2. Принципы проектирования БД

Построение моделей процессов является основным краеугольным камнем при проектировании баз данных (БД). Моделирование процессов используется для документирования бизнес-процессов, которые существуют в организации. Модель процесса показывает, как между собой взаимодействуют и обрабатываются потоки данных, а также как организовано хранение информации и взаимодействие с внешними сущностями.

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).

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 [17].

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.

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

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 и 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.

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

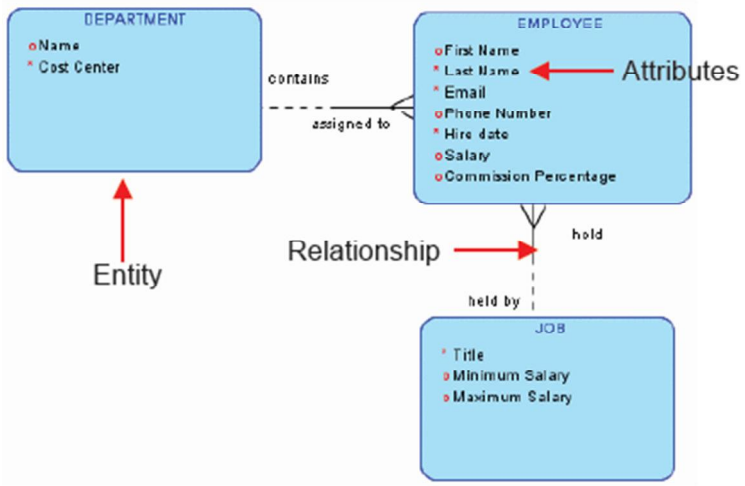
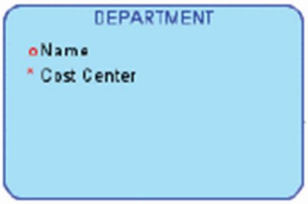
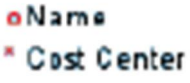


Рисунок 2.5 – ER-диаграмма в системе Баркера.

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

Таблица 2.1. Компоненты диаграммы сущность-связь

Компонент	Назначение	Графическое представление
Entity – Сущность	Объект или понятие, о котором вы хотите хранить информацию.	
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

личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. 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.

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 имени сущности, в форме существительного прописными буквами. Имя не должно содержать дефисов и символа подчеркивания.

Компонент «Сущность» можно классифицировать на следующие типы:

- первичная сущность – сущность независящая 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».

2.3. Атрибуты

Атрибут сущности - это именованная характеристика, являющаяся некоторым свойством сущности [18]. Атрибуты уточняют, определяют, классифицируют, определяют количественное или качественное состояние сущностей, они представляют собой тип описания или детализации информации о сущностях и не являются ими.

Наименование атрибута должно быть выражено существительным в единственном числе и являться уникальным. Атрибуты изображаются в пределах прямоугольника, определяющего сущность.

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 имя атрибута не должно содержать в себе имя сущности. Например, для сущности «Employee», данный атрибут не допустим employee_phone_number, 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».

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.

2.4. Отношения

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 изображаются линией, соединяющей две сущности [19].

На рисунке 2.6 представлен пример отношения между сущностями «DEPARTAMENT» и «EMPLOYEE»

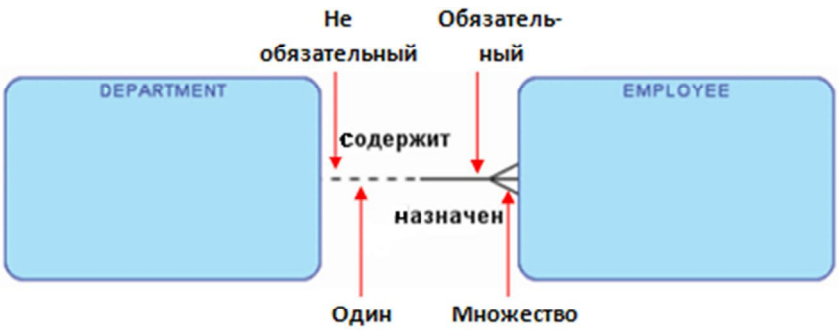


Рисунок 2.6 – Отношение между сущностями

Обязательными компонентами отношения являются:

- имя связи;
- мощность отношения.

Имя связи – задается рядом с сущностью, к которой данная связь относится, используются только строчные буквы [20]. Имя связи отражает действие, которое необходимо выполнить. В качестве примера на рисунке 2.2 представлены имена связей «содержит» и «назначен».

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

Модальность связи определяет минимальное значение экземпляров сущностей. Каждая связь может иметь одну из двух модальностей связи (рисунок 2.7).

Рисунок 2.7 – Модальность связи

Модальность "может" означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром, в этом случае допускаются значения null для связи.

Модальность "должен" означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

Связь может иметь разную модальность с разных концов.

Максимальное значение сущности может быть одно, тогда на диаграмме оно обозначается в виде линии или множество, тогда используется знак ⋹.

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 много-ко-многим (M:M);
- связь рекурсивная.

