

Т.Е. Войтюк, И.С. Осетрова

**ОСНОВЫ ПРОЕКТИРОВАНИЯ
РЕЛЯЦИОННЫХ БАЗ ДАННЫХ СРЕДСТВАМИ
ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ**



**Санкт-Петербург
2020**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Т.Е. Войтюк, И.С. Осетрова
ОСНОВЫ ПРОЕКТИРОВАНИЯ
РЕЛЯЦИОННЫХ БАЗ ДАННЫХ СРЕДСТВАМИ
ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 11.03.02 Инфокоммуникационные
технологии и системы связи
в качестве учебно-методического пособия для реализации основных
профессиональных образовательных программ высшего образования
бакалавриата,

 УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2020

Войтюк Т.Е., Осетрова И.С., Основы проектирования реляционных баз данных средствами инструментальной среды– СПб: Университет ИТМО, 2020.
– 70 с.

Рецензент(ы):

Ананченко Игорь Викторович, кандидат технических наук, доцент, доцент (квалификационная категория "доцент практики") факультета инфокоммуникационных технологий, Университета ИТМО.

Учебно-методическое пособие «Основы проектирования реляционных баз данных средствами инструментальной среды» предназначено для студентов очной формы обучения, осваивающих профессиональную образовательную программу «Инфокоммуникационные системы» направления подготовки 11.03.02 Инфокоммуникационные технологии и системы связи ОГНП «Трансляционные информационные технологии». В пособии представлены общие теоретические вопросы, связанные с проектированием баз данных и практическое руководство по проектированию с использованием инструментальной среды. В результате изучения материала студенты смогут: описывать цели и ключевые бизнес-требования, предъявляемые при разработке баз данных, использовать моделирование данных для создания архитектуры реляционной базы данных, проектировать диаграммы сущность-связь, применять теорию нормальных форм, сопоставлять логическую модель данных с физическими моделями. Для построения моделей данных приводятся рекомендации по применению средства моделирования Oracle SQL Developer Data Modeler.



УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2020
© Войтюк Т.Е., Осетрова И.С., 2020

Содержание

Введение	4
1. Моделирование процессов	5
1.1. Диаграмма потока данных	5
1.2. Компоненты диаграммы и их назначение.....	6
1.3. События	10
1.4. Установка и настройка инструментальной среды моделирования данных 11	
1.5. Проектирование диаграммы потока данных	15
Резюме	18
Контрольные вопросы.....	18
2. Логическая модель данных	19
2.1. Диаграмма сущность-связь.....	22
2.2. Сущность	24
2.3. Атрибут	25
2.4. Отношение.....	25
2.5. Уникальный идентификатор.....	29
2.6. Использование инструментальной среды для построения логической модели	31
Резюме	34
Контрольные вопросы.....	34
3. Теория нормальных форм.....	35
3.1. Первая нормальная форма	36
3.2. Вторая нормальная форма	37
3.3. Третья нормальная форма.....	39
3.4. Нормальная форма Бойса-Кодда.....	40
3.5. Четвертая нормальная форма	42
3.6. Пятая нормальная форма	44
3.7. Преобразование отношений	44
Резюме	47
Контрольные вопросы.....	48
4. Реляционная и физическая модели данных.....	49
4.1. Реляционная модель данных	49
4.2. Преобразование ER-модели в реляционную модель	52
4.3. Соглашения по наименованию объектов реляционной модели	53
4.4. Использование инструментальной среды при проектировании реляционной модели данных	55
4.5. Физическая модель данных	65
Резюме	68
Контрольные вопросы.....	68
Литература	69

Введение

Пособие предназначено для студентов второго курса, обучающихся по образовательной программе «Инфокоммуникационные системы» направления подготовки 11.03.02 Инфокоммуникационные технологии и системы связи ОГНП «Трансляционные информационные технологии».

В пособии рассматриваются концепции моделирования и основные принципы проектирования реляционных баз данных, рассмотрены возможности инструментальной среды Oracle SQL Developer Data Modeler, предназначенной для моделирования данных, описана методика проектирования реляционных моделей и реляционных баз данных под управлением СУБД Oracle.

В результате изучения материала студенты смогут:

- Разрабатывать модели процессов;
- Проектировать модели данных различных типов;
- Проектировать диаграммы сущность-связь;
- Моделировать реляционные базы данных;
- Применять теорию нормальных форм для преобразования ненормализованных данных в реляционные таблицы;
- Преобразовывать логические модели данных в реляционные модели;
- Преобразовывать реляционные модели данных в физические модели;
- Разрабатывать структуру объектов базы данных;
- Определять тип, размер и формат данных;
- Обеспечивать целостность данных базы.

1. Моделирование процессов

При проектировании любой базы данных (БД) основной и главной задачей является моделирование процессов организации, данные которой необходимо хранить. Моделирование процессов используется для документирования бизнес-процессов, которые существуют в организации [16]. Модель процесса определяет, как между собой взаимодействуют и обрабатываются потоки данных, а также характер, роль и область применения данных, характеризует связи и ограничения для участвующих объектов, описывает, как организовано хранение информации и взаимодействие с внешними источниками. Моделирование процессов позволяет проводить анализ, имитировать поведение физической БД без физической реализации и помогает предсказывать дальнейшее развитие системы.

Хорошо спроектированная модель процессов определяет прочную основу для разработки физической базы данных и уже на ранних стадиях позволяет предотвращать ошибки, связанные с физическим проектированием БД, тем самым уменьшая стоимость ошибки.

1.1. Диаграмма потока данных

Для документирования моделей процессов используется диаграмма потока данных или DFD (Data Flow Diagram). Диаграмма позволяет представить задачу в виде функциональных компонентов (процессов), объединенных в сеть и связанных потоками данных. Главная цель таких средств – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами [9].

Часто построение диаграммы потоков данных игнорируется из-за отсутствия временных ресурсов на разработку модели системы, однако она может быть полезна во многих отношениях, и опытные разработчики на начальном этапе проектирования всегда строят такую диаграмму, так как она позволяет:

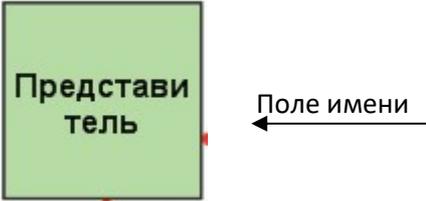
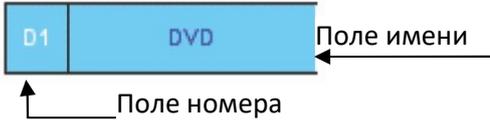
- документировать границы проекта, так что разработчик точно будет знать, кто должен быть вовлечен в реализацию данного проекта и с какой целью;
- понять участникам проекта, работающим над конкретной задачей, конечную её роль в рамках всего проекта;
- увидеть неэффективность текущего способа реализации задачи, тем самым дает возможность усовершенствовать структуру будущей БД на этапе проектирования;

- определить основные информационные потоки, которые будут реализованы в проектируемой системе, и промоделировать ее поведение.

1.2. Компоненты диаграммы и их назначение

Основные компоненты, используемые при построении диаграммы потока данных, их графическое представление и назначение представлены в таблице 1.1.

Таблица 1.1. Основные компоненты диаграммы потока данных

Компонент	Назначение	Графическое представление
Процесс	Функция системы, преобразующая входной поток информации в выходной по определенному алгоритму.	
Внешняя сущность	Показывает входы в систему и/или выходы из систем, включает в себя класс объектов или субъектов, работающих с данной системой.	
Поток данных	Представление потока информации между объектами системы.	
Накопитель данных	Механизм, позволяющий хранить данные для последующей обработки процессами системы.	

Процесс представляет собой дискретную часть работы [1]. Это действие, которое имеет определенную точку начала и конца и служит для преобразования входной информации в выходную информацию по определенному алгоритму.

Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно-реализованное логическое устройство [9].

Каждый процесс должен иметь, по крайней мере, один вход и один выход, а также уникальное имя, которое является глаголом или фразой с глаголом. Процесс должен иметь уникальный номер (идентификатор) для ссылок на него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели [17]. Имя процесса определяет действия, которые непосредственно реализуются данным процессом. Необязательное поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс [11].

Внешняя сущность – это субъект или объект вне системы, являющаяся источником или приемником информации, обрабатываемой процессом. Поэтому внешняя сущность может быть связана информационным потоком только с процессом. Она может быть представлена подразделением, другой системой, материальным предметом или отдельным человеком. При проектировании в графическом объекте «внешняя сущность» указывается ее имя, которое должно быть именем существительным.

Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой системы. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой информационной системы (ИС), если это необходимо, или, наоборот, часть процессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность [9].

Компонент «накопитель данных» определяет данные, находящиеся в состоянии покоя. Данные, которые содержит накопитель, могут выбираться в любом порядке, доступ к ним возможен в любой момент после определения.

Накопитель данных может быть реализован физически в виде ящика в картотеке, таблицы в оперативной памяти, файла на цифровом носителе [2]. Накопитель данных имеет идентификатор для организации доступа к нему, который принято представлять в виде «D№», где № - уникальный номер накопителя, а имя, определяющее его содержимое, должно быть существительным. Следует заметить, что внешние сущности не могут сохранять информацию непосредственно в накопителе данных, информация от них всегда должна быть предварительно обработана процессом.

Поток данных – это компонент, который моделирует передачу информации или других физических компонентов из различных частей системы. Этот компонент определяет движения информации (в/из процесса).

Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемой по почте в виде писем, записанной на компакт-диски или флэш-диски, переносимой с одного компьютера на другой. Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока. Каждый поток данных имеет имя, отражающее его содержание [6].

Информационные потоки можно разделить на 3 типа:

- поток данных – поток информации между двумя элементами, (процесс – внешняя сущность или процесс – накопитель данных);
- материальный поток – поток материальных объектов между двумя элементами;
- временной поток – временная зависимость, показывающая, что процесс должен выполняться последовательно и не может быть выполнен, пока не завершится последовательность действий на предыдущем шаге его исполнения.

В связи с тем, что модель процесса определяет, как информация преобразуется в результате обработки, каждый информационный поток должен быть входом или выходом для одного из процессов разрабатываемой модели. Информационные потоки не могут быть двунаправленными [11]. Если есть необходимость, чтобы поток передавал информацию на вход и выход процесса или другого объекта системы, необходимо реализовать два потока.

Концепция построения модели процессов устанавливает ограничение на создание информационного потока между двумя накопителями данных, так как передача информации из одного накопителя в другой должна сопровождаться предварительной обработкой. По этой же причине недопустима организация передачи информации от внешней сущности в накопитель данных.

Важную роль при построении диаграммы потока данных играет ее специальный вид – контекстная диаграмма, которая предназначена для моделирования систем наиболее общим образом. Такой тип диаграммы позволяет устанавливать границы анализируемой системы, отразить взаимодействие системы с внешними сущностями и определяет основные информационные потоки, существующие в системе (рисунок 1.1).

В качестве примера построения диаграммы потока данных рассмотрен процесс аренды DVD диска в магазине. В этом процессе участвует внешняя сущность «Человек», который хочет стать клиентом данного магазина, и система «Магазин». Основными информационными потоками, связанными с «Человеком», являются посещение магазина для регистрации или возврата DVD диска. Занесение в БД информации о клиенте, о возврате DVD и о новом комиссионном извещении относятся к основным информационным потокам системы «Магазин».

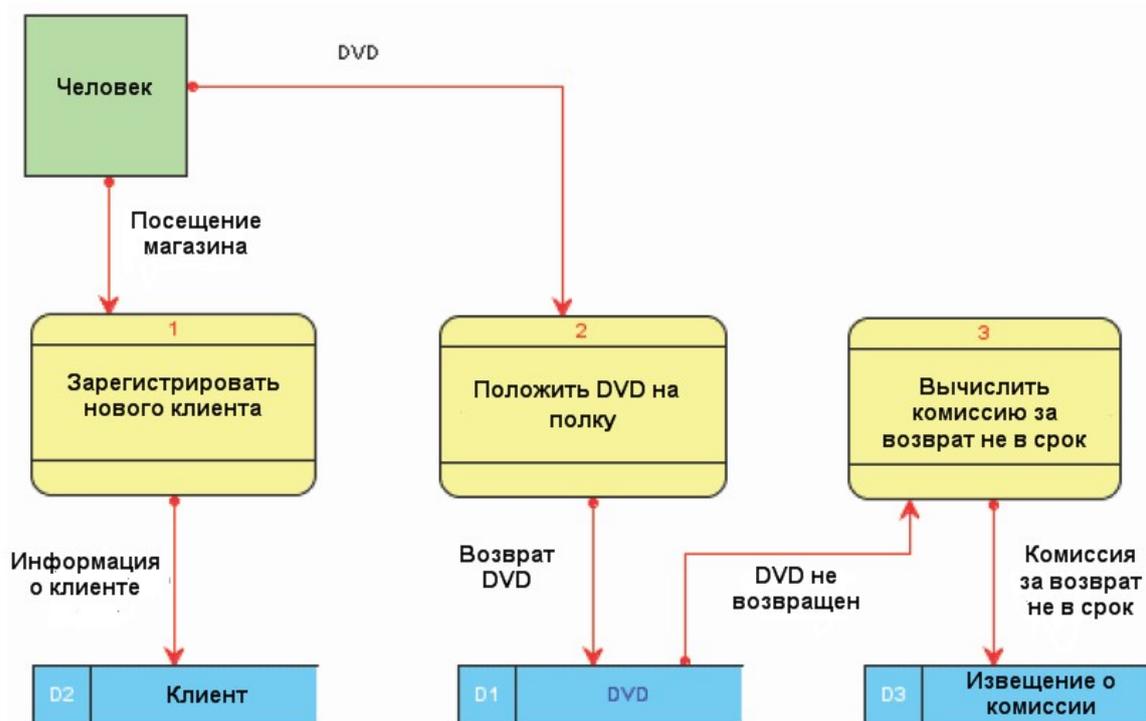


Рис. 1.1 – Контекстная диаграмма

В системе «Магазин» реализованы 3 процесса, обеспечивающие его функционирование: зарегистрировать нового клиента, положить DVD на полку, вычислить комиссию за поздний возврат; и 3 накопителя данных: «Клиент», «DVD», «Извещение о комиссии».

Процессы №1 и №2 преобразуют информацию, предоставленную человеком, и записывают ее в накопитель данных D2 и D1 соответственно. Процесс №3 не взаимодействует напрямую с внешней сущностью. Этот процесс запускается в конце каждого рабочего дня и, анализируя информацию в накопителе «DVD», производит расчет комиссии в случае невозврата DVD в срок с последующим сохранением этой информации в накопителе D3.

Представленная диаграмма показывает, что при проектировании информационных потоков они должны обязательно входить или выходить в процессы. В свою очередь, каждый из процессов может быть представлен совокупностью более детальных процессов и информационных потоков между ними. Таким образом, строится иерархия DFD с контекстной диаграммой в корне дерева. Этот процесс декомпозиции продолжается до тех пор, пока процессы могут быть эффективно описаны с помощью коротких спецификаций обработки (спецификаций процессов) [9].

1.3. События

Полезным механизмом, помогающим проектировать DFD-диаграммы, является определение событий, происходящих в системе, и их анализ. События являются причиной запуска процессов и возникают в результате их завершения.

Можно выделить три основных типа событий:

- внешние события. Это события, происходящие вне области действия системы. Например, решение человека стать клиентом магазина и его регистрация;
- внутренние события. События, происходящие в системе. Например, инвентаризация DVD дисков с последующим изменением в БД информации (каталожного номера, названия, описания и других параметров);
- временные события. Это такая категория событий, которые происходят, когда фактические дата и время достигают заданных. Данная категория событий происходит по расписанию. Например, регистрацию в магазине нужно подтверждать каждый год, получая новую карточку этого магазина.

Анализируя ежедневные события, происходящие в той или иной области бизнеса, дирекции, отделе и т.д., можно также определить реакцию на эти события.

В таблице 1.2 представлены события, которые могут происходить в системе «Магазин» и реакция на них.

Таблица 1.2. События и реакция на них

Событие	Реакция
Покупатель вернулся	Зафиксировать возврат DVD
Партия новых DVD поступила в магазин	Оплата поставщику счета
Вакансия продавца открыта	Поиск сотрудника на данную должность

Анализ событий, происходящих в системе, и фиксирование реакции на эти события определяют новые функциональные требования к ней. Вследствие этого в системе появляются новые информационные потоки, процессы, накопители, которые ранее не были учтены.

1.4. Установка и настройка инструментальной среды моделирования данных

Для построения DFD диаграмм корпорация Oracle предоставляет удобное пользовательское графическое средство – Oracle SQL Developer Data Modeler, которое не требует приобретения лицензии [16]. Для получения доступа к загрузке программного обеспечения необходимым и достаточным условием является регистрация на сайте разработчика <https://www.oracle.com>.

Пакет Oracle SQL Developer Data Modeler представляет собой универсальный, полностью автономный инструмент с поддержкой логического, реляционного, многомерного моделирования и моделирования типов данных [16]. Возможность конструирования моделей данных на разных уровнях позволяет сформировать исчерпывающие концептуальные блок-схемы связей между всеми элементами системы и превратить их в рабочие реляционные модели данных [16]. С помощью пакета Oracle SQL Developer Data Modeler пользователи могут создавать, расширять и модифицировать модели данных, а также сравнивать свои и уже существующие модели [15].

Существует два варианта установки Oracle SQL Developer Data Modeler: архив с Data Modeler со встроенной виртуальной машиной JAVA (JVM), набором библиотек и компилятором (JDK); архив, содержащий только Data Modeler. Выбор необходимой версии загрузки зависит от операционной системы и наличия на компьютере JDK не ниже версии 1.8, что соответствует требованиям на момент издания данного учебно-методического пособия.

Для установки Oracle SQL Developer Data Modeler на операционную систему Windows 64-bit перейдите по ссылке <https://www.oracle.com/tools/downloads/sql-data-modeler-downloads.html>.

Выберите пакет, соответствующий данной операционной системе. Распакуйте содержимое архива, в содержимом полученного каталога найдите файл `datamodeler.exe` и запустите его. Файл EXE сконфигурирован таким образом, что автоматически запускает JVM, поставляемую в этом архиве.

В случае, если на компьютере уже установлен JDK или планируется использовать язык `java` и инструменты разработки, отличные от Oracle SQL Developer Data Modeler, но также требующие наличия JDK, то целесообразно для установки выбрать пакет, содержащий только Data Modeler. В этом случае предварительно необходимо произвести установку и настройку JDK.

Комплект последних версий JDK можно свободно загружать с сайта <https://www.oracle.com/java/technologies/>. При выполнении практических работ в учебном классе требуемый дистрибутив можно скопировать из сетевой папки, содержащий программное обеспечение для обучения.

1. Загрузите с сайта или скопируйте из сетевой папки дистрибутивов на локальную машину Java Development Kit.

