения для определения новых значений	UPDATE Book_copy SET b_publisher = CONCAT('издательство ', b_publisher)
Обновление в таблице Book значений столбца с указанием условия отбора строк для обновления	UPDATE Book SET b_publisher = 'AZ' WHERE b_publisher = 'Лира';
Обновление в таблице Book значений нескольких столбцов с указанием условия отбора строк для обновления	UPDATE Book SET b_publisher = 'AZ', b_title = 'Цветы' WHERE b_publisher = 'Лира';

Окончание таблицы 7.2

1	2
Обновление в столбце b_publisher таблицы Book значения 'ИНФРА-М' на значение 'Y' во всех строках, где b_publisher='ИНФРА-М'	UPDATE Book SET Book.b_publisher = 'Y' FROM (SELECT * FROM Book WHERE b_publisher='ИНФРА-М') AS BS WHERE Book.b_id_book = BS.b_id_book

В таблице 7.3 приведены примеры использования **оператора DELETE**.

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

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

Таблица 7.3 – Примеры инструкций DELETE

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

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

В таблице 7.4 представлены допустимые подстановочные символы для шаблонов LIKE (см. таблицу 7.3), их значения и примеры использования.

Таблица 7.4 – Подстановочные символы, используемые в шаблонах LIKE

Подстановочный знак	Значение	Пример	Результат
%	Символ-шаблон, заменяющий любую последовательность символов	WHERE [title] LIKE 's%'	Строка, начинающаяся с s: Samantha или sven
_ (подчеркивание)	Символ-шаблон, заменяющий любой одиночный символ	WHERE [titleofcourtesy] LIKE 'm_.'	Ms. Mr.
[<список символов>]	Один символ из списка	WHERE [firstname] LIKE '[sp]%'	Строка, в которой первый символ s или p: Sara или Paul
[<диапазон символов>]	Один символ из диапазона. При этом можно перечислить сразу несколько диапазонов (например, [0-9a-z])	WHERE [freight] LIKE '6[5-7]%'	Строка, в которой второй символ – цифра 5, 6 или 7: 65,83 или 677,54
[^<список, диапазон символов>]	Исключает из поискового образца символы из списка или диапазона	WHERE [shipaddress] LIKE '[^0-9]%'	Строка, в которой первый символ не цифра
ESCAPE ' '	Для поиска символа, который является подстановочным знаком, его указывают после Escape-символа с помощью ключевого слова ESCAPE	WHERE col1 LIKE '!_%' ESCAPE '!'	Поиск строки, которая начинается со знака подчеркивания (_), используя в качестве Escape-символа (!)
'ymd'	Для поиска даты используется форма без разделителей, которая не зависит от языка входа в систему для всех типов данных даты и времени	WHERE [birthdate] = '19581208'	1958-12-08 00:00:00.000

Агрегатные функции выполняют вычисления над значениями в наборе строк. Все агрегатные функции, за исключением COUNT(*), игнорируют значения NULL (таблица 7.5).

Таблица 7.5 – Агрегатные функции

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

Выражения в функциях AVG и SUM должны содержать 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.

8 Язык SQL. Работа с представлениями

Цель работы: научиться создавать представления в СУБД MS SQL Server.

Теоретические положения

Представление – это именованный запрос на выборку, сохраненный в БД, который выглядит и работает как таблица, при обращении по имени создает виртуальную таблицу, наполняя ее актуальными данными из БД, с которой можно работать так же, как с реально существующей на диске таблицей. Физически представление реализовано в виде SQL-запроса, на основе которого производится выборка данных из одной или нескольких таблиц или представлений. Представление часто применяется для ограничения доступа пользователей к конфиденциальным данным в таблице [3–5, 7].

Для создания представлений средствами Transact-SQL используется следующая конструкция:

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

Рассмотрим составляющие данной конструкции.

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

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

WITH ENCRYPTION – данный параметр предписывает серверу шифровать код SQL-запроса. Это гарантирует, что пользователи не смогут просмотреть код запроса и использовать его. Если при определении представления необходимо скрыть имена исходных таблиц и колонок, а также алгоритм объединения данных, то следует использовать эту опцию.