2. Установите JDK. Для этого запустите исполняемый файл, например для 64-разрядной операционной системы Windows файл будет иметь имя `jdk-13.0.2_windows-x64_bin.exe`, и следуйте инструкциям по инсталляции для вашей платформы.

Процедура инсталляции предлагает по умолчанию имя каталога для установки JDK. В имя каталога входит номер версии, это упрощает работу, в случае появления новых версий JDK.

3. Настройте переменные окружения `JAVA_HOME`, `PATH`.

Установка переменной окружения `PATH` позволяет запустить исполняемый файл без перехода в содержащий его каталог. При вводе любой исполняемой команды, не обнаружив файла в текущем каталоге, система производит его поиск в каталогах, указанных в переменной `PATH`.

Переменная `JAVA_HOME` используется многими приложениями Java для определения расположения JDK в файловой системе. Она имеет приоритет над переменной `PATH`, если установлена.

Для создания и настройки переменных окружения для работы с Java в ОС Windows 10 выполните следующие действия:

- ✓ Через файл-менеджер выберите правой кнопкой мыши «Компьютер»/«This PC», выберите пункт «Свойства»/«Properties» и далее «Дополнительные параметры системы»/«Advanced System Settings».

- ✓ перейдете на вкладку «Дополнительно»/«Advanced» и нажмите на кнопку «Переменные среды»/«Environment variables». Откроется окно «Переменные среды».

- ✓ для создания переменной среды пользователя «User Variables», щелкните на кнопке «Создать»/«New», откроется окно Новая пользовательская переменная.

- ✓ в поле «Имя переменной»/«Variable name» введите имя переменной `JAVA_HOME`, а в поле «Значение переменной»/«Variable value» – путь к JDK, например `C:\Program Files\Java\jdk-13.0.2`. Нажмите ОК, переменная должна появиться в списке переменных среды.

- ✓ в окне «Системные переменные»/«System Variables» найдите системную переменную `PATH`. Выделите ее и нажмите кнопку «Изменить».

- ✓ в окне «Изменение системной переменной» создайте новую переменную со значением «`%JAVA_HOME%\bin`» (подкаталог `bin` каталога JDK). Нажмите ОК.

4. Проверьте правильность настроек.

- ✓ откройте командную строку (команда `cmd`) и наберите команду `set`. ОС выдаст список установленных переменных окружения с их

значениями. В списке должны присутствовать переменные JAVA_HOME и PATH.

✓ наберите команду: java -version. Должно появиться сообщение о версии Java Runtime Environment (JRE).

После установки и настройки JDK распакуйте архив, содержащий SQL Developer Data Modeler, найдите и запустите файл datamodeler.exe и следуйте инструкциям первого запуска.

При запуске среды моделирования SQL Developer Data Modeler открывается главное окно, которое состоит из двух частей. С левой стороны находится окно навигации Browser для нахождения и выбора объектов. С правой стороны расположено окно проектирования для просмотра информации о выбранных объектах (рисунок 1.2).

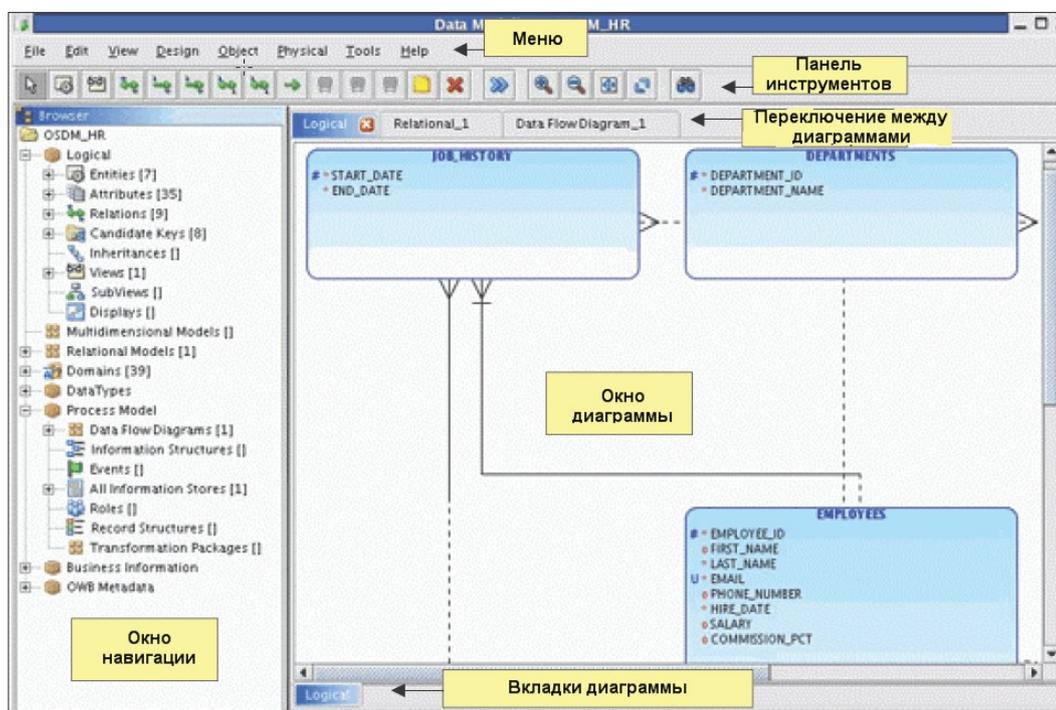


Рисунок 1.2 – Главное окно Oracle SQL Developer Data Modeler

Меню в верхней части окна содержит стандартные пункты, такие как сохранить файл, открыть, закрыть, а также пункты, необходимые для моделирования данных.

На панели инструментов отображаются пиктограммы, действия которых связано с изменением выбранного отображения данных.

SQL Developer Data Modeler позволяет представлять данные в виде логической модели, реляционной модели или схемы потока данных. Для того чтобы увидеть название любой пиктограммы, необходимо навести на нее

указатель мыши. Все действия, связанные с пиктограммами, также доступны из меню объекта окна навигации.

Окно навигации располагается в левой части окна моделирования данных и предназначено для иерархического отображения дерева объектов при моделировании данных.

Окно диаграммы находится справа от окна навигации и предназначено для проектирования диаграммы. Вкладки в верхней части окна панели позволяют перемещаться между разрабатываемыми диаграммами. Вкладки в нижней части каждой диаграммы позволяют выбрать подвид диаграммы в пределах типа диаграммы.

Для настройки SQL Developer Data Modeler под конкретного разработчика и решения определенной задачи используется стандартный набор опций. Для установки опций в пункте меню Tools (инструменты) выберите пункт Preferences (привилегии).

Основные свойства, необходимые для настройки работы в пакете, располагаются в пункте Data Modeler:

- Default Designer Directory. Данное свойство устанавливает директорию, используемую для хранения проектов;
- Default Import Directory. Позволяет установить директорию для импорта дизайна диаграммы;
- Show log after Import. Данный параметр определяет, будет ли показано окно журнала после импорта. В журнал записываются информационные сообщения, предупреждения и ошибки, связанные с импортом моделей;
- Select Relational Models. Опция позволяет отключать диалоговое окно выбора реляционной модели при переходе к вкладке проектирования данных. При отключенной опции все модели подключены;
- Show Properties Dialog on new Object. Свойство позволяет применять свойства существующего объекта к создаваемому.

Пункт Data Modeler -> Model позволяет определить параметры модели данных, например:

- параметр Default RDBMS Type позволяет применять типы данных, определенные в конкретной системе управления базами данных (СУБД);
- параметр Default RDBMS Site позволяет установить размещение узла (распределенной) базы данных;
- параметр Columns & Attributes defaults позволяет устанавливать значения по умолчанию для столбцов и атрибутов, например опция Nulls Allowed разрешает иметь нулевые значения в столбце. При отключенной опции новые столбцы и атрибуты являются обязательными, т.е. требует обязательного ввода значения. В данном параметре возможно указать тип данных по умолчанию при создании объектов модели.

Установки свойств, влияющих на внешний вид диаграммы, реализуется в пункте Data Modeler -> Diagram.

При проектировании модели данных можно установить стандарт для наименования логических объектов, реляционных моделей и доменов в пункте Data Modeler -> Naming Standard. Применённые к модели стандарты наименования можно просматривать, добавлять и изменять в течении всего цикла проектирования. SQL Developer Data Modeler автоматически проверяет, соответствуют ли наименования объектов стандартам, установленным в данном пункте, любые нарушения представляются как ошибки или предупреждения системы проектирования.

1.5. Проектирование диаграммы потока данных

Для создания диаграммы потока данных в пакете SQL Developer Data Modeler необходимо перейти в окне навигации к пункту Process Model -> Data Flow Diagram. Для создания новой диаграммы потока данных необходимо выбрать пункт New Data Flow Diagram правой кнопкой мыши.

Создание процесса в окне проектирования, появившемся справа от окна навигатора, требует выполнения следующих шагов:

1. На панели инструментов нажмите на пиктограмму  «New Process» (новый процесс), а затем щелкните в любом свободном месте окна проектирования. Появится окно свойств процесса – Process Properties;
2. В окне свойств процесса необходимо указать имя процесса и, в случае необходимости, добавить другую информацию о процессе. Изменения будут приняты, после нажатия на кнопку «ОК» объект «Процесс» появится на схеме;

Для внесения изменений в определение процесса дважды щелкните на нем мышкой, окно свойств процесса повторно откроется.

Создание внешней сущности и накопителя данных идентично созданию процесса. Внешнюю сущность можно создать, нажав на пиктограмму  (New External Agents) в панели инструментов. Накопитель данных можно создать, нажав на пиктограмму  (New Information Stores) в панели инструментов.

Для создания потока необходимо выбрать пиктограмму  (New Flows), затем последовательно нажать на объект, из которого хотите организовать исходящий поток информации, и выбрать объект, который будет принимать ваш поток. Доступ к свойствам потока осуществляется двойным щелчком по объекту «поток».

Документирование событий, являющихся причиной возникновения процессов, происходит согласно следующему алгоритму:

1. Откройте окно свойств процесса, которому хотите определить событие, дважды щелкнув по нему кнопкой мыши;
2. В окне слева выберите пункт Events (события);
3. Нажмите на знак "+";
4. Укажите, будет ли это новое событие или оно уже существует.
 - ✓ Если это новое событие, введите его имя и выберите тип события, нажав на кнопку «ОК» подтвердите изменения.
 - ✓ Если это событие уже существует, выберите его из списка событий и нажмите кнопку «ОК».
5. После нажатия на кнопку «ОК» объект «Событие» создается в системе, и в графическое представление процесса добавится круг, указывающий, что данный процесс инициирован событием.

Пакет SQL Developer Data Modeler позволяет создавать процессы трех типов:

- простые;
- составные;
- трансформационные.

Тип процесса указывается в свойствах выбранного процесса и изменяет графическое представление процесса (рисунок 1.3).



Рисунок 1.3 – Три типа процессов: простой, составной, трансформационный (пользовательский и базовый)

Простой процесс – это низкоуровневый автономный процесс, который обычно имеет один вход и один выход и не может быть разделен на подпроцессы. Например, процесс «получить заказ от покупателя» является одной конкретной задачей и не может быть разделен на подпроцессы.

Составной процесс является процессом более высокого уровня. К такому процессу может быть применена декомпозиция, и на более детальном уровне диаграммы потока данных он может быть представлен в виде простых процессов. Примером может служить процесс пересылки, который можно декомпозировать на следующие простые процессы: проверить инвентарь, упаковать груз, уведомить клиента об отправке и т.д.

Для того чтобы увидеть более детальную информацию о процессе, необходимо нажать правой кнопкой мыши на треугольник в верхнем углу и выбрать действие Go To Diagram (перейти к диаграмме), откроется окно с диаграммой нижнего уровня.

Существует ряд рекомендаций для построения составных процессов:

- каждый уровень DFD диаграммы должен быть в своей собственной схеме;
- ни один из уровней не должен иметь более семи процессов. Если необходимо проектировать больше семи, то решением может быть введение в схему дополнительных промежуточных уровней, группирующих простые процессы;
- необходимо убедиться, что все потоки, описанные на верхнем уровне, присутствуют также на нижних уровнях;
- на нижнем уровне возможно добавление дополнительных информационных потоков для представления данных, подверженных преобразованию.

Трансформационный процесс – это процесс, который для своего выполнения требует наличия входных и выходных параметров и набора простых процедур, их последовательного преобразования, которые могут быть объединены в один автономный блок. Примером такого процесса может служить ETL процесс [3], задача которого состоит в извлечении данных из внешних источников, их преобразования по заданным правилам к заданной структуре для загрузки в хранилище данных.

Трансформационный процесс имеет следующие характеристики:

- может быть повторно использован;
- имеет неограниченный уровень декомпозиции;
- имеет интерфейс с входными и выходными параметрами;
- преобразует входные параметры в выходные;
- не взаимодействует с другими процессами на диаграмме.

Для создания трансформационного процесса в пакете SQL Developer Data Modeler необходимо выполнить следующие действия:

- В панели навигации перейдите в пункт Process Model -> Transformation Packages и правой кнопкой мыши выберите New Package (новый пакет);
- В дереве объектов появится новый пакет, щелкните правой кнопкой мыши по нему, выберите New Transformation Task (новая задача преобразования);
- В открывшемся новом окне на панели проектирования можно создать новый трансформационный процесс;

- Необходимо определить информационные потоки, связывающие процесс с накопителями данных;
- В появившихся блоках Input Parameters (входные параметры) и Output Parameters (выходные параметры) необходимо ввести параметры, которые будут существовать в каждом из информационных потоков.

После того, как базовый процесс создан, необходимо вернуться к вкладке разрабатываемой DFD диаграммы и для процесса, определенного как пользовательский трансформационный процесс, определить правило преобразования [10]. Для этого в свойствах процесса, вкладке General (основное), необходимо заполнить поле Use Transformation Task именем базового трансформационного процесса.

Резюме

В данной главе были рассмотрены общие вопросы о моделировании процессов организации, назначении компонентов и составлении диаграмм потоков данных, а также описаны способы проектирования диаграмм с помощью пакета Oracle SQL Developer Data Modeler и его настройка для выполнения конкретных задач.

Контрольные вопросы

1. Какие основные компоненты используются при проектировании DFD?
2. Для чего необходимо определять событие при проектировании диаграммы потока данных?
3. Каково основное назначение процесса?
4. Является ли внешняя сущность частью проектируемой системы?
5. В каких случаях недопустимо создавать потоки между двумя объектами?
6. Где можно изменить основные настройки Oracle SQL Developer Data Modeler?
7. Какой процесс называется составным?
8. Чем принципиально отличается трансформационный процесс от простого и составного?
9. Какие правила необходимо соблюдать для реализации декомпозиции процессов?
10. Может ли быть информационный поток двунаправленным?

2. Логическая модель данных

Для накопления данных, формирования на их основе информации, ее использования в удобном для потребителя виде во всех современных отраслях экономики используются базы данных. При проектировании базы данных на первом этапе создается ее начальный прототип – логическая модель данных. Логическая модель позволяет описать понятия в рамках данной предметной области, взаимосвязь между ними, а также отразить ограничения, связанные с их конкретным применением. Концептуальный выбор логической модели данных предопределяет уровень эффективности той или иной программной реализации базы данных [5].

Логические модели данных можно классифицировать на следующие виды [16]:

- иерархические;
- сетевые;
- реляционные;
- многомерные.

Иерархическая модель данных обеспечивает представление данных в виде иерархической (древовидной) структуры, в которой элементы данных являются узлами дерева, а связи представлены в виде дуг [16]. Иерархическая структура предполагает неравноправие между данными, так как одни данные жестко подчинены другим (рисунок 2.1) [13].

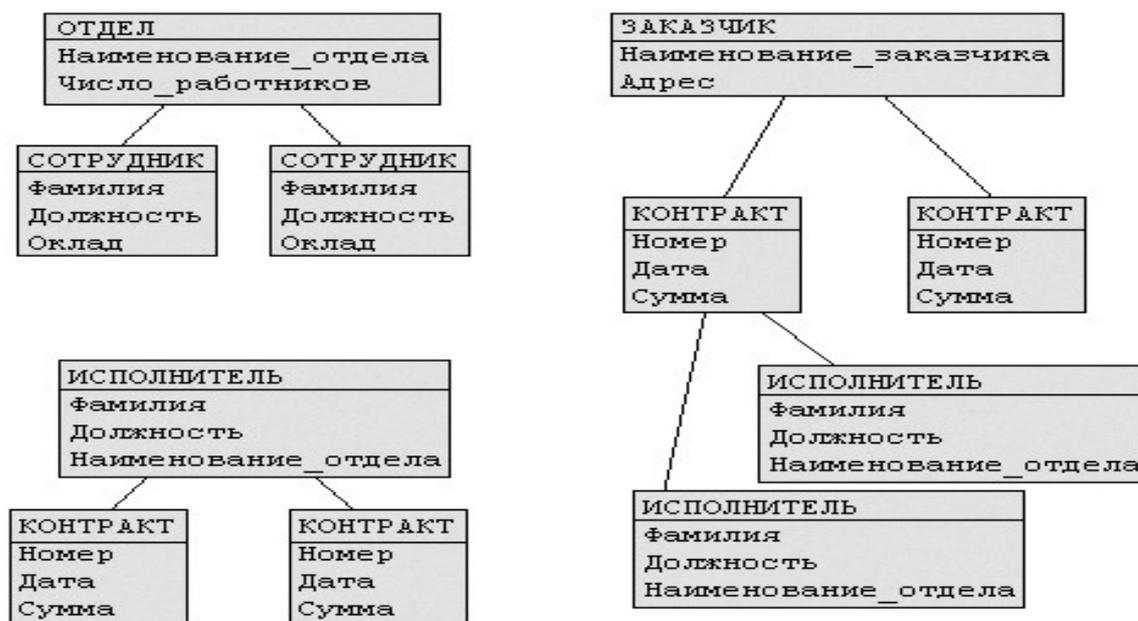


Рисунок 2.1 – Иерархическая структура данных

В иерархической модели данных запись-потомок должна иметь в точности одного предка. Подобные структуры четко удовлетворяют требованиям многих, но далеко не всех реальных задач, так как обладают следующими недостатками: неэффективность, медленный доступ к сегментам данных нижних уровней иерархии, четкая ориентация на определенные типы запросов [16].

Сетевая модель данных является расширением иерархической модели. Разница между иерархической моделью данных и сетевой состоит в том, что сетевая содержит горизонтальные связи помимо вертикальных, это позволяет записи-потомку иметь любое число предков (рисунок 2.2) [16].

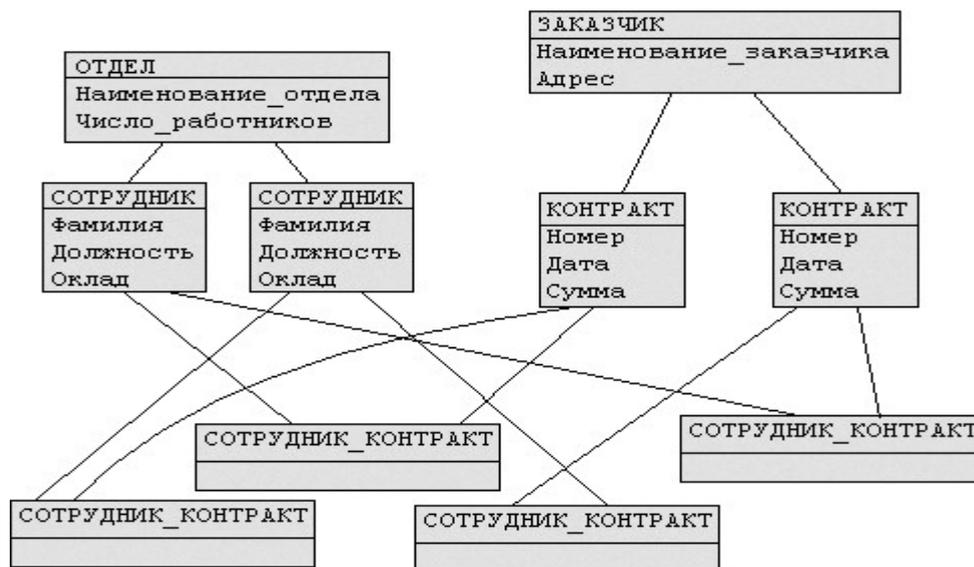


Рисунок 2.2 – Сетевая структура данных

Сетевая модель наследует многие недостатки иерархической модели. Главный из них состоит в необходимости четко определять на физическом уровне связи данных и столь же четко следовать этой структуре связей при формировании запросов к базе [16].

Реляционная модель данных предоставляет средства описания данных на основе только их естественной структуры, т.е. без потребности введения какой-либо дополнительной структуры для целей машинного представления [10].

Представление данных не зависит от способа их физической организации. Это свойство обеспечивается за счет использования математической теории отношений (рисунок 2.3) [16].



Рисунок 2.3 – Реляционная структура данных

Многомерная модель данных – это расширение реляционной модели. В отличие от реляционной модели, где основным понятием является «отношение», в многомерной модели основным понятием является многомерный массив, называемых гиперкубом. В одной базе данных, построенной на многомерной модели, может храниться множество таких кубов, на основе которых можно проводить совместный анализ показателей. Конечный пользователь в качестве внешней модели данных получает для анализа определенные срезы или проекции кубов, представляемые в виде обычных двумерных таблиц или графиков [16].

Многомерные модели данных используют, если целью создания БД является не исполнение транзакций, а анализ данных [4].

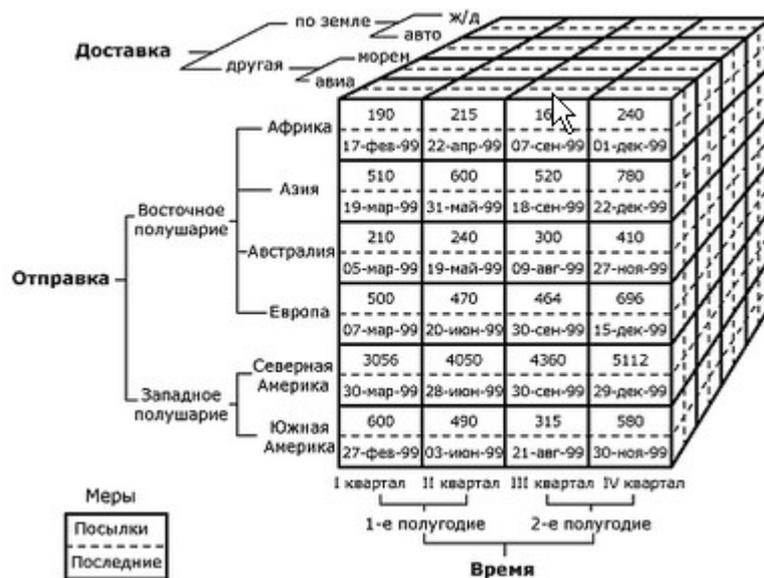


Рисунок 2.4 – Многомерная структура данных

2.1. Диаграмма сущность-связь

Логическое моделирование данных не зависит от аппаратного или программного обеспечения и может включать в себя как графические, так и текстовые компоненты. Графические компоненты помогают наглядно представить блоки данных и отношения между ними, а текстовые компоненты предоставляют более детальную информацию о данных и их связях. Одним из наиболее удобных инструментов унифицированного представления данных, независимого от реализующего его программного обеспечения, является диаграмма "сущность-связь" – ERD (entity – relationship diagram) [16].

Использование диаграммы сущность-связь является полезным механизмом для определения требований к информации. Она легко изменяема и отражает все процессы, происходящие в системе до начала её реализации [16].

Для графического представления ERD в основном используют одну из двух систем обозначений, Баркера или Бахмана. Для построения собственных диаграмм можно использовать любую из этих систем обозначения, так как они обе поддерживаются пакетом Oracle SQL Developer Data Modeler и лишь представляют различные методологии моделирования данных [16].

Все примеры, представленные в этой главе, используют систему обозначений Баркера, которая будет рассмотрена подробно [16].

Основные отличия обозначений Бахмана от обозначений Баркера состоят в следующем:

- сущность представляется прямоугольником без сглаженных углов;
- атрибуты помечаются специальным символом «*», если они не могут содержать нулевые значения. Не существует специального символа для атрибутов, которые могут содержать нулевые значения. Специальный символ «P» устанавливается перед уникальным атрибутом, а символ «F» перед атрибутом, созданным через отношение;
- линия отношения заканчивается стрелкой для максимальной мощности, определенной более чем одним экземпляром сущности. Минимальная мощность обозначается контуром круга или заполненным кругом;
- для каждого атрибута указывается тип данных [16].

Пример диаграммы сущность-связь с использованием обозначений системы Баркера представлена на рис. 2.5 [16].

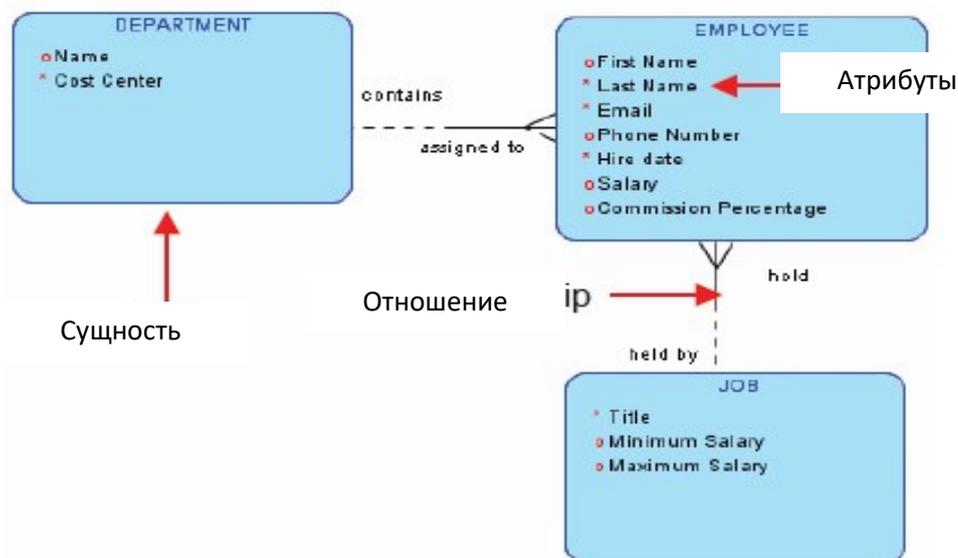
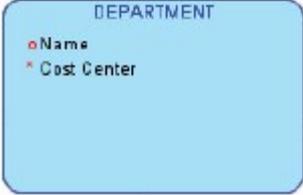
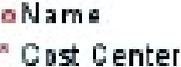


Рисунок 2.5 – ER-диаграмма в системе Баркера.

В таблице 2.1 представлены основные компоненты диаграммы сущность-связь, их назначение и графическое представление в системе Баркера [16].

Таблица 2.1. Компоненты диаграммы сущность-связь

Компонент	Назначение	Графическое представление
Entity – Сущность	Объект или понятие, о котором хранится информация.	
Relationships – Отношения	Естественная ассоциация, которая существует между двумя или более сущностями.	
Attributes – Атрибуты	Описание сущностей и конкретных фрагментов информации, которые должны быть известны.	

2.2. Сущность

Сущность – это некоторый объект, выделяемый (идентифицируемый) пользователем в предметной области, отличающим его от других объектов. Сущности одного и того же типа образуют класс сущности или тип сущности [16].

Необходимо различать такие понятия, как тип сущности (класс сущности) и экземпляр сущности. Понятие тип сущности относится к набору однородных элементов, личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности определяет конкретную единицу из набора. Каждый экземпляр сущности должен быть идентифицирован единственным образом для четкого распознавания среди всех других экземпляров этого типа сущности. Примеры типов и экземпляров сущностей представлены в таблице 2.2 [16].

Таблица 2.2. Тип сущности и экземпляры сущности

Тип сущности	Экземпляр сущности
ЧЕЛОВЕК	Торговый представитель, страховщик, сотрудник, клиентов
МЕСТО	Государство, область, округ, город
ПРЕДМЕТ	инвентарный объект, транспортное средство, продукт
ОРГАНИЗАЦИЯ	Агентство, фирма, отдел
СОБЫТИЕ	Запрос на обслуживание, претензия, выборы

При проектировании ER-диаграммы графический объект «Сущность» представляет собой прямоугольник с указанием наименования сущности в форме существительного прописными буквами. Наименование не должно содержать дефисов и символа подчеркивания [16].

Компонент «Сущность» можно классифицировать на следующие типы:

- первичная сущность – сущность, не зависящая от существования любой другой сущности, например, «Клиент»;
- характеристическая сущность – это сущность, которая зависит от существования другого объекта. Примером такой сущности может служить сущность «Заказ», которая зависит от сущности «Клиент».

- ассоциативная сущность – это сущность, которая зависит от существования двух и более объектов. Примером может служить сущность «Позиция заказа», зависящая от сущностей «Заказ» и «Продукт» [16].

2.3. Атрибут

Атрибут сущности – это характеристика сущности, которая позволяет определить характерное для сущности свойство, и имеет наименование. Атрибуты уточняют, классифицируют, определяют количественное или качественное состояние сущностей, они представляют собой тип описания или детализации информации о сущностях и не являются ими [16].

Атрибуты определяют в пределах прямоугольника, формирующего сущность, с указанием наименования, которое должно быть выражено существительным в единственном числе и являться уникальным [16].

Атрибуты обладают следующими характеристиками:

- атрибуты размещаются в блоке сущности на ERD;
- имена атрибутов должны быть уникальны и начинаться с прописной буквы;
- атрибуты классифицируют сущность, поэтому наименование атрибута не должно содержать в себе наименование сущности.

Например, для сущности «Employee» атрибут с наименованием `employee_phone_number` недопустим, необходимо создать атрибут с именем `phone_number`.

- атрибуты можно определить как «not null», то есть для данного атрибута не допустимы значения null. Для этого необходимо поместить идентификационный символ «*» перед именем атрибута. Для создания атрибута, позволяющего принимать пустые значения «nulls allowed», перед именем необходимо указать идентификационный символ «o» [16].

Множество значений (область определения) атрибута называется доменом. Например, для атрибута ВОЗРАСТ домен (назовем его ЧИСЛО_ЛЕТ) задается интервалом целых чисел больших нуля, поскольку людей с отрицательным возрастом не бывает [16].

2.4. Отношение

Отношения представляют собой бизнес-правила для связи сущностей. Одна сущность может быть связана с другой сущностью или сама с собою.

Отношения позволяют по одной сущности находить другие сущности, связанные с нею. Отношение всегда содержит в себе два правила и графически изображается линией, соединяющей две сущности [16].

На рисунке 2.6 представлен пример отношения между сущностями «DEPARTMENT» и «EMPLOYEE» [16].



Рисунок 2.6 – Отношение между сущностями

Обязательными компонентами отношения являются:

- имя связи;
- мощность отношения.

Имя связи задается рядом с сущностью, к которой данная связь относится, используются только строчные буквы. Имя связи отражает действие, которое необходимо выполнить. В качестве примера на рисунке 2.6 представлены имена связей «содержит» и «назначен» [16].

Мощность определяет минимальное и максимальное количество экземпляров сущностей в отношениях и характеризуется модальностью связи и типом связи [16].

Модальность связи определяет минимальное значение экземпляров сущностей. Каждая связь может иметь одну из двух модальностей связи, представленных на рисунок 2.7 [16].



Рисунок 2.7 – Модальность связи

Модальность «может» показывает, что в отношении двух сущностей экземпляр первой сущности может быть связан с нулём и более экземплярами второй сущности. Таким образом, для модальности «может» допускается наличие null-значения в связи.

Модальность «должен» определяет жесткое отношение между двумя сущностями. Она показывает, что экземпляр первой сущности обязан быть связан с одним и более экземплярами второй сущности.

Связь может иметь разную модальность с разных концов.

Максимальным значение количества экземпляров сущности, участвующих в отношении, может быть единица, тогда на диаграмме связь обозначается в виде линии. При множественной связи между сущностями используется знак \Leftarrow .

Максимальные значения экземпляров сущности для каждого правила отношения, определяется типом связи (рисунок 2.8):

- связь «один к одному» (1:1);
- связь «один ко многим» (1:M) или «многие к одному» (M:1);
- связь «многие ко многим» (M:M);
- связь рекурсивная.

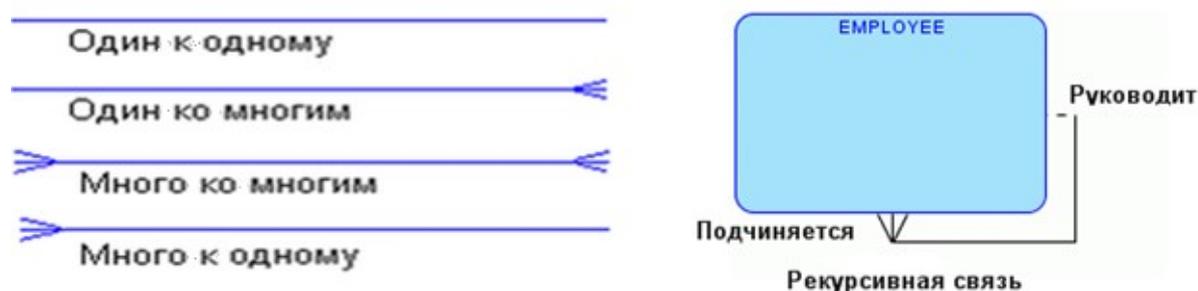


Рисунок 2.8 – Графическое представление отношений с учетом типов связей

Связь типа «один к одному» означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). На диаграмме данный вид связи обозначается в виде простой линии [16]. Наличие связи «один к одному» чаще всего показывает на ошибку проектирования логической модели. Для устранения ошибки проектирования достаточно объединить в одну сущности, участвующие в этом типе связи.

Связь типа «один ко многим» означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи, он представлен на рисунке. 2.6. Первая сущность называется родительской (мощность связи 1), вторая сущность называется дочерней (мощность связи множество). В случае использования связи «многие к одному» несколько экземпляров левой сущности (дочерней) связаны с одним экземпляром правой сущности (родительской). Для обозначения множества экземпляров сущности на диаграмме используется знак \Leftarrow [16].

Связь типа «многие ко многим» означает, что каждый экземпляр одной из сущностей отношения может быть связан с несколькими экземплярами другой сущности этого отношения. Такой тип связи чаще всего используется на верхних уровнях ERD на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа «один ко многим» путем создания промежуточной сущности [16].

Рекурсивная связь появляется в том случае, когда одна и та же сущность выступает неоднократно в разных ролях. Рекурсивная связь также иногда называется одиночной связью. На рисунке 2.8 представлена рекурсивная связь, которая показывает, что любой руководитель является экземпляром сущности «EMPLOYEE» и руководит сотрудниками, которые также являются экземплярами сущности «EMPLOYEE» [16].

Описанный графический синтаксис позволяет однозначно читать диаграммы, пользуясь следующей схемой построения фраз:

<Каждый экземпляр СУЩНОСТИ 1> <МОДАЛЬНОСТЬ СВЯЗИ>
<НАИМЕНОВАНИЕ СВЯЗИ> <ТИП СВЯЗИ> <экземпляр СУЩНОСТИ 2> [16].

Графический синтаксис позволяет читать диаграммы как для связей, определяющих отношение слева направо, так и для связей, определяющих отношение справа налево. Определение связей для рисунка 2.6 с учетом направления чтения:

- Слева направо: «Отдел может содержать одного или несколько сотрудников»;
- Справа налево: «Каждый сотрудник должен быть назначен на должность в один и только один отдел» [16].

Связи между сущностями можно представить не только в виде ERD, но и с помощью матрицы отношений.

Матрица отношений – это матрица, показывающая наличие связей и качество связей между сущностями, представленными в строках матрицы, и сущностями, описанными в столбцах матрицы [16].

Таблица 2.3. Матрица отношений

	ПОКУПАТЕЛЬ	ТОВАР	ЗАКАЗ	СКЛАД
ПОКУПАТЕЛЬ			оформляет	
ТОВАР	оформляется		входит в	храниться
ЗАКАЗ		содержит		
СКЛАД		хранит		

Матрица отношений формируется по следующему принципу:

- все сущности текущего уровня моделирования перечисляются в наименовании столбцов и строк матрицы;
- если сущность, представленная в наименовании строки матрицы, имеет отношение с сущностью, представленной в наименовании столбца матрицы, то в ячейке на пересечении этой строки и столбца записывается наименование связи для этих сущностей;
- если между сущностями не существует отношения, то ячейка матрицы не заполняется;
- если существуют отношения между сущностями, описанные в ячейках выше главной диагонали матрицы, то они зеркально отображаются на ячейки, лежащие ниже главной диагонали, при этом наименование связи обычно меняется, так как меняется направление чтения связи;
- рекурсивные связи задаются в ячейках, находящихся на главной диагонали матрицы [16].

2.5. Уникальный идентификатор

Атрибут или группа атрибутов, которая уникально определяет данную сущность, называется ключом сущности или уникальным идентификатором сущности [16].

Уникальный идентификатор (UID) – это специальный атрибут или группа атрибутов, однозначно идентифицирующая конкретный экземпляр сущности. При проектировании ERD для формирования уникального идентификатора перед атрибутом или группой атрибутов устанавливается специальный символ «#».

Каждый компонент уникального идентификатора должен быть обязательным, поэтому такой компонент не может принимать значение null [16].