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

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

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

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

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

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

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

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

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

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

Задание

В разрабатываемой базе данных необходимо реализовать 10 представлений с использованием 10 стандартных функций T-SQL.

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

Контрольные вопросы

1 Что такое представление и в каких случаях целесообразно его использовать? Перечислить способы создания представлений. Перечислить операторы SQL, с помощью которых представления создаются, удаляются и изменяются.

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

3 Перечислить ограничения при создании представлений.

9 Язык SQL. Создание хранимых процедур и пользовательских функций

Цель работы: научиться создавать хранимые процедуры для вставки, удаления, изменения данных и пользовательские функции с использованием 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 воз-

вращать параметры. При этом клиент осуществляет только вызов процедуры по ее имени, затем сервер базы данных выполняет блок команд, составляющих тело вызванной процедуры, и возвращает клиенту результат [2–5, 7].

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

Хранимые процедуры имеют следующее определение:

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

Рассмотрим составляющие данной конструкции.

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

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

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

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

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

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

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

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

Более подробно вопросы создания хранимых процедур различных видов рассмотрены в [2–5, 7].

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

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

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

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

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

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

us_s_number_of_library_ticket = @num

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

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

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

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

Пользовательские функции (User Defined Functions, UDF).

Пользовательские функции – это процедуры T-SQL, которые инкапсулируют повторно 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.

Далее приведен пример простой скалярной функции 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?

10 Язык SQL. Создание курсоров

Цель работы: научиться создавать курсоры в SSMS.

Теоретические положения

SQL-сервер способен возвращать в результате выполнения запроса сотни тысяч строк. Клиентские приложения не всегда могут справиться с такими объемами данных, т. к. для их хранения требуется много памяти. Решением этой проблемы является использование курсора.

Курсоры SQL-сервер 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.

Скорость выполнения операций обработки данных с помощью курсора заметно ниже, чем у стандартных средств SQL-сервер.

Типы и поведение курсоров.

Существует четыре основных типа курсоров, различающихся по предоставляемым возможностям. Тип курсора определяется на стадии его создания и не может быть изменен.

1 Статические курсоры. При открытии статического курсора (курсора моментального снимка) сервер выбирает все данные, соответствующие заданным критериям, и сохраняет полный результирующий набор строк в системной БД tempbd. На время открытия курсора сервер устанавливает блокировку на все строки, включенные в полный результирующий набор курсоров. Статический курсор не изменяется после создания и всегда отображает тот набор данных, который существовал в БД на момент его открытия.

2 Динамические курсоры. При использовании динамических курсоров не создается полная копия исходных данных, а выполняется динамическая выборка данных из исходных таблиц при обращении пользователя к тем или иным данным. Сервер блокирует строки на время выборки и все изменения, вносимые пользователями в полный результирующий набор курсора, будут видны в курсоре при выборке. Если пользователь внес изменения в данные уже после из выборки курсором, то эти изменения не будут отражаться в курсоре.

3 Последовательные курсоры. Они не разрешают выполнять выборку данных в обратном направлении, т. е. пользователь может выбирать строки данных только от начала к концу.

4 Ключевые курсоры. Ключевой курсор представляет собой набор ключей, идентифицирующих строки полного результирующего набора курсора. Так как сохраняется информация только о ключевых полях строк, ключевые курсоры отражают все изменения, вносимые другими пользователями.

Управление курсорами.

При работе с курсором можно выделить пять основных операций.

1 Создание курсора. Перед тем как использовать курсор, его необходимо создать.

2 Открытие курсора. Сразу после создания курсор не содержит никаких данных так же, как переменная после объявления не содержит значения. Операция открытия курсора наполняет курсор данными.

3 Выборка из курсора и изменение строк данных с помощью курсора. После того как в курсор занесены данные, можно приступать к его использованию. В зависимости от того, какой тип курсора был объявлен при его создании, можно выполнять только чтение или еще и изменение данных.