Примером уникального идентификатора для сущности EMPLOYEE может быть табельный номер сотрудника (рисунок 2.9) [16].

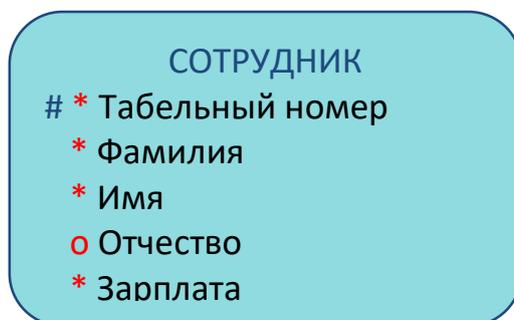


Рисунок 2.9 – Графическое представление сущности «СОТРУДНИК»

Каждая сущность должна иметь как минимум один уникальный атрибут для однозначной идентификации каждого экземпляра сущности, в противном случае будет невозможно однозначно идентифицировать экземпляры [16].

Уникальный идентификатор может быть информационным, когда значения уникального идентификатора имеет значение для бизнес-процесса, например, идентификатор участвует в определении отношения между сущностями, тогда этот уникальный атрибут становится первичным ключом. В случае, когда просто необходимо однозначно определить экземпляры сущности и атрибут является неинформационным для других сущностей, тогда такой атрибут называют синтетическим ключом или уникальным ключом [16].

Объект может иметь более одного уникального идентификатора, например для сущности «СОТРУДНИК» можно задать второй уникальный групповой идентификатор, состоящий из полей «Фамилия», «Имя», «Зарплата». При возникновении такой ситуации необходимо выбрать один из уникальных идентификаторов как первичный и отметить его символом «#», а остальные идентификаторы становятся вторичными (уникальными ключами) и не имеют специального обозначения на ERD, поэтому поля «Фамилия», «Имя», «Зарплата» не имеют специального обозначения на рисунке 2.9 [16].

Если составной уникальный идентификатор является первичным ключом, то все атрибуты, входящие в первичный ключ, на диаграмме отмечаются специальным символом «#». Для сущности «ТЕАТРАЛЬНЫЙ БИЛЕТ» будут отмечены два атрибута («Дата представления», «Номер места»), так как совместно они образуют её первичный ключ (рисунок 2.10) [16].

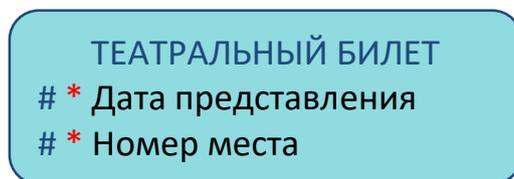


Рисунок 2.10 – Графическое представление сущности «ТЕАТРАЛЬНЫЙ БИЛЕТ»

Если на идентификацию сущности влияют атрибуты, определенные в другой сущности, и между сущностями существует связь, то однозначная идентификация экземпляра сущности изображается вертикальной чертой на линии связи [16].

Две сущности «ACCOUNT» (счет) и «BANK» (банк) представлены на рисунке 2.11. Атрибут «NUMBER», то есть идентификационный номер банка, является уникальным идентификатором и первичным ключом для сущности «BANK». Сущность «ACCOUNT» также содержит в себе атрибут «NUMBER», однако «ACCOUNT» - счет не может существовать без привязки к конкретному банку. Поэтому сущность «ACCOUNT» является зависимой от значения

идентификационного номер банка, что соответствует атрибуту NUMBER для сущности «BANK». Такая связь на диаграмме определяется вертикальной чертой, черта размещается рядом с сущностью, идентификация которой зависит от атрибутов другой сущности.

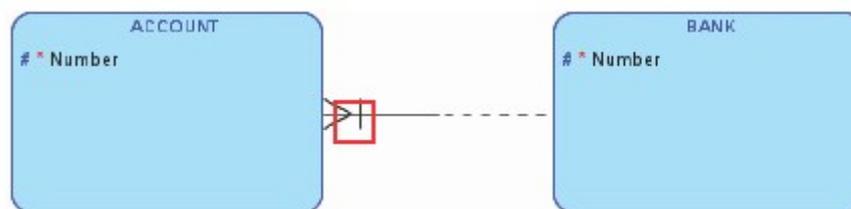


Рисунок 2.11 – Зависимость сущности «ACCOUNT» от атрибута сущности «BANK»

2.6. Использование инструментальное среды для построения логической модели

Инструментальное средство проектирования Oracle SQL Developer Data Modeler позволяет проектировать логические модели с использованием систем Баркера и Бахмана.

Шаги по проектированию логической модели данных, определяющей работу учебного центра, следующие:

1. Закройте все модели, которые в настоящее время открыты. Для этого выберите File → Close.
2. Выберите вкладку «Логическая модель».
3. Выберите пиктограмму создания новой сущности  (New Entity) и нажмите левой кнопкой мыши в любом месте в окне проектирования модели.
4. Введите для сущности имя COURSE и выберите пункт Attributes (атрибуты) слева в окне навигации.
5. Выберите пиктограмму Add (добавить) .
6. Задайте «Course Id» для имени атрибута, выберите флажок Primary UID, для установки первичного уникального ключа, нажмите пиктограмму Add (добавить) .
7. Задайте «Name» для имени атрибута, нажмите пиктограмму Add (добавить) .
8. Задайте «Duration» для имени атрибута, нажмите пиктограмму Add (добавить) .
9. Задайте «Fee» для имени атрибута, нажмите кнопку «ОК».

10. Нажмите в любом пустом месте окна проектирования диаграмм. Выберите пиктограмму создания новой сущности  (New Entity) и нажмите левой кнопкой мыши в любом месте в окне проектирования модели.
11. Введите для сущности имя INSTRUCTOR и выберите пункт Attributes (атрибуты) слева в окне навигации.
12. Выберите пиктограмму Add (добавить) .
13. Задайте «Instructor Id» для имени атрибута, выберите флажок Primary UID, для установки первичного уникального ключа, нажмите пиктограмму Add (добавить) .
14. Задайте «Name» для имени атрибута, нажмите пиктограмму Add (добавить) .
15. Задайте «Street» для имени атрибута, нажмите пиктограмму Add (добавить) .
16. Задайте «City» для имени атрибута, нажмите пиктограмму Add (добавить) .
17. Задайте «State» для имени атрибута, нажмите пиктограмму Add (добавить) .
18. Задайте «Postal Code» для имени атрибута, нажмите пиктограмму Add (добавить) .
19. Задайте «Phone Number» для имени атрибута, нажмите кнопку «ОК».
20. Нажмите в любом пустом месте окна проектирования диаграмм. Выберите пиктограмму создания новой сущности  (New Entity) и нажмите левой кнопкой мыши в любом месте в окне проектирования модели.
21. Введите для сущности имя STUDENT и выберите пункт Attributes (атрибуты) слева в окне навигации.
22. Выберите пиктограмму Add (добавить) .
23. Задайте «Student Id» для имени атрибута, выберите флажок Primary UID, для установки первичного уникального ключа, нажмите пиктограмму Add (добавить) .
24. Задайте «Name» для имени атрибута, нажмите пиктограмму Add (добавить) .
25. Задайте «Street» для имени атрибута, нажмите пиктограмму Add (добавить) .

26. Задайте «City» для имени атрибута, нажмите пиктограмму Add (добавить) .

27. Задайте «State» для имени атрибута, нажмите пиктограмму Add (добавить) .

28. Задайте «Postal Code» для имени атрибута, нажмите пиктограмму Add (добавить) .

29. Задайте «Phone Number» для имени атрибута, нажмите кнопку «ОК».

Сущности и их атрибуты созданы. Теперь необходимо создать отношения между ними:

30. Выберите пиктограмму создания нового отношения 1:M .

31. Нажмите на сущность INSTRUCTOR, а затем на сущность COURSE.

32. Дважды щелкните на отношение, которое было создано.

33. Выберите свойство Cardinality (мощность) слева в окне навигации.

34. Введите «teaches» для поля «Name on Source» (имя источника) и «taught by» для поля «Name on Target» (имя цели), и нажмите кнопку «ОК».

35. Если имена отношения не появились на диаграмме, выполните следующие действия:

- выберите Tools → General Options (общие параметры);
- разверните пункт Diagram (диаграмма) и выберите Logical Model (логическая модель);
- выберите флажок Show Source/Target Name и нажмите кнопку «ОК».

36. Создайте еще одно отношение между двумя сущностями COURSE и STUDENT. Выберите пиктограмму для создания нового отношения типа M:N .

37. Нажмите на сущность COURSE, а затем на сущность STUDENT.

38. Дважды щелкните на отношение, которые было создано.

39. Выберите свойство Cardinality (мощность) слева в окне навигации.

40. Введите «taken by» для поля «Name on Source» (имя источника) и «enrolled in» для поля «Name on Target» (имя цели), и нажмите кнопку «ОК».

Для того чтобы создать подвид для сущностей, выделите все объекты на диаграмме и щелкните правой кнопкой мыши по сущности, выберите пункт Create SubView from selected (создать подвид из выбранных объектов).

Возможно, придется переместить некоторые сущности для того, чтобы максимизировать пространство на экране и свести к минимуму пересечение линий отношений. Инструментальная среда Oracle SQL Developer Data Modeler позволяет перемещать объекты модели, также делать линии прямыми или

ломаными. Чтобы использовать возможность применения ломаных линий, щелкните правой кнопкой мыши по сущности, отключите свойство Auto Route.

Резюме

Прототип проектируемой базы данных определяет логическая модель. Она не зависит от будущей физической реализации базы данных. Целью логического моделирования является определение сущностей, их атрибутов и связей между ними, для понимания движения данных, их анализа и возможности получения необходимой итоговой информации. Диаграмма сущность-связь является графическим представлением логической модели, для ее построения можно использовать различные системы обозначения. Современные инструментальные средства разработки позволяют быстро и детализировано разрабатывать модели данных и поддерживают различные системы обозначения.

Контрольные вопросы

1. Какие типы логических моделей данных существуют?
2. Для чего используется диаграмма сущность-связь?
3. Какие системы обозначения используются для проектирования ER-диаграмм?
4. Какие типы связей существуют между сущностями?
5. Для чего используется матрица отношений?
6. В чем различие между типом сущности и экземпляром сущности?
7. Какими характеристиками обладают атрибуты?
8. Что характеризует мощность отношения?
9. Для чего используется уникальный идентификатор?
10. Может ли сущность содержать несколько уникальных идентификаторов?

3. Теория нормальных форм

Нормализация является реляционной концепцией базы данных. Если корректно разработана ERD, то таблицы базы данных, созданные с помощью пакета Oracle SQL Developer Data Modeler, будут соответствовать правилам нормализации [16].

Основу нормальных форм образуют функциональные зависимости. Функциональная зависимость – это связь типа «многие к одному» между двумя множествами атрибутов рассматриваемого отношения. В отношении могут существовать функциональные зависимости, образуемые атрибутами первичного ключа, а также атрибутами, которые не связаны с первичным ключом, т.е. атрибутами потенциальных ключей. Целью нормализации является устранение в отношении зависимостей, которые образованы потенциальными ключами.

Каждое формальное правило нормализации реляционной базы данных имеет соответствующую модель интерпретации данных.

Существуют следующие виды интерпретации данных:

- первая нормальная форма (1НФ). Все атрибуты должны быть однозначными;
- вторая нормальная форма (2НФ). Все не ключевые атрибуты должны полностью зависеть от уникального идентификатора сущности;
- третья нормальная форма (3НФ). Атрибут, не имеющий уникальный идентификатор, не может зависеть от другого атрибута, не имеющего уникальный идентификатор;
- нормальная форма Бойса-Кодда (НФБК). Любой атрибут, от которого полностью функционально зависит некоторый другой атрибут, может являться ключом;
- четвертая нормальная форма (4НФ). В случае существования многозначной зависимости между сущностями атрибуты, не участвующие в зависимости, функционально зависят от главной сущности;
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или PJ/NF). Любая зависимость соединения в отношении следует из существования некоторого возможного ключа в отношении [16].

Преимущества нормализации:

- нормализация гарантирует, что каждый атрибут принадлежит соответствующей сущности, которой он был назначена, а не другой сущности.

- нормализация устраняет избыточное хранение информации. Это упрощает логику приложения, потому что разработчикам не нужно думать о нескольких копиях одной и той же части информации.
- нормализация гарантирует, что существует один атрибут в одном месте, с одним именем, с одним значением, в любой момент времени [16].

Данные, которые не были «нормализованы», считаются «ненормализованными» данными. Ненормализованные данные появляются тогда, когда при разработке базы данных не была предварительно построена ERD диаграмма. Примером ненормализованных данных может служить бланк заказа товара на рисунке 3.1 [16].

<u>Ordered by:</u>		<u>Ship to:</u>		Order ID
Customer ID	: <input type="text"/>	Ship Via	: <input type="text"/>	Order Date
Customer Name	: <input type="text"/>	Name	: <input type="text"/>	
Address Line 1	: <input type="text"/>	Address Line 1	: <input type="text"/>	
Address Line 2	: <input type="text"/>	Address Line 2	: <input type="text"/>	
Address Line 3	: <input type="text"/>	Address Line 3	: <input type="text"/>	
City, State ZIP	: <input type="text"/> , <input type="text"/> <input type="text"/>	City, State ZIP	: <input type="text"/> , <input type="text"/> <input type="text"/>	

<u>Item ID</u>	<u>Color</u>	<u>Size</u>	<u>Quantity</u>	<u>Description</u>	<u>Price</u>

Order Total:

Рисунок 3.1 – Бланк заказа товара

3.1. Первая нормальная форма

Первая нормальная форма гарантирует, что каждый атрибут имеет одно значение для каждого экземпляра сущности. Не должно быть атрибутов, которые имеют несколько значений для экземпляра сущности. В примере, представленном на рисунке 3.1, поля «Item ID», «Color», «Size», «Quantity», «Description», «Price» имеют более одного значения для поля «Customer ID», так как покупатель может приобрести несколько позиций товара в рамках одного заказа, поэтому бланк заказа на товар не является первой нормальной формой [16].

Для преобразования ненормализованных данных к первой нормальной форме необходимо сначала представить все данные бланка заказа в виде

сущности, в которой имена атрибутов являются столбцами таблицы, а в строках размещаются экземпляры сущности, то есть конкретные данные по каждому заказу. Для преобразования данных из формы заказа в таблицу необходимо:

- определить, какие поля формы заказа будут являться атрибутами;
- сгруппировать все атрибуты в одной сущности;
- определить, какой атрибут будет служить в качестве первичного ключа [16].

Такая сущность не будет являться первой нормальной формой, так как она будет содержать дублирующие записи. Для того чтобы преобразовать данную сущность к первой нормальной форме, необходимо выполнить следующие действия (рисунок 3.2):

- создать дополнительную сущность и переместить атрибуты, влияющие на дублирование значений;
- создать для новой сущности первичный ключ, если требуется добавить дополнительные атрибуты к ключу;
- создать отношение типа 1:М между двумя сущностями [16].

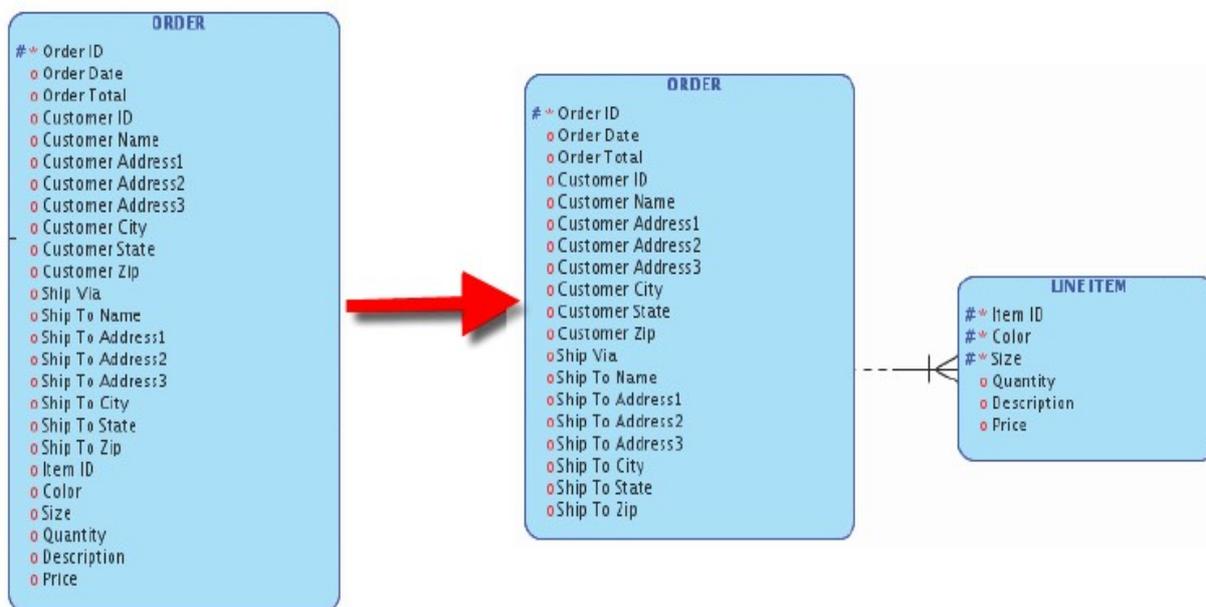


Рисунок 3.2 –Первая нормальная форма

3.2. Вторая нормальная форма

Вторая нормальная форма гарантирует, что каждый атрибут зависит от уникального идентификатора сущности, то есть каждый конкретный экземпляр содержит значения атрибутов, характерные только для него и определяемые

значением уникального идентификатора. Первичный уникальный идентификатор является первичным ключом. Первичный ключ может включать в себя несколько атрибутов, в таком случае он называется составным первичным ключом [16]. Неключевые атрибуты функционально полно зависят от составного ключа, если они функционально зависят от всего ключа в целом, но не находятся в функциональной зависимости от какого-либо из входящих в него атрибутов [8].

Для преобразования модели ко второй нормальной форме (2НФ) необходимо выбрать из сущностей атрибуты, которые зависят от части первичного ключа, но не зависят от полностью определенного ключа. Для выбранных атрибутов необходимо создать новые сущности [16].

Для преобразования модели из рисунка 3.2 ко второй нормальной форме необходимо выполнить следующие действия:

- определить сущности, имеющие составной ключ;
- переместить атрибуты, которые зависят только от части составного ключа, в новую сущность;
- переместить в новую сущность атрибуты составного ключа, которые однозначно идентифицируют неключевые атрибуты. Определить эти атрибуты как первичный ключ для новой сущности [16].

На рисунке 3.3 представлена вторая нормальная форма. Атрибуты «Description» и «Price» зависят от атрибута «Item ID», являющегося частью составного ключа, поэтому они перемещены в новую сущность «ITEM». Первичным ключом для сущности «ITEM» является атрибут «Item ID» [16].

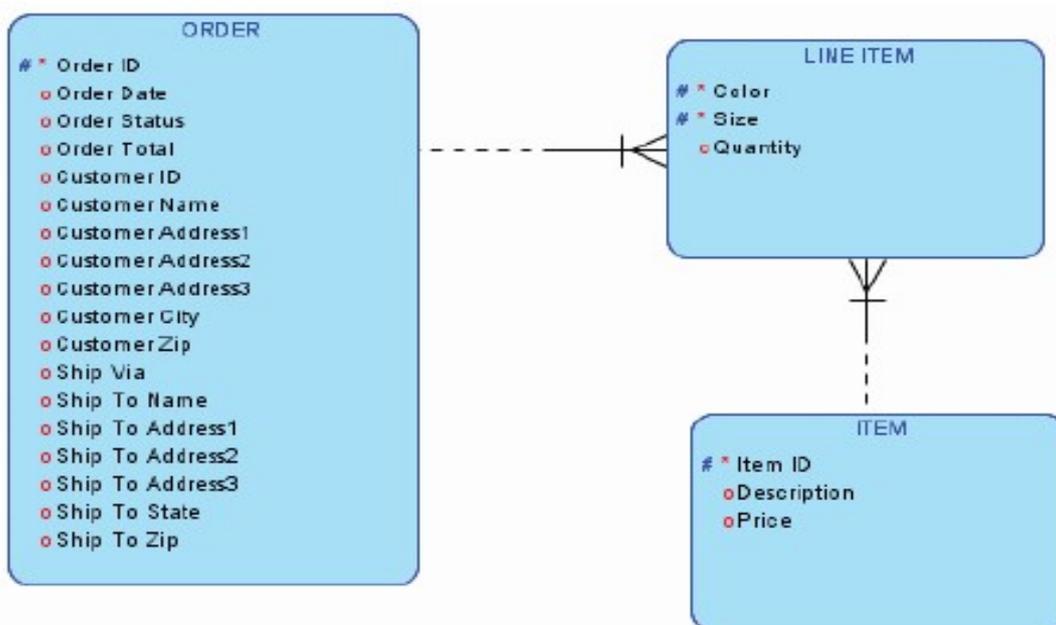


Рисунок 3.3 – Вторая нормальная форма

3.3. Третья нормальная форма

Третья нормальная форма гарантирует, что каждый атрибут зависит только от уникального идентификатора своей сущности [16].

Для преобразования модели к третьей нормальной форме (3НФ), необходимо удалить из сущности атрибуты, которые напрямую не относятся к первичному ключу данной сущности, а зависят от других атрибутов этой сущности. Для преобразования модели данных к третьей нормальной форме необходимо выполнить следующие действия:

- найти атрибуты, которые напрямую не зависят от первичного ключа;
- переместить атрибуты, не зависящие напрямую от первичного ключа, в новую сущность;
- определить ключ для новой сущности, который однозначно определяет все атрибуты [16].

В примере на рисунке 3.4 сформирована новая сущность «CUSTOMER», содержащая атрибуты «Customer ID», «Name», «Address1», «Address2», «Address3», «City», «State», «Zip». Эти атрибуты не зависели напрямую от первичного ключа «Order ID», так как даже если пользователь не сделал ни разу заказ, его данные могут храниться в БД, если он заполнил форму регистрации клиента. Без применения третьей нормальной формы хранение информации о клиенте, который еще не оформлял заказ, было бы невозможно. В качестве первичного ключа для новой сущности выбран атрибут «Customer ID» [16].

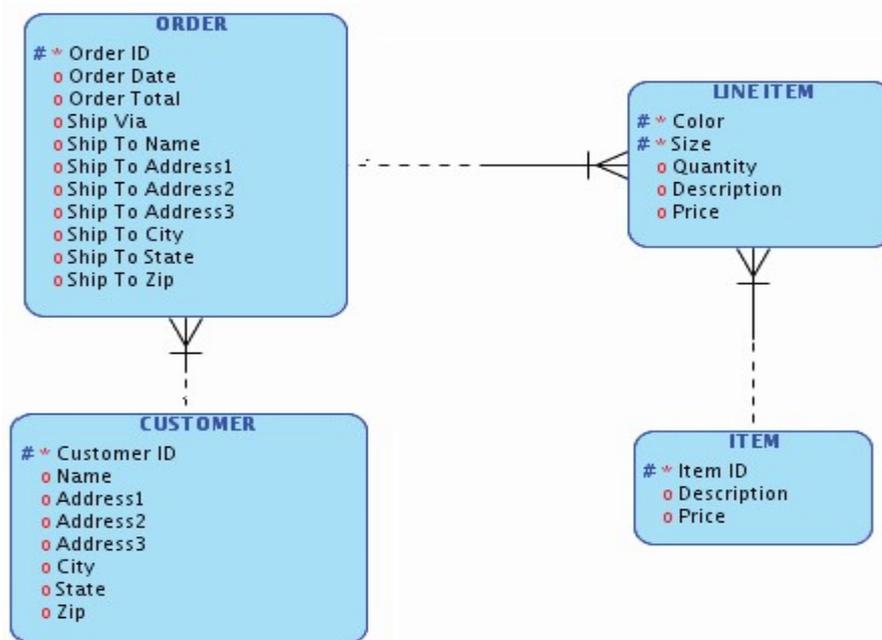


Рисунок 3.4 – Третья нормальная форма

3.4. Нормальная форма Бойса-Кодда

Часто при приведении модели данных к одной из нормальных форм возникает ситуация, когда часть одного ключа входит в состав другого ключа, если в отношении существуют первичный и потенциальные ключи. В этом случае возникает проблема приведения отношений к той или иной форме. Для таких ситуаций сформулирована нормальная форма Бойса-Кодда.

Модель находится в нормальной форме Бойса-Кодда (НФБК), если модель удовлетворяет условиям 3 нормальной формы и любой атрибут, от которого полностью функционально зависит некоторый другой атрибут, может являться ключом.

Потенциальный ключ – это атрибут или множество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (атомарности).

В отношении может быть одновременно несколько потенциальных ключей. В этом случае один из них выбирается в качестве первичного ключа отношения, а другие потенциальные ключи становятся альтернативными ключами или уникальными идентификаторами, которые рассмотрены в главе 2.5.

Все потенциальные ключи равно пригодны в качестве первичного ключа. Обычно в качестве первичного ключа выбирают тот ключ, который имеет меньший размер физического хранения и/или включает меньшее количество атрибутов.

Рассмотрим в качестве примера горнолыжный курорт, имеющий несколько подъемников, доставляющих отдыхающих на различную высоту. Существует функциональная зависимость «ТАРИФ» → «НОМЕР ПОДЪЕМНИКА».

Таблица 3.1. Экземпляры сущности «ТАРИФ НОМЕР ПОДЪЕМНИКА»

№ подъемника	Время начала	Время окончания	Тариф
1	08:30	9:30	«Эконом»
1	12:30	13:30	«Эконом»
1	09:30	12:30	«Стандарт»
2	08:30	09:30	«Премиум-В»
2	09:30	13:30	«Премиум-А»
2	14:00	16:30	«Премиум-В»

Рассматриваемое отношение обладает следующими ограничениями:

- Каждый тариф можно применять только к одному подъемнику.
- Каждый подъемник может иметь несколько тарифов.
- Каждому периоду работы соответствует определенный подъемник с конкретным тарифом.

Рассматриваемое отношение находится в третьей нормальной форме, но не в нормальной форме Бойса – Кодда, так как содержит перекрывающиеся ключи: «ТАРИФ-НОМЕР ПОДЪЕМНИКА» и «ТАРИФ-ПЕРИОД».

Недостатком данной структуры является следующее: по ошибке можно приписать тариф «Эконом» к подъемнику 2, который гораздо длиннее подъемника 1, поэтому экономный тариф к нему применяться не может.

Для перехода от ЗНФ к НФБК необходимо детерминанты, не являющиеся потенциальными ключами, выделить в отдельные отношения, что продемонстрировано в таблицах 3.2 и 3.3.

Таблица 3.2. Экземпляры сущности «ПОДЪЕМНИК ТАРИФ»

Тариф	№ подъемника	Участник бонусной программы
«Эконом»	1	Да
«Стандарт»	1	Нет
«Премиум-А»	2	Да
«Премиум-В»	2	Нет

Таблица 3.3. Экземпляры сущности «ТАРИФ ПЕРИОД»

Тариф	Время начала	Время окончания
«Эконом»	08:30	9:30
«Эконом»	12:30	13:30
«Стандарт»	09:30	12:30
«Премиум-А»	09:30	13:30
«Премиум-В»	08:30	9:30
«Премиум-В»	14:00	16:30

Ситуация, когда отношение будет находиться в ЗНФ, но не в НФБК, возникает при условии, что отношение имеет два (или более) возможных ключа, которые являются составными и имеют общий атрибут. На практике такая ситуация встречается достаточно редко, поэтому для всех прочих отношений ЗНФ и НФБК эквивалентны.

3.5. Четвертая нормальная форма

Четвертая нормальная форма касается отношений, в которых имеются повторяющиеся наборы данных. Декомпозиция, основанная на функциональных зависимостях, не приводит к исключению такой избыточности. В этом случае используют декомпозицию, основанную на многозначных зависимостях [14].

Многозначная зависимость является обобщением функциональной зависимости и рассматривает соответствия между множествами значений атрибутов [14].

Для примера четвертой нормальной формы рассмотрим сущность «СОТРУДНИК» ИТ-компания, атрибутами которой являются «Имя», «Проект», «Документация».

Таблица 3.4. Экземпляры сущности «СОТРУДНИК»

Имя	Проект	Документация
А	Back-end	Документация по Back-end
А	Front-end	Документация по Back-end
А	Back-end	Документация по Front-end
А	Front-end	Документация по Front-end
В	БД	Документация по БД
В	БД	Документация по Workflow
добавление информации		
В	Back-end	Документация по БД
В	Back-end	Документация по Workflow

Из сущности «СОТРУДНИК» можно получить информацию о проектах, в которых он занят, и написанных им инструкциям. Пусть сотрудник А занят в проекте по разработке «Front-end» и по разработке «Back-end». Он написал соответствующую документацию «Документация по Back-end» и «Документация по Front-end». Сотрудник В занят в проекте по разработке БД и является автором «документации по БД» и по управлению потоком работ «Документация по Workflow». Отношение, описанное выше, представлено в таблице 3.4.

Из представленной таблицы очевидно, что экземпляры сущности имеют значительную избыточность. Это приводит к аномалии обновления. Например, если добавить информацию о том, что сотрудник В будет также участвовать в проекте по разработке back-end, то данная ситуация приводит к необходимости добавить 2 строки в таблицу вместо одной, по одной для каждой написанной им документации.

Аномалия обновления исчезает при замене сущности «СОТРУДНИК» двумя сущностями: «СОТРУДНИК_ПРОЕКТ» (таблица 3.5) и «СОТРУДНИК_ДОКУМЕНТАЦИЯ» (таблица 3.6).

Таблица 3.5. Отношение «СОТРУДНИК ПРОЕКТ»

Имя	Проект
А	Back-end
А	Front-end
В	БД
В	Back-end

Таблица 3.6. Отношение «СОТРУДНИК ДОКУМЕНТАЦИЯ»

Имя	Документация
А	Документация по Back-end
А	Документация по Front-end
В	Документация по БД
В	Документация по Workflow

В представленном примере аномалия обновления возникает из-за того, что в сущности «СОТРУДНИК» определены следующие зависимости:

- множество значений атрибута «Проект» от множества значений атрибута «Имя»;
- множество значений атрибута «Документация» от множества значений атрибута «Имя».

3.6. Пятая нормальная форма

Во всех предыдущих формах единственной операцией, необходимой для устранения избыточности сущности, была декомпозиция ее на две сущности. В некоторых случаях декомпозиция на две сущности вызывает потери, которые, однако, можно избежать, если к существующей сущности применить декомпозицию на три и более сущности. Такие зависимости называются зависимости по соединению, а такие сущности называют 3-декомпозируемые сущности или n-декомпозируемые, если применяется декомпозиция более, чем на 3.

Основная характеристика зависимости по соединению – аномалия обновления. Такие отношения не являются ни многозначными, ни функциональными, поэтому требуется введение пятой нормальной формы.

Сущность находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в ней определяется только его возможными ключами. То есть каждая новая сущность, полученная при декомпозиции, содержит не менее одного возможного ключа и не менее одного неключевого атрибута [7].

3.7. Преобразование отношений

На практике при реализации БД отношение «многие ко многим» не используются, они заменяются двумя отношениями «многие к одному».

Для преобразования М:М отношения необходимо определить атрибуты, которые могут быть включены в новую сущность, являющуюся пересечением существующих. В примере на рисунке 3.5 отношение М:М между сущностью «ORDER» (заказ) и «PRODUCT» (продукт) может быть преобразовано, если создать новую сущность «ORDER ITEM», определяющую позицию заказа.

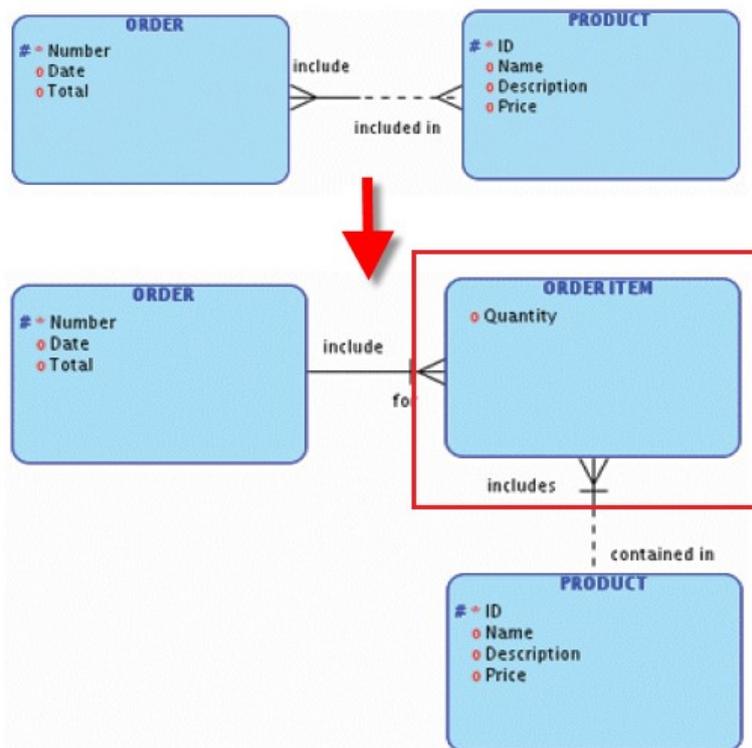


Рисунок 3.5 – Преобразование отношения «многие ко многим»

Уникальным идентификатором в этой сущности будет атрибут, устанавливающий связь между сущностями «ORDER» и «PRODUCT».

Новая сущность обладает следующими характеристиками:

- отношения со стороны новой сущности должны быть всегда обязательными;
- содержит атрибуты, определяющие количество или время;
- определяется двумя исходящими отношениями (с указанием отношений).

Если при моделировании в инструментальной среде SQL Developer Data Modeler у сущностей, имеющих отношение M:M, нет дополнительных атрибутов, которые можно перенести в новую сущность, то на диаграмме можно оставить отношение M:M между сущностями. При формировании реляционной модели SQL Developer Data Modeler создаст таблицу пересечений в реляционной модели самостоятельно.

Для моделирования иерархических данных обычно используют множество отношений типа M:1. Примером иерархических отношений может быть компания, состоящая из одной или нескольких подразделений. Каждое подразделение может состоять из одного или нескольких отделов. Каждый отдел может состоять из одной или более рабочих групп.

Уникальный идентификатор верхнего уровня для набора иерархических сущностей может передаваться через множество отношений, чтобы однозначно

определить экземпляры сущностей нижних уровней иерархии. Это приводит к дублированию атрибутов. Решением данной проблемы может быть применение рекурсивной связи для сущности, в которой определены все атрибуты компании.

На рисунке 3.6 представлена рекурсивная связь для иерархических данных, используя сущность «ORGANIZATION». Рекурсивная связь не может быть обязательной. Если бы каждая организация должна была входить в другую организацию, то иерархия организаций была бы бесконечно большой. Чтобы этого не происходило, рекурсивное отношение должно быть необязательным в обоих направлениях. Кроме того, могут существовать многие организации одного и того же типа, для этого диаграмма дополняется сущностью «ORGANIZATION TYPE», чтобы хранить имя и идентификатор организации.

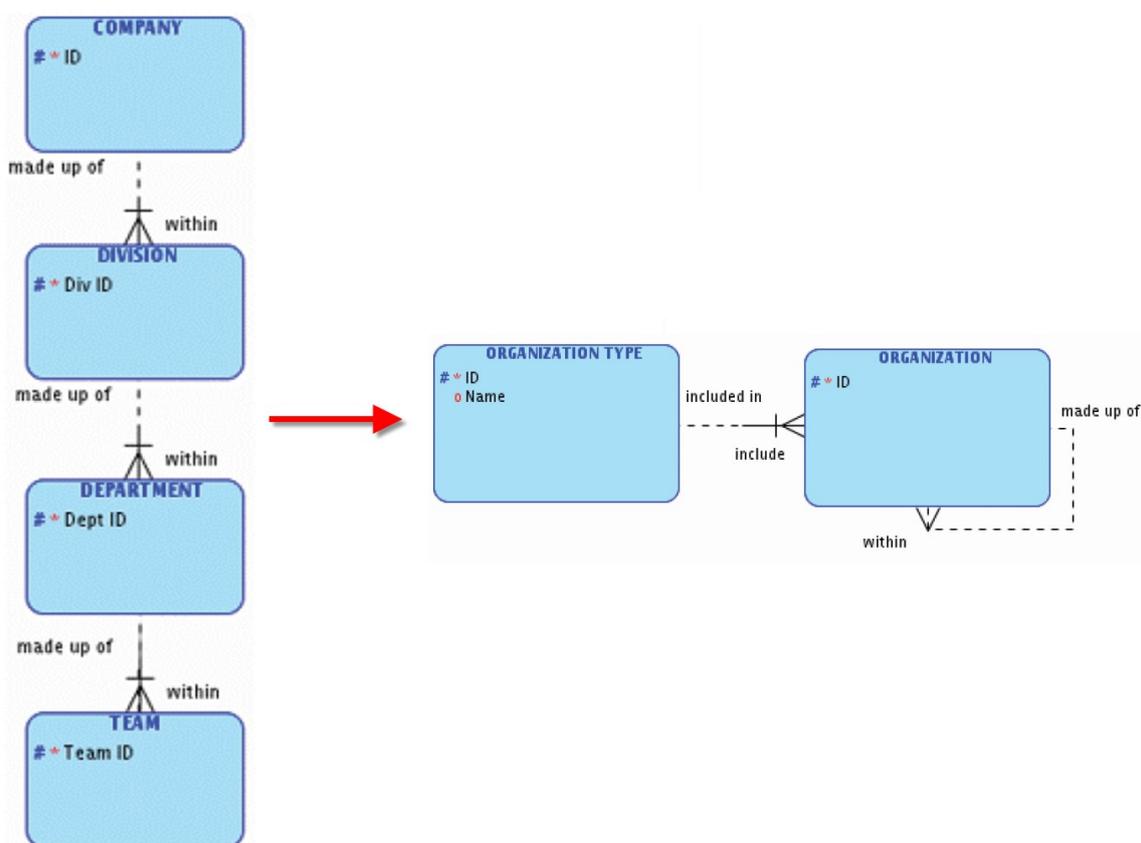


Рисунок 3.6 – Рекурсивная связь для иерархических данных

Взаимоисключающее отношение – это такое отношение, при котором сущность имеет связь либо с сущностью А, либо с сущностью В. Оба отношения могут быть действительными, но в разные моменты времени. Для изображения таких отношений на ER-диаграмме используют дуги, пересекающиеся входящие в дуги взаимоисключающие связи.