4 Закрытие курсора. Когда все операции обработки данных завершены, и надобность в курсоре отпадает, его необходимо закрыть. При закрытии курсора сервер освобождает пространство в системной БД tempdb, выделенное курсором при его открытии.

5 Освобождение курсора. Операция удаляет курсор как объект.

Более подробно курсоры описаны в конспекте лекций и в [2–5, 7].

Далее рассмотрены примеры создания курсора для просмотра информации о студентах и выдачи информации об их числе.

```
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.s_number_of_library_ticket, Student.s_last_name, Student.s_first_name,
Student.s_patronymic, Student.s_year_of_entrance, Student.s_faculty
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 */
```

Курсор для просмотра заказов и подсчета общего количества заказанных книг.

```
DECLARE curs2 CURSOR
GLOBAL SCROLL KEYSET
TYPE_WARNING   /* Сервер будет информировать пользователя о неявном изменении типа курсора, если он несовместим с запросом SELECT */
FOR
SELECT          /* Что будет показано в курсоре */
```

```
Lecturer.l_last_name, Lecturer.l_first_name, Lecturer.l_patronymic, Book.b_author_last_name, Book.b_title, Order_of_book.ob_quantity, Book.b_price
FROM Book INNER JOIN (Lecturer INNER JOIN Order_of_book ON Lecturer.l_number_of_library_ticket = Order_of_book.ob_l_number_of_library_ticket)
ON Book.b_id_book = Order_of_book.ob_b_id_book
FOR UPDATE       /* Курсор для обновления */
OPEN GLOBAL curs2
DECLARE
  @@l_name varchar(20),
  @@f_name varchar(20),
  @@patronim varchar(20),
  @@author  varchar(20),
  @@title varchar(20),
  @@qty int,
  @@price int,
  @@Counter int,
  @@Var1 int
SET @@Counter = 1
SET @@Var1 = 0
WHILE @@COUNTER< @@CURSOR_ROWS       /* Пока счетчик просмотренных строк меньше их общего числа */
BEGIN
FETCH curs2 INTO @@l_name, @@f_name, @@patronim, @@author, @@title, @@qty    /* Просматриваем строки и значения */
SET @@Counter = @@Counter + 1          /* Меняем значение счетчика при переходе к другой строке */
SET @@Var1 = @@Var1 + @@qty * @@price    /* Суммируем стоимости заказанных книг */
END
SELECT @@Var1              /* выводим сумму на экран */
CLOSE curs2
DEALLOCATE curs2
```

Задание

В разрабатываемой БД необходимо реализовать три курсора статического и динамического типов, содержащих не менее трех различных функций.

Содержание отчета: тема и цель работы; SQL-код пяти курсоров.

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?

11 Язык SQL. Работа с триггерами

Цель работы: научиться создавать триггеры в SSMS.

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. Более подробно триггеры описаны в конспекте лекций и в [4].

Для создания триггера используется следующая инструкция T-SQL:

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

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

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

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

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

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

Для изменения триггера используется команда ALTER TRIGGER.

Далее приведен пример триггера, который будет запрещать удаление записей таблицы «Issuing_books», если текущий пользователь не владелец БД 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     /*операция, на которую срабатывает триггер (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'
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?

12 Назначение прав доступа пользователям к объектам базы данных средствами SQL

Цель работы: изучить основы управления правами доступа к объектам БД 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.

С каждым защищаемым объектом в SQL Server связаны разрешения, предоставляемые участнику. Управление разрешениями в Database Engine выполняется на уровне сервера (назначение разрешений именам входа и ролям сервера) и на уровне БД (назначение разрешений пользователям и ролям БД).