На рисунке 3.7 изображено взаимоисключающее отношение между сущностью «BANK ACCOUNT» и сущностями «INDIVIDUAL», «COMPANY».

Каждый банковский счет должен принадлежать одному и только одному физическому лицу или должен принадлежать одной и только одной компании.

Взаимоисключающие отношения обладают следующими характеристиками:

- все концы связей в дуге должны быть либо обязательными, либо не обязательными;
- связь может входить только в одну дугу;
- количество связей в дуге может быть любым;
- связи в дуге часто имеют одинаковые имена;
- связи, входящие в дуги, должны идти от одного и того же класса объектов.

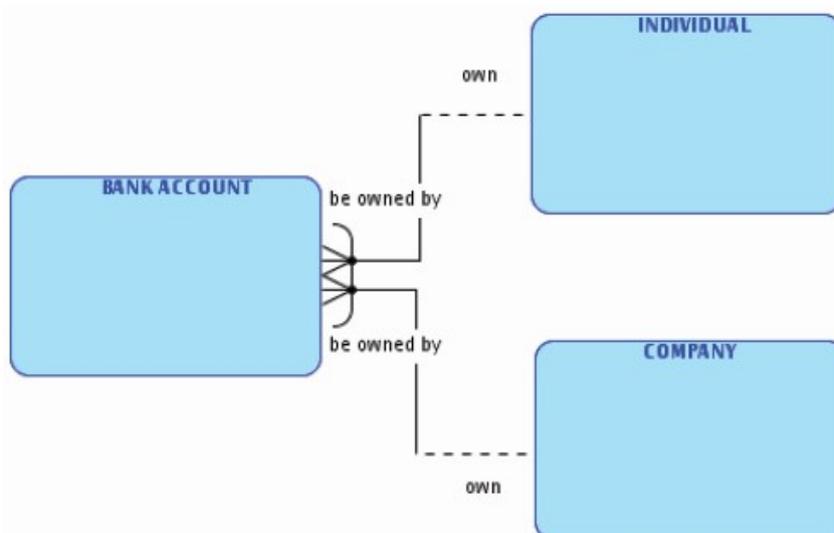


Рисунок 3.7 – Взаимоисключающие отношения

Резюме

Хранение данных в базе данных осуществляется в реляционных таблицах, а доступ к ним обеспечивается с помощью операторов языка SQL. Но для того, чтобы преобразовать ненормализованные данные в таблицу, необходимо соблюдать основные правила преобразования ненормализованных данных, которые описаны в первой, второй и третьей нормальных формах. Для реляционной базы данных не требуется приводить данные к четвертой и пятой нормальной форме, однако о существовании таких форм необходимо помнить. В реляционной базе данных не реализуются отношения «многие ко многим», они заменяются двумя отношениями «многие к одному», а иерархические отношения заменяются рекурсивной связью.

Контрольные вопросы

1. В какую форму должны быть преобразованы данные, чтобы быть пригодными для обработки в реляционной базе данных?
2. Сколько значений может иметь каждый атрибут в случае преобразования ненормализованных данных в первую нормальную форму?
3. Может ли первичный уникальный идентификатор состоять из нескольких атрибутов?
4. Что декларирует нормальная форма Бойса-Кодда?
5. Может ли в третьей нормальной форме атрибут зависеть от уникального идентификатора сущности, в которой он не определен?
6. Что представляет собой процесс нормализации базы данных, из каких этапов он состоит?
7. Почему тип связи «многие ко многим» не реализуем в реляционной базе данных?
8. В каком случае используются взаимоисключающие отношения между сущностями?
9. В чем состоит преимущество нормализованных данных?
10. Какую последовательность действий необходимо выполнить для преобразования модели ко второй нормальной форме?

4. Реляционная и физическая модели данных

Логическая модель данных описывает данные, необходимые для реализации бизнес-процессов. Эта модель используется на ранних стадиях разработки проекта, поэтому она должна быть полностью независимой от любой реализации БД. Логическая модель данных используются в качестве основы для реализации любого типа систем управления базами данных (СУБД) или файловых систем [16].

Логической моделью данных является высокоуровневое представление, которое реализовано с помощью формального языка и которое обеспечивает однозначную трактовку и концептуально «работоспособное» решение [16]. Детальная проработка логической модели данных приводит к снижению временных, технических и материальных затрат на этапе физической реализации структуры базы данных.

4.1. Реляционная модель данных

Процесс проектирования баз данных часто называют созданием реляционной модели. Реляционная модель данных – модель данных, которая впервые была предложена британским учёным, сотрудником компании IBM Эдгаром Франком Коддом (E.F.Codd) в 1970 году в статье «A Relational Model of Data for Large Shared Data Banks» [10]. В настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД. В реляционной модели достигается гораздо более высокий уровень абстракции по сравнению с иерархической или сетевой моделями. В упомянутой статье Э.Ф. Кодда утверждается, что «реляционная модель предоставляет средства описания данных на основе только их естественной структуры, т.е. без потребности введения какой-либо дополнительной структуры для целей машинного представления» [10]. Таким образом, представление данных не зависит от способа их физической организации, а обеспечивается за счет использования математической теории отношений [16].

Реляционную модель строят на стадии проектирования базы данных. Модель обеспечивает описание объектов базы данных, которые должны быть созданы в процессе генерации БД [16].

На рисунке 4.1 представлена реляционная модель, состоящая из двух объектов: таблица DEPARTMENTS (отделы) и таблица EMPLOYEES (сотрудники). Каждая таблица содержит первичный ключ. В таблице EMPLOYEES существует два внешних ключа [16]. Первый ключ ссылается на

Парадигма реляционной модели данных состоит в представлении данных в виде двумерных таблиц. Таблицы состоят из нескольких строк, каждая строка – из набора столбцов. В таблице все строки имеют одинаковую структуру, определенную столбцами, однако в некоторых строках может не существовать значений отдельных столбцов [16].

Таким образом, основными компонентами реляционной модели являются:

- таблица – простая структура, обеспечивающая организацию и хранение информации. Таблица состоит из столбцов и строк. Каждый столбец используется для хранения конкретного типа значения. В таблице 4.1 представлена сущность EMPLOYEES, предназначенная для хранения информации о сотрудниках в виде таблицы.
- строка. Каждая строка описывает нового работника. В таблице 4.1 каждая строка описывает в полном объеме все свойства, определенные в реляционной модели на рисунке 4.1.
- столбец. Каждый столбец содержит информацию определенного типа: ID (идентификатор), Name (имя), Address (адрес), Birth_date (дата рождения), Dept_ID (идентификатор департамента), которая записывается для каждого работника.
- первичный ключ – столбец или набор столбцов, который уникально идентифицирует каждую строку в таблице. Каждая таблица должна иметь первичный ключ, а первичный ключ должен быть уникальным. В примере на рисунке 4.1 столбец ID таблицы EMPLOYEES является первичным ключом (то есть каждый сотрудник имеет уникальный идентификационный номер), соответственно в таблице DEPARTMENTS столбец ID также первичный ключ, который однозначно идентифицирует каждую отдельную строку.
- внешний ключ – столбец или сочетание столбцов в одной таблице, который ссылается на первичный ключ в той же или другой таблице. В примере Dept_ID внешний ключ идентифицирует отдел, в который нанят работник [16].

Таблица 4.1. Таблица EMPLOYEES

ID	Name	Address	Birth_date	Dept_ID
110	Jones	12 Oxford Street	03-03-66	10
301	Smith	53 Hayes Drive	08-12-53	20
134	Gonzales	5609 Maple Court	10-02-87	40

4.2. Преобразование ER-модели в реляционную модель

При преобразовании ER-модели в реляционную модель меняется терминология. Для преобразования необходимо выполнить следующие действия [16]:

- Каждой сущности, определенной в ER-модели, необходимо поставить в соответствие реляционную модель данных (табличное представление). В зависимости от среды проектирования и реализации имена сущностей и отношений в логической и реляционной модели могут различаться, так как при физической реализации могут накладываться дополнительные синтаксические ограничения, обусловленные требованиями конкретной СУБД. Обычно в имени не должно содержаться пробелов и специальных символов. Также имя имеет ограничение по длине, и не рекомендуется или недопустимо использовать зарезервированные слова.
- Каждый атрибут сущности становится атрибутом соответствующего отношения или столбцом таблицы. Переименование атрибутов происходит по той же схеме, что и переименование сущностей. Атрибутам ставится в соответствие тип данных, поддерживаемый конкретной СУБД, а также допустимость или недопустимость NULL значений, т.е. устанавливается обязательность атрибута.
- Первичный уникальный идентификатор трансформируется в первичный ключ или PRIMARY KEY. Атрибуты, входящие в состав первичного ключа, автоматически получают свойство обязательности (NOT NULL).
- Вторичные уникальные идентификаторы сущности становятся уникальными ограничениями целостности, автоматически получают свойство обязательности (NOT NULL).
- Отношения между сущностями преобразуются во внешний ключ. Для этого каждой подчиненной сущности, участвующей в отношении, добавляется набор атрибутов основной сущности, являющихся первичным ключом. Добавляются дополнительные столбцы в подчиненную таблицу для обеспечения организации связи между объектами. Этот набор атрибутов становится внешним ключом (FOREIGN KEY) подчиненной сущности.
- Для атрибутов, определяющих внешний ключ, задается свойство обязательности, т.е. недопустимости неопределенных значений (признак NOT NULL), или опциональности (необязательности), в этом случае для атрибута допустимо значение NULL.

- Ограничения целостности БД обязана поддерживать. Некоторые ограничения определяются в проверочных ограничениях (CHECK). Другие сложные правила требуют дополнительного программирования [16].

4.3. Соглашения по наименованию объектов реляционной модели

Для преобразования логической модели данных в реляционную модель необходимо определить правила наименования так, чтобы все участники проекта могли однозначно интерпретировать модель.

В этом разделе объясняются стандарты наименования, используемые в Oracle, однако описанные стандарты не являются единственными для использования.

Пакет моделирования Oracle SQL Developer Data Modeler предоставляет возможности наименования объектов с использованием следующих механизмов:

- Глоссарий позволяет предопределить слова и аббревиатуры, определить, какой тип они представляют (первичный – Primary, класс – Class, модифицируемый – Modified, спецификатор – Qualifier);
- Шаблон наименования позволяет определить шаблон для наименования объектов. Например, для первичного ключа можно в шаблоне указать суффикс _PK. Если данный шаблон применить к таблице {table}_PK, то к наименованию первичного ключа всегда автоматически будет добавляться суффикс;
- Ограничение имен позволяет установить ограничение наименования объектов при проектировании. Например, использовать в наименовании атрибутов сущностей только строчные буквы. Данное ограничение устанавливается в свойствах модели;
- Изменение имен позволяет изменять правила наименования объектов, созданных ранее. Автоматически переименовываются объекты, для которых установлены правила;
- Формат имени позволяет устанавливать правила наименования, использующие сокращения или шаблоны;
- Проверка правил наименования позволяет проверить модель и убедиться, что объекты характеризуются полнотой с точки зрения моделирования.

Основные правила наименования объектов реляционной модели:

- Имена таблиц задаются во множественном числе. Идея состоит в том, что содержимое таблицы – множество строк, характеризующих объект (служащий, клиент), поэтому использование множества является более целесообразным;

- Имена столбцов сохраняют имена атрибутов. Однако специальные символы, которые можно встретить в именах атрибутов, заменяются символом подчеркивания. В именах атрибутов запрещается использовать пробелы. Стандарт SQL считает символ пробела разделителем, если он не используется в кавычках, т.е. представляет собой строку. Пример преобразования имени атрибута «Department ID» – это столбец с именем DEPARTMENT_ID. Так как длина имени обычно ограничена, то в имени столбца используется значительно больше аббревиатур, чем в имени атрибута;

- Аббревиатуры. Использование уникальных аббревиатур для каждой таблицы является полезным механизмом для наименования внешних ключей и столбцов, содержащих внешний ключ.

Рекомендуется задавать аббревиатуры для имен сущностей, имеющие смысл (EMPLOYEE – EMP), или формировать аббревиатуру, удалив гласные из наименования (FLIGHT – FLT). Эти правила не гарантируют уникальность, однако могут минимизировать дублирование имен.

Для столбцов, которые не могут содержать значения NULL, рекомендуется добавлять «NN» в конец имени.

Рекомендуемое правило наименования для ограничения целостности внешнего ключа:

<краткое имя таблицы> <краткое имя таблицы первичного ключа>_<fk>.

Например, внешнему ключу, устанавливающему связь между таблицами EMPLOYEES и DEPARTMENTS, можно задать имя emp_dept_fk.

Столбцы, содержащие внешний ключ, рекомендуется именовать, используя в качестве префикса имени столбца аббревиатуру таблицы, на которую внешний ключ ссылается (dept_no). Ограничение длины имени атрибута составляет 22 символа.

Если имеется два или более внешних ключей между двумя таблицами, внешние ключи и столбцы внешнего ключа будут иметь одинаковые имена. Для решения этой проблемы необходимо добавить имя отношения этим ключам, чтобы однозначно идентифицировать их. Если вы не измените самостоятельно имена ключей, то Oracle SQL Developer Data Modeler самостоятельно добавит номер в конце второго внешнего ключа для обеспечения уникальности имен.

Для определения имени ограничения целостности CHECK рекомендуется использовать следующее правило:

<краткое имя таблицы>_ck<значение числовой последовательности>.

Например, для таблицы EMPLOYEES ограничение целостности CHECK может быть задано с именем emp_ck1.

Каждая СУБД может иметь свои собственные ограничения по наименованию объектов. При проектировании БД необходимо ознакомиться с допустимыми соглашениями использования имен.

Для СУБД Oracle существуют следующие ограничения наименования объектов БД:

- для наименования объектов можно использовать любые алфавитно-цифровые символы;
- имена объектов должны начинаться с буквы;
- максимальная длина имени – 30 символов;
- в имени не могут использоваться специальные символы. Исключением являются символы «\$», «#» и «_»;
- имена таблиц должны быть уникальными в пределах схемы (базы данных). Они не должны повторять имена других объектов, например временных таблиц, синонимов или представлений;
- имена столбцов в рамках в одной таблицы должны быть уникальными. При наличии одноименных столбцов автоматически в конец имени проставляется порядковый номер его повторения. В этом случае имя столбца соответствует шаблону «COLUMN_NAME#», где # –номер повторения;
- при задании имен запрещается использовать зарезервированные слова языка программирования, например: VARCHAR, Number, Sequence, Values, Level, или Type.

В случае, если символ пробела присутствует в наименовании, то при генерации DDL в Oracle SQL Developer Data Modeler наименование размещается в кавычках.

4.4. Использование инструментальной среды при проектировании реляционной модели данных

Прежде чем преобразовать логическую модель данных в реляционную модель, необходимо убедиться, что логическая модель является полностью завершенной. Для верификации модели используются правила проектирования, заданные в Oracle SQL Developer Data Modeler, проверка которых показывает, что все объекты в модели определяются полностью. Правила проектирования объекта не гарантируют, что модель спроектирована корректно с точки зрения бизнес-логики, однако с их помощью можно выявить некоторые ошибки, например, несоответствия наименований сущностей.

Для применения правил проектирования к конкретной логической модели необходимо в пункте меню Tools (инструменты) выбрать Design Rules (правила проектирования). Для запуска проверки на конкретное правило проектирования необходимо выбрать его из списка и нажать кнопку «Apply Selected» (применить выбранное). Для запуска проверки на все правила используется кнопка «Apply All» (применить все).

Для перехода от логической модели к реляционной необходимо преобразовать все простые сущности в таблицы. Наименования таблиц необходимо задавать таким образом, чтобы можно было легко определить, от какой сущности таблица образована. Часто при проектировании реляционной модели в качестве имен таблиц выбирается имя сущности во множественном числе, так как таблица будет содержать набор строк.

Для наименования сущностей правой кнопкой мыши нажмите на сущность, имя которой необходимо изменить, и выберите Properties. В пункте General установите значение для поля Short Name, определяющее аббревиатуру для сущности. Эта аббревиатура будет использована в качестве аббревиатуры для реляционной таблицы. В поле «Preferred Abbreviation» определяется предпочтительная аббревиатура, которая будет использована в качестве наименования таблицы.

Успешно завершив проверку объектов логической модели на корректность использования правил проектирования и определив имена параметров, можно преобразовать спроектированную логическую модель в реляционную. Для этого необходимо выбрать пиктограмму Engineer to Relational Model (преобразование к реляционной модели) на панели инструментов, появится окно преобразования моделей (рисунок 4.2).

Просмотр объектов, которые будут спроектированы для реляционной модели, можно осуществить, развернув дерево объектных типов логической и реляционной модели. Обратите внимание, что таблицы будут создаваться в реляционной модели.

Применение стандартов наименования, которые были определены ранее, осуществляется через вкладку General Options. Вкладка располагается в нижней части окна преобразования моделей, затем необходимо выбрать опцию «Apply name translation» (применить преобразование имен). Если при проектировании логической модели были установлены предпочтительные аббревиатуры для сущностей, атрибутов, связей, то при преобразовании эти аббревиатуры будут применены, так как в свойствах General Options по умолчанию установлена опция «Use preferred abbreviations» (использование предпочтительных аббревиатур).

Нажатие на кнопку «Engineer» приводит к преобразованию логической модели в реляционную. Если результаты преобразования не корректны, необходимо нажать на кнопку «Cancel» (отмена), внести изменения в

логическую структуру и повторно выполнить шаги, связанные с преобразованием моделей.