Для предоставления разрешений permission на защищаемый объект securable участнику principal используется инструкция GRANT, общая концепция которой имеет следующий вид: GRANT <some permission> ON <some object> TO <some user, login, or group>. Синтаксис инструкции GRANT:

```
GRANT  ALL  | permission [ ( column [ ,...n ] ) ] [ ,...n ]
    [ ON [ class :: ] securable ] TO principal [ ,...n ]
    [ WITH GRANT OPTION ] [ AS principal ]
```

Инструкция DENY запрещает разрешение для участника. Предотвращает наследование разрешения участником через его членство в группе или роли. DENY имеет приоритет над всеми разрешениями, но не применяется к владельцам объектов или членам с предопределенной ролью сервера sysadmin (членам предопределенной роли сервера sysadmin и владельцам объектов не может быть отказано в разрешениях). Синтаксис инструкции DENY:

```
DENY  ALL | <permission>  [ ( column [ ,...n ] ) ] [ ,...n ]
   [ ON [ <class> :: ] securable ]
   TO principal [ ,...n ]
   [ CASCADE] [ AS principal ]  [;]
```

Инструкция REVOKE удаляет разрешение, выданное или запрещенное ранее (иногда говорят, что REVOKE используется для неявного отклонения доступа к объектам БД):

```
REVOKE [ GRANT OPTION FOR ]
    {   [ ALL [ PRIVILEGES ] ]   |   permission [ ( column [ ,...n ] ) ] [ ,...n ]  }
    [ ON [ class :: ] securable ]
    { TO | FROM } principal [ ,...n ]
    [ CASCADE] [ AS principal ]
```

Охарактеризуем назначение параметров инструкций.

ALL – этот параметр устарел и сохранен только для поддержки обратной совместимости. Он не предоставляет все возможные разрешения (см. документацию https://docs.microsoft.com/). Вместо ALL следует предоставлять конкретные разрешения permission.

class – указывает класс защищаемого объекта securable, для которого предоставляется разрешение. Квалификатор области :: является обязательным.

TO principal – имя участника. Состав участников, которым можно предоставлять разрешения, меняется в зависимости от защищаемого объекта.

WITH GRANT OPTION – позволяет пользователю, получающему разрешение, получить также возможность предоставлять данное разрешение другим участникам.

GRANT OPTION FOR – указывает, что возможность предоставлять указанное разрешение будет отменена. Данный аргумент необходим при использовании аргумента CASCADE. Если участник обладает указанным разрешением без параметра GRANT, будет отменено само разрешение.

TO | FROM principal – имя участника. Участники, у которых может быть отменено разрешение на доступ к защищаемому объекту, различны.

CASCADE – позволяет отзывать права не только у данного пользователя, но и у всех пользователей, которым он предоставил данные права. Аргумент CASCADE необходимо использовать совместно с GRANT OPTION FOR. Каскадная отмена разрешения, выданного с помощью WITH GRANT OPTION, приведет к отмене разрешений GRANT и DENY для этого разрешения.

Задание

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

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

Контрольные вопросы

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

Список литературы

1 **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: 20.04.2024.

2 **Агальцов, В. П.** Базы данных [Электронный ресурс]: учебник: в 2 т. Т. 2: Распределенные и удаленные базы данных / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2021. – 271 с. – Режим доступа: https://znanium.com/catalog/document?id=377105. – Дата доступа: 20.04.2024.

3 **Агальцов, В. П.** Базы данных [Электронный ресурс]: учебник: в 2 кн. Кн. 1: Локальные базы данных / В. П. Агальцов. – Москва: ФОРУМ; ИНФРА-М, 2020. – 352 с.: ил. – Режим доступа: https://znanium.com/catalog/product/1068927. – Дата доступа: 20.04.2024.

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

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

6 **Куликов, С. С.** Реляционные базы данных в примерах: практическое пособие для программистов и тестировщиков [Электронный ресурс] / С. С. Куликов. – Минск: Четыре четверти, 2023. – 424 с. – Режим доступа: https://svyatoslav.biz/relational_databases_book/. – Дата доступа: 20.04.2024.

7 **Шустова, Л. И.** Базы данных [Электронный ресурс]: учебник / Л. И. Шустова, О. В. Тараканов. – Москва: ИНФРА-М, 2023. – 304 с. – Режим доступа: https://znanium.com/catalog/document?id=426288. – Дата доступа: 20.04.2024.