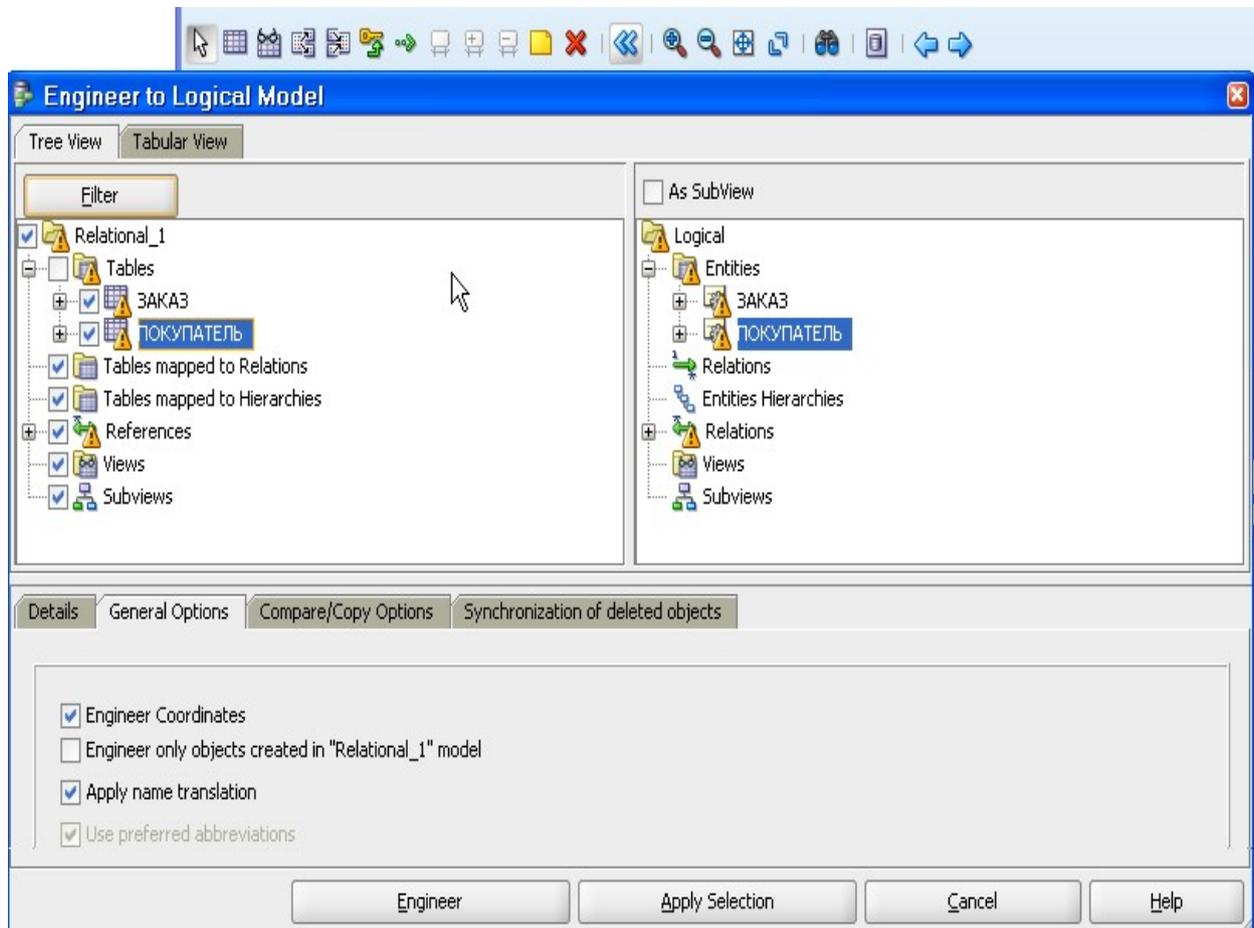


Рисунок 4.2 – Окно преобразования моделей

Каждый атрибут, определённый в сущности, преобразуется в столбец таблицы. Обязательные атрибуты преобразуются в столбцы с ограничением целостности NOT NULL (NN). В примере на рисунке 4.3 атрибуты сущности DEPARTMENT преобразуются в столбцы таблицы DEPARTMENTS, а атрибуты сущности EMPLOYEE отображаются в столбцы таблицы EMPLOYEES.

Для каждого атрибута необходимо соблюдать следующие правила наименования:

- Избегать использования зарезервированных слов SQL для имен столбцов, таких как NUMBER;
- Использовать согласующиеся аббревиатуры, чтобы не вводить других программистов и пользователей в заблуждение. Например, для NUMBER использовать NO или NUM;
- Использовать короткие имена для столбцов, чтобы сократить время, используемое для разбора SQL предложения.

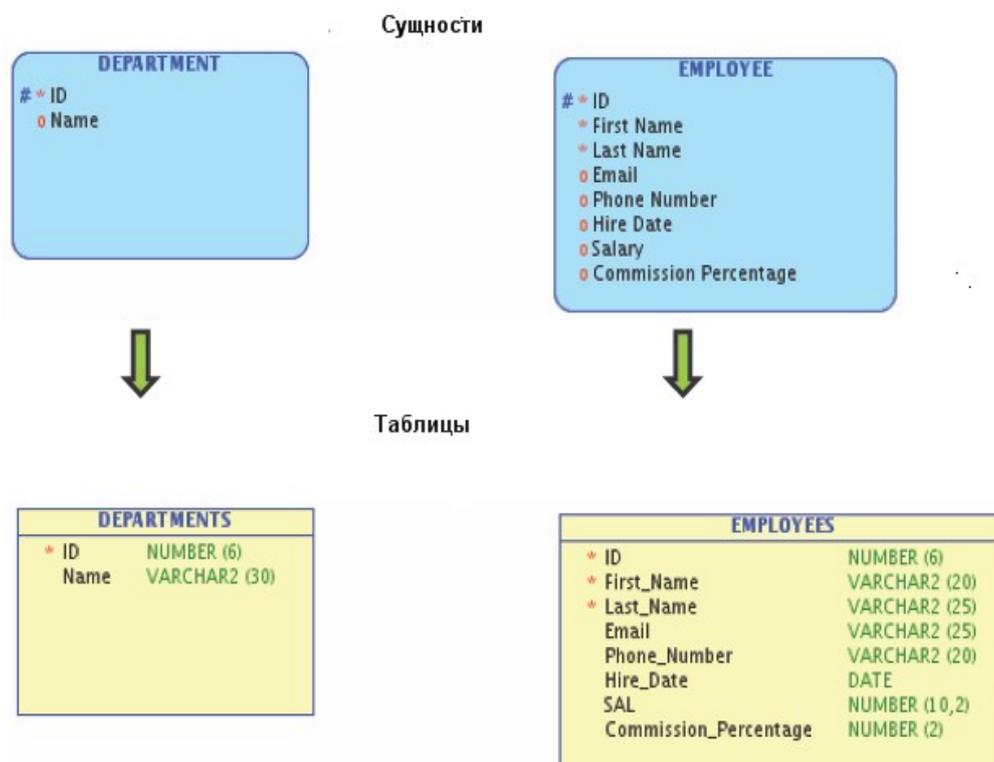


Рисунок 4.3 – Преобразование атрибутов сущности в столбцы таблицы

В реляционной модели можно указать аббревиатуру, которая будет использована при выборке, добавлении или изменении информации в таблице.

Глоссарий – это механизм, который используется во время процесса преобразования моделей для перевода наименований, используя соответствия наименований заданным аббревиатурам. Данные соответствия указываются в глоссарии. Рекомендуем применять только один глоссарий к модели, так как использование нескольких глоссариев может привести к неоднозначной ситуации. Например, аббревиатура «AP» может быть «Accounts-Payable» (счета, подлежащие оплате), а также может быть сокращением от «Actual Placement» (фактическое размещение).

Создание и использование глоссария состоит из двух этапов: создания глоссария и просмотра глоссария. На первом этапе создается глоссарий или модифицируется уже существующий. При определении глоссария указываются все элементы модели и аббревиатуры, которые будут использоваться при преобразовании логической модели данных к реляционной. Кроме того, можно задать различные параметры проектирования, например, символы-разделители, которые будут использоваться при проектировании. Чтобы получить доступ к глоссарию, необходимо выбрать Glossary Editor (редактор глоссария) в пункте меню Tools (инструменты).

Завершив создание глоссария, его необходимо добавить в список правил, применяющихся при проектировании. В пункте меню Tools (инструменты) необходимо выбрать Preferences(настройки) → Naming Standarts (стандарты наименования), в появившемся окне добавить необходимый глоссарий. Выбранный глоссарий будет добавлен, и во всех именах атрибутов появится разделяющий символ нижнего подчеркивания «_», а также дополнительные сокращения, определенные в словаре.

Использование дополнительных аббревиатур, отсутствующих в глоссарии, возможно при выполнении следующих действий:

- создать файл в формате CSV (Comma Separated Values), содержащий сокращения для набора имен. В каждой строке файле должна содержаться информация об имени и аббревиатуре для этого имени. Необходимо принять во внимание, что имена чувствительны к регистру;
- в пункте меню Tools (инструменты) выбрать Name Abbreviations (наименования аббревиатур);
- в диалоговом окне Name Abbreviations выбрать файл с расширением *.csv, который был создан на первом шаге. Затем необходимо выбрать объекты, требующие применения сокращений, или объекты, для которых необходимо изменить аббревиатуру, нажать кнопку «ОК»;
- в появившемся окне журнала регистрации можно увидеть результат переименования. Просмотрев изменения, необходимо закрыть окно, нажав на кнопку «Close»;
- результаты переименования столбцов отображаются на реляционной диаграмме.

Атрибуты, являющиеся частью уникального идентификатора сущности, преобразуются в столбцы с ограничением целостности первичного ключа в реляционной модели. На рисунке 4.4 уникальный идентификатор ID сущности DEPARTMENT преобразуется в первичный ключ DEPT_PK таблицы DEPARTMENTS.

Все столбцы в качестве первичного ключа должны быть обязательными и не должны допускать ввода значений NULL.

При преобразовании уникальных идентификаторов в ключи в окне преобразования моделей разверните узел Entities (сущности) для того, чтобы увидеть список ключей, которые будут созданы для данной сущности на основе этих уникальных идентификаторов.

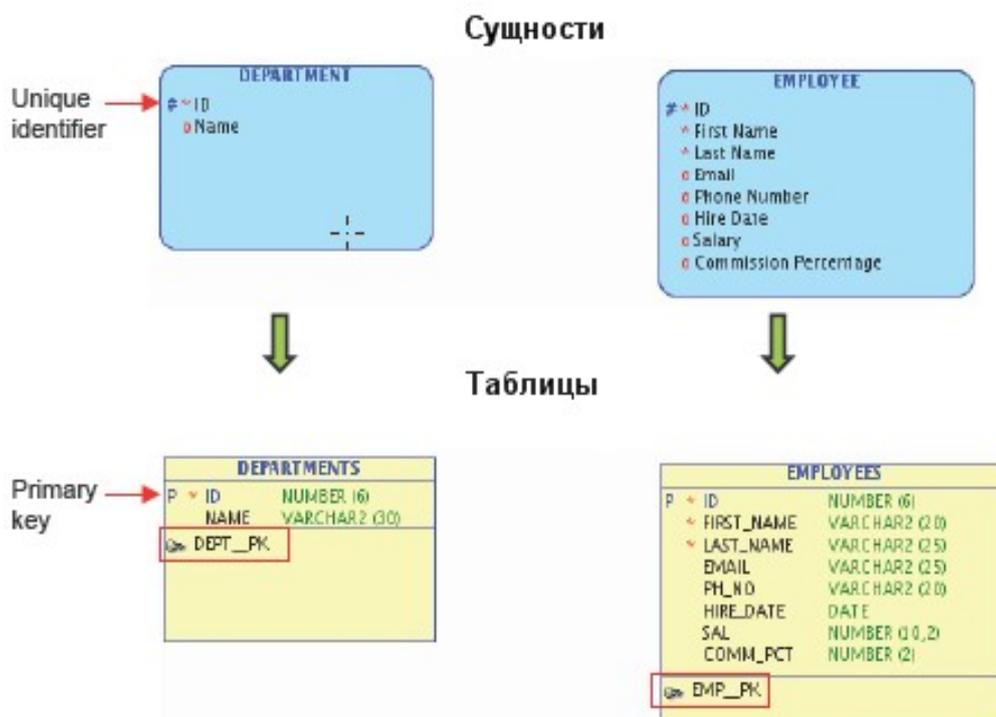


Рисунок 4.4 – Преобразование уникальных идентификаторов в первичный ключ

После преобразования логической модели к реляционной модели для отношения «один ко многим» (1:M) столбец первичного ключа исходной таблицы дублируется в зависимую таблицу и становится столбцом, определяющим внешний ключ. На рисунке 4.5 первичный ключ ID таблицы DEPARTMENTS становится столбцом таблицы EMPLOYEES, определяющим ограничение целостности внешнего ключа.

Необходимо учесть, что для нового столбца наименование формируется из аббревиатуры таблицы и наименования столбца первичного ключа. Способ формирования наименования задается в шаблоне.

Преобразование отношения с использованием внешнего ключа подчиняется следующим правилам:

- если первичный ключ таблицы включает в себя внешний ключ, столбец внешнего ключа может быть определен в результате идентификации отношения и, возможно, уже был добавлен;
- для обязательного отношения «один к одному» (1:1) внешний ключ будет сформирован для таблицы, имеющей обязательное отношение. При этом будет установлено ограничение NOT NULL для обеспечения условия обязательного существования значения;
- если отношение «один к одному» (1:1) не является обязательным в обоих направлениях, внешний ключ может быть создан в любой таблице;

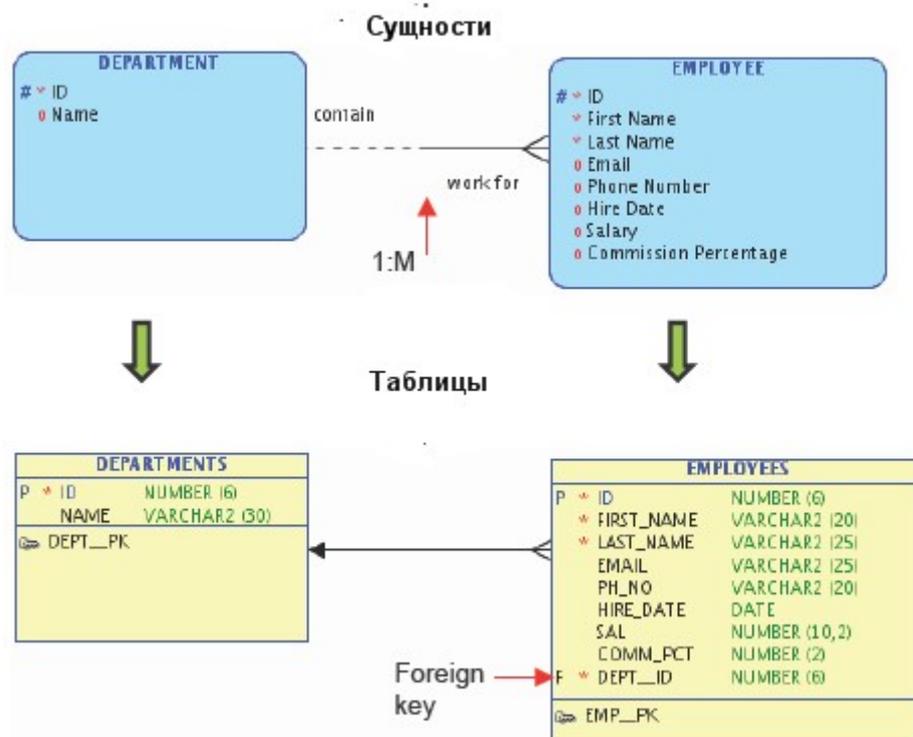


Рисунок 4.5 – Реализация отношения 1:M

- внешний ключ добавляется к таблице, имеющей рекурсивное отношение «один ко многим» (1:M), в которой определен первичный ключ. Столбец внешнего ключа ссылается на значения из столбца первичного ключа;

- рекурсивное отношение «один к одному» (1:1) разрешается путем добавления уникального внешнего ключа к таблице, в которой определен первичный ключ. Таким образом, столбец внешнего ключа ссылается на столбец первичного ключа.

Для просмотра возможных внешних ключей в окне преобразования моделей необходимо развернуть узел Relations, а затем нажать на кнопку «Engineer».

Шаблон, определяющий комбинации predefined переменных, можно создавать для определения наименований ключей, индексов и ограничений целостности. Доступ к окну Naming Templates (шаблоны наименования) осуществляется через пункт меню Tools (инструменты) → Naming Standard → Templates.

Для каждого элемента реляционной модели можно установить или predefined переменную установки шаблона. Predefined переменные включают в себя следующие зарезервированные слова:

- {table} – таблица;
- {table abbr} – аббревиатура таблицы;

- {child} – наследник;
- {child abbr} – аббревиатура наследника;
- {parent} – родитель;
- {parent abbr} – аббревиатура родителя;
- {column} – столбец;
- {column abbr} – аббревиатура столбца;
- {ref column} – ссылочный столбец;
- {ref column abbr} – аббревиатура ссылочного столбца;
- {seq nr} – номер последовательности;
- {model} – модель;
- alphanumeric constants – алфавитно-цифровые константы.

Используя комбинацию predefined переменных и функции SUBSTR, можно создавать шаблон наименования для каждого элемента.

Функция SUBSTR имеет следующий синтаксис:

SUBSTR (index, length, direction, expression),

index – начальный индекс позиции, с которой формируется новая строка результата;

length – количество символов, выбираемых из строки;

direction – направление движения курсора;

expression – выражение для которого задается шаблон.

Результаты применения шаблона для таблицы с наименованием SYSADMIN и модели с наименованием USERORACLE приведены в таблице 4.2.

Таблица 4.2. Применение шаблонов

Шаблон	Результат
{table}_FK	SYSADMIN_FK
SUBSTR(5,6,FRONT,{model})	ORACLE
SUBSTR(1,3,FRONT,{table})	SYS
SUBSTR(1,3,FRONT,TABLE)	TAB (where "TABLE" is a constant not a variable)
HELLO_SUBSTR(1,3,FRONT,{table})_ SUBSTR(5,6,FRONT,{model})_{seq nr}	HELLO_SYS_ORACLE_1

Шаблоны можно применять для объектов сформированной реляционной модели. Для этого необходимо выбрать:

- таблицу;
- опцию «Naming Rules»;
- типы объектов, для которых будет применяться шаблон
- подтвердить применение шаблона.

Шаблон также можно применять ко всем компонентам модели. Для этого выберите:

- модель в дереве окна навигации (правой кнопкой мыши);
- опцию «Apply Naming Standards to Keys and Constraints» (применить стандарты наименования для ключей и ограничений целостности);
- типы объектов, для которых необходимо применить шаблоны, нажмите кнопку «ОК».
- обратите внимание, что имена первичных ключей изменились согласно шаблону, который был задан в виде {таблица} _РК.

Удобным механизмом для представления различных аспектов жизненного цикла, владения или использования объектов являются префиксы. Префиксы можно создавать для временного изменения наименования объекта или для фиксации изменений, в этом случае дальнейшая модификация не допустима.

Фиксированные изменения – это изменения, которые происходят с объектами во время дизайна модели. Фиксированные префиксы можно применять к таблицам, представлениям, столбцам и индексам.

Временное изменение наименования объекта происходит при создании сценария DDL. В этом случае необходимо определить `old_prefix`, а затем применить подстановку имен во время создания DDL операторов. Этот подход не влияет на имена объектов в моделях.

Чтобы добавить префикс к модели, выполняются следующие действия:

- нажмите правой кнопкой мыши на реляционной модели и выберите «Change Object Names Prefix» (изменить префикс в наименовании объектов);
- установите новый префикс, нажмите кнопку «Add new prefix» (добавить новый префикс), выберите объекты, к которым хотите применить данный префикс и нажмите кнопку «Apply»;
- в появившемся диалоговом окне, показывающем количество изменений, нажмите кнопку «ОК»;
- обратите внимание, что имена таблиц теперь имеют заданный префикс.

Для взаимоисключающих отношений, определенных в логической модели, при преобразовании к реляционной модели столбец, определяющий внешний

ключ, создается в каждой подчиненной таблице отношения и включается в дугу. На рисунке 4.6 представлена реализация взаимоисключающих отношений между таблицей EMPLOYEE и COUNTRY, COUNTY или OTHER_STATE.

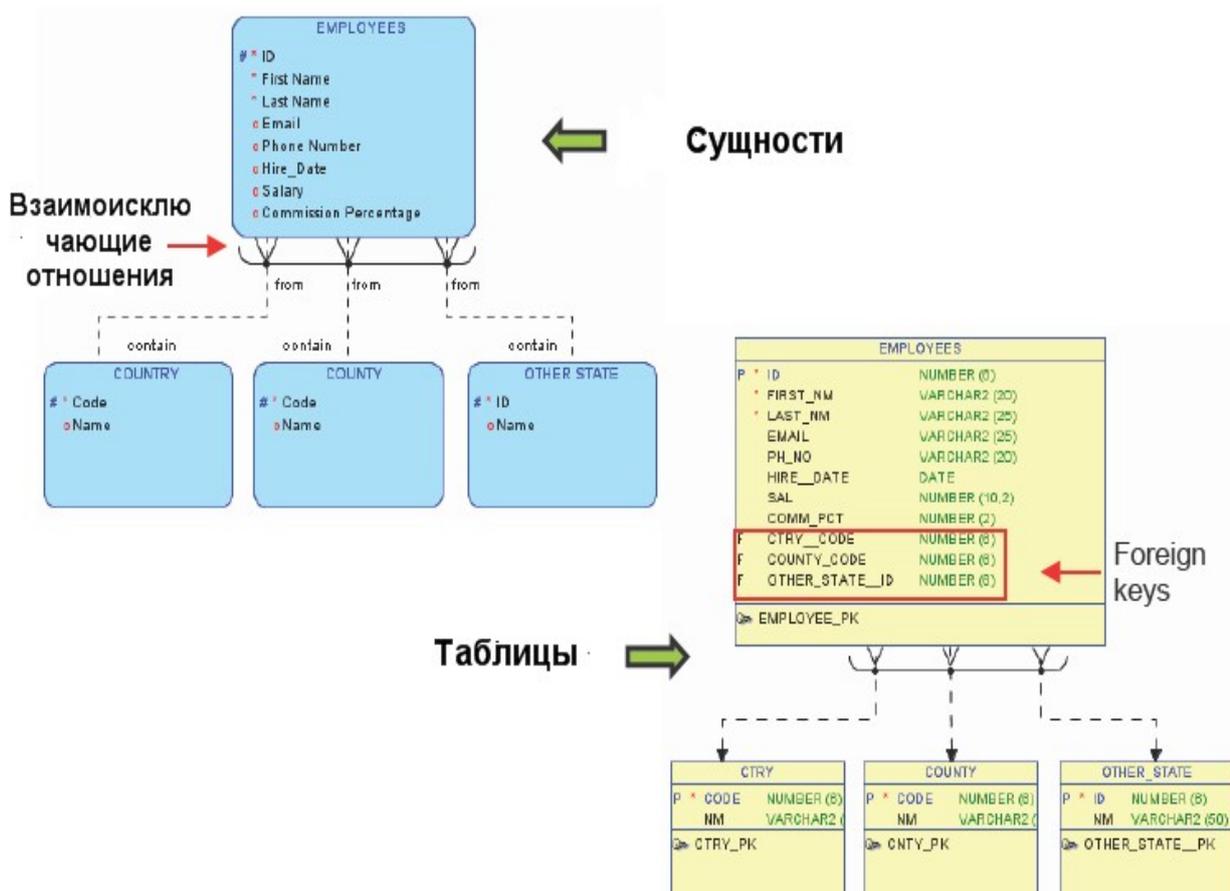


Рисунок 4.6 – Реализация взаимоисключающих отношений

Во время преобразования из логической модели в реляционную взаимоисключающего отношения столбец, определяющий внешний ключ, создается для каждого отношения, которое включено в дугу.

Для успешного преобразования логической модели в реляционную необходимо установить основные параметры преобразования. Эти параметры определены во вкладке General Options (основные параметры) окна преобразования моделей. Можно выбрать следующие параметры:

- Engineer Coordinates – преобразование координат. Данный параметр позволяет задать координаты объектов так, что при преобразовании из логической модели в реляционную объекты будут отображены в той же последовательности.
- Engineer only objects created in “Logical” model – преобразование объектов, созданных только в логической модели. Используя данную опцию, можно запускать процесс преобразования моделей многократно.

Данный параметр также определяет, что только объекты, созданные в логической модели, создаются в реляционной модели.

- **Apply name translation** – применение преобразования имен. Установка в качестве наименований для объектов реляционной модели аббревиатур, указанных в глоссарии, или аббревиатур, установленных в окне свойств сущности, определенной в логической модели.

- **Use preferred abbreviations** - использование заданных сокращений. Данный атрибут позволяет применять аббревиатуры, заданные в глоссарии и свойствах сущности.

Вкладка **Compare/Copy Option** позволяет сравнить свойства логических и реляционных объектов во время процесса преобразования моделей.

Объекты, которые необходимо преобразовать в реляционную модель, размещены в левой части окна. Выбрав требуемый тип объекта, в правой части окна можно увидеть список свойств, доступных для передачи в реляционную модель.

Управляя списком свойств, доступным для объекта преобразования, можно показывать или исключать различные уровни детализации для таблиц, сущностей, представлений и отношений. Однако нет возможности управлять уровнями детализации для столбцов.

Для просмотра свойств, установленных во вкладке **Compare/Copy Option**, необходимо развернуть дерево в окне преобразования со стороны, представляющей логическую модель, и со стороны, представляющей реляционную модель, выбрать интересующий объект. В нижней части окна появится вкладка **Details**, откройте ее для просмотра сведений о сравнении объектов двух моделей.

По умолчанию удаленные объекты не выбираются в процессе преобразования моделей. Однако в дереве модели можно найти удаленные объекты, свойства которых можно просмотреть. Для того чтобы развернуть все дерево и просмотреть все удаленные объекты для логической модели, необходимо выбрать **Synchronizing Deleted Objects** (Синхронизация удаленных объектов). Существует возможность из списка удаленных объектов сформировать объекты реляционной модели во время процесса преобразования.

4.5. Физическая модель данных

Процесс разработки структуры базы данных не заканчивается на разработке реляционной модели. Для завершения разработки необходимо создать физическую модель и сгенерировать скрипт, который будет содержать операторы SQL, необходимые для создания базы данных. Физическая модель данных – это модель, полностью зависящая от среды реализации. Она

определяет способы размещения данных и способы доступа к ним. Создание физической модели средствами пакета Oracle SQL Developer Data Modeler позволяет описать базу данных в терминах реляционной модели. Пакет позволяет создавать табличные пространства, таблицы, представления, триггеры и т.д. Каждая реляционная модель может иметь одну или несколько физических моделей, в зависимости от задач разработчика. Каждая физическая модель основана на объектах RDBMS.

Для создания SQL-скрипт в Oracle SQL Developer Data Modeler необходимо выполнить следующие действия:

- нажать на пиктограмму «Generate DDL» панели инструментов, откроется окно редактора DDL файлов;
- из раскрывающегося списка выбрать базу данных, физическую модель которой необходимо создать (рисунок 4.7);
- нажать на кнопку «Generate». По умолчанию выбираются все объекты реляционной модели. Если необходимо сгенерировать SQL-скрипт только для некоторых объектов, отметьте их.
- нажмите кнопку «OK» для завершения процесса создания SQL-скрипт.

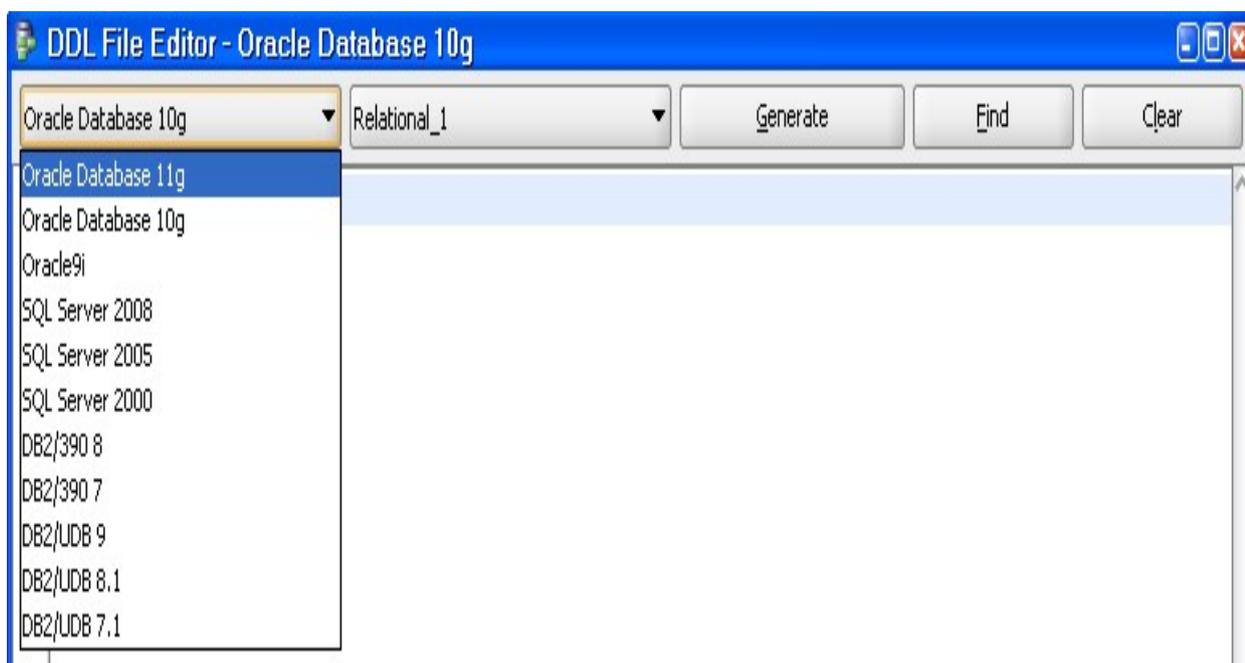


Рисунок 4.7 – Редактор DDL файлов

В случае, если выбраны все объекты базы данных, то после нажатия кнопки «OK» произойдет ошибка проектирования, так как табличное пространство, связанное с таблицей, не определено никакими файлами данных.

Чтобы исправить ошибку, закройте это окно и определите файлы данных для табличного пространства. Сгенерируйте DDL-скрипт повторно.

Вы также можете создать оператор удаления объектов DROP. Этот оператор необходимо создавать, если в БД уже существуют объекты с заданным именем и их надо удалить перед созданием новых объектов с тем же именем. Будьте осторожны с этим оператором, если работаете с производственными системами, так как выполнение DROP приводит к удалению объекта, например, таблицы, а вновь созданный объект не будет содержать информации, например, будет создана новая, но пустая таблица.

Для формирования корректного DDL-скрипта можно указать значения по умолчанию. Для этого выберите пункт меню Tools (инструменты) → Preferences (общие параметры) → DDL:

- добавление операторов завершения строки для баз данных IBM DB2 и UDB реализовано опцией «STATEMENT TERMINATION CHARACTER FOR DB2 AND UDB»;
- создание триггеров для замещения типов при формировании физических моделей для Oracle и IBM UDB реализовано опцией «CREATE TYPE SUBSTITUTION TRIGGERS FOR ORACLE AND UDB»;
- создание триггера для реализации дуги внешнего ключа в физической модели реализовано опцией «CREATE TRIGGERS FOR FK ARC CONSTRAINT» для дуги внешнего ключа. Определяет возможность создания триггера в сгенерированном DDL-скрипте для реализации внешних ключей, объединенных дугой;
- «CREATE TRIGGERS FOR NON TRANSFERABLE FK» – создание триггеров для внешних ключей, не подлежащих передаче. Определяет, будет ли создаваться триггер для непередаваемого внешнего ключа. Если внешний ключ может передаваться, то устанавливается опция Transferable (Updateable) – обновляемый;
- Для проверки приведения типов и определения типа данных (домен, логический тип, специальный тип) определена опция «USE DATA TYPE KIND»
- «USE “SCHEMA” PROPERTY IN COMPARE FUNCTIONALITY» – использование свойств схемы для сравнения функциональности;
- «CASE SENSITIVE NAMES IN COMPARE FUNCTIONALITY» позволяет учитывать регистр имен в операциях сравнения;
- «GENERATE SHORT FORM OF NOT NULL CONSTRAINT» позволяет создавать краткую форму ограничения целостности NOT NULL;
- «USE QUOTED IDENTIFIERS» позволяет использовать идентификаторы в кавычках;
- «CREATE DOMAINS DURING IMPORT» позволяет создавать домены при импорте данных;

- «AUTOMATIC INDEX GENERATION» позволяет автоматически генерировать индекс. Индексы могут создаваться автоматически, если атрибута является первичным ключом, уникальным ключом или внешним ключом.

Резюме

Инструментальная среда Oracle SQL Developer Data Modeler обладает широкой функциональностью и позволяет проектировать логическую, реляционную и физическую модель данных, используя синтаксис языка конкретной СУБД. Пакет обеспечивает проверку поддержки целостности данных, использует механизмы верификации и правила наименования объектов модели, облегчает процесс проектирования и проверки модели, освобождая проектировщика от рутинных действий, и позволяет сосредоточиться на выявлении логических ошибок и узких мест в проектируемой базе данных.

Контрольные вопросы

1. В каком виде ER-диаграмма представляет данные: двумерная таблица, многомерная таблица, иерархическая структура, объектно-ориентированная структура?
2. Каково назначение ключевых полей в таблицах реляционной базы данных?
3. Как строится реляционная база данных?
4. О чем говорит правило целостности сущностей?
5. В какое ограничение целостности преобразуется отношение между сущностями?
6. В какое ограничение целостности преобразуется первичный уникальный идентификатор сущности?
7. В какое ограничение целостности преобразуется вторичный уникальный идентификатор сущности?
8. Какие соглашения по наименованию объектов реляционной модели необходимо соблюдать?
9. Для чего используется верификация модели в Oracle SQL Developer Data Modeler?
10. На основе какой модели данных Oracle SQL Developer Data Modeler позволяет автоматизировано строить реляционную модель данных?

Литература

1. Churcher C. Beginning SQL Queries: From Novice to Professional 2nd ed. Edition // Apress, 2016. – 240 p.
2. Date C. J. E. F. Codd and Relational Theory: A Detailed Review and Analysis of Codd's Major Database Writings // Lulu Publishing Services, 2019. – 304 p.
3. Kimball R. The Data Warehouse Toolkit. The Definitive Guide to Dimensional Modeling // John Wiley & Sons Inc, 2013. – 608 p.
4. Гудов А. М. Базы данных и системы управления базами данных. Программирование на языке PL/SQL. Учебное пособие / Завозкин С. Ю., Рейн Т. С. – Кемерово: Кемеровский государственный университет, 2010. – 133 с.
5. Васильев А. Е. Развитие логических моделей данных [Электронный ресурс] – Режим доступа: http://citforum.ru/database/articles/ref_vs_nav_models/ (дата обращения: 25.02.2020).
6. Вендров А. М. CASE-технологии. Современные методы и средства проектирования информационных систем [Электронный ресурс] – Режим доступа: <http://www.codenet.ru/db/other/case/> (дата обращения: 21.02.2020).
7. Костюкова А. П. Базы данных: общий курс. Часть 1: учебное пособие. – М.: Академия Естествознания, 2018. – 82 с.
8. Туманов В. Основы проектирования реляционных баз данных [Электронный ресурс] – Режим доступа: <https://www.intuit.ru/studies/courses/1095/191/lecture/4975?page=2> (дата обращения: 27.02.2020).
9. Калянов Г.Н. Консалтинг при автоматизации предприятий: подходы, методы средства. – М.: СИНТЕГ, 1997. – 316 с.
10. Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных // Архив журнала «Системы управления базами данных» № 1. – 1995. – С. 145-160.
11. Балдин К. В. Информационные системы в экономике. 7-е издание. Учебник / Уткин В. Б. – М.: Дашков и Ко, 2017. – 395 с.
12. Кузнецов С.Д. Базы данных: Вводный курс [Электронный ресурс] – Режим доступа: http://citforum.ru/database/osbd/glava_18.shtml (дата обращения: 01.03.2020).

13. Коваленко Т. Работа с базами данных [Электронный ресурс] / Сирант О. – Режим доступа: <https://www.intuit.ru/studies/courses/3439/681/lecture/14023?page=1> (дата обращения: 25.02.2020).
14. Зеленков Ю.А. Введение в базы данных [Электронный ресурс] – Режим доступа: http://www.mstu.edu.ru/study/materials/zelenkov/ch_4_2.html (дата обращения: 28.02.2020).
15. Oracle SQL Developer Data Modeler – инструментарий проектирования баз данных [Электронный ресурс] – Режим доступа: <https://softline.ru/about/news/9826> (дата обращения: 22.02.2020).
16. Войтюк Т.Е. Проектирование и реализация баз данных. – СПб: СПбГУ ИТМО, 2013. – 95 с.
17. Козленко Л. Проектирование информационных систем. Часть 1. Этапы разработки проекта: стратегия и анализ [Электронный ресурс] – Режим доступа: <https://compress.ru/article.aspx?id=11764> (дата обращения: 20.02.2020).

Войтюк Татьяна Евгеньевна
Осетрова Ирина Станиславовна

Основы проектирования реляционных баз данных средствами инструментальной среды

Учебно-методическое пособие

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО

Н.Ф. Гусарова

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверский пр., 49