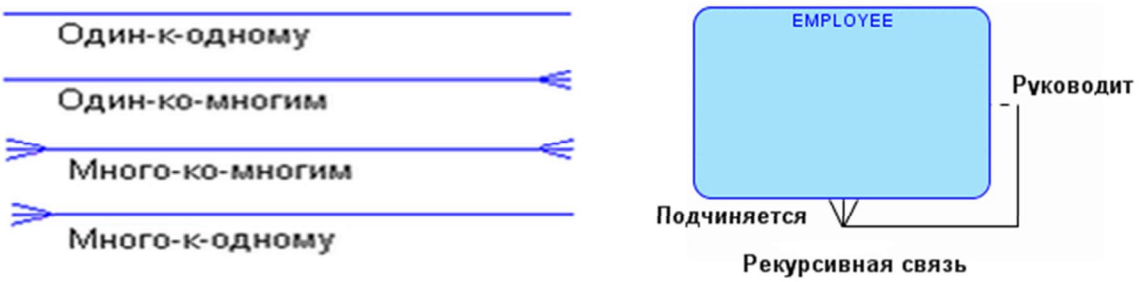


Рисунок 2.8 – Графическое представление отношений с учетом типов связей

Связь типа один-к-одному означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего

одну сущность, неправильно разделенную на две. На диаграмме данный вид связи обозначается в виде линии.

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 рис. 2.2. Левая сущность называется родительской, правая – дочерней. 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 ⋹.

Связь типа много-ко-многим означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. 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.

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 2.4 представлена рекурсивная связь, которая показывает, что любой руководитель является экземпляром сущности «EMPLOYEE» и руководит сотрудниками, которые также являются экземплярами сущности «EMPLOYEE»

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>.

Каждая связь может быть прочитана как слева направо, так и справа налево. Определение связей для рисунка 2.2 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».

Связи между сущностями можно представить не только в виде ERD, но и с помощью матрицы отношений (таблица 2.3).

Таблица 2.3. Матрица отношений

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

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

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;
- в случае если между сущностями не существует отношения, то ячейка матрицы не заполняется;
- если существует отношение 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.

2.5. Применение уникальных идентификаторов

Атрибут или группа атрибутов, которая уникально определяющих данную сущность называется ключом сущности или уникальным идентификатором сущности [21].

Уникальный идентификатор (UID) – это специальный атрибут или группа атрибутов, который однозначно идентифицирует конкретный экземпляр сущности. При проектировании ERD для указания, что данный атрибут является уникальным идентификатором, перед его именем ставится специальный символ «#».

Каждый компонент уникального идентификатора должен быть обязательным, т.е. такой компонент не может принимать значение null.

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

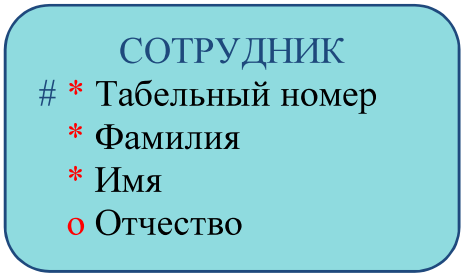


Рисунок 2.9 – Графическое представление сущности СОТРУДНИК

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

Уникальный идентификатор может быть информационным, когда значения уникального идентификатора имеет какое-то 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.

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 имею специального обозначения на рисунке 2.5.

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

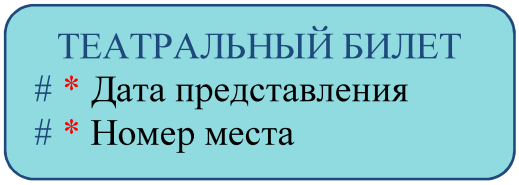


Рисунок 2.10 – Графическое представление сущности ТЕАТРАЛЬНЫЙ БИЛЕТ

В случае 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.

На рисунке 2.11 представлены две сущности «ACCOUNT» (счет) и «BANK». Уникальным идентификатором, являющимся первичным ключом для сущности «ACCOUNT» является атрибут «NUMBER», то есть номер счета. Сущность также содержит в себе атрибут «NUMBER», однако этот атрибут идентифицирует конкретный банк «BANK». В банках, имеющих различные идентификаторы, могут существовать одинаковые счета, поэтому сущность «ACCOUNT» становится зависимой от значения атрибута сущности «BANK» и на диаграмме связь отмечается вертикальной чертой, которая размещается рядом с сущностью, которая зависит от экземпляра другой сущности, т.е. от значений её атрибутов.

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

2.6. Установка Oracle SQL Developer Data Modeler

Для построения ERD диаграмм корпорация Oracle предоставляет удобное пользовательское графическое средство Oracle SQL Developer Data Modeler.

Пакет Oracle SQL Developer Data Modeler представляет собой универсальный, полностью автономный инструмент с поддержкой логического, реляционного, многомерного моделирования и моделирования типов данных. 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. С помощью пакета Oracle SQL Developer Data Modeler пользователи могут создавать, расширять и модифицировать модели данных, а также сравнивать свои модели с уже существующими [8].

Чтобы установить пакет, перейдите по ссылке на http://www.oracle.com/technetwork/developer-tools/datamodeler/downloads/datamodeler-087275.html и выберите пакет, соответствующий вашей операционной системе. В целях обучения данный пакет предоставляется бесплатно.

Распакуйте содержимое архива. Перейдите в каталог с содержимым архива и выполните запуск datamodeling.exe. Для запуска приложения потребуется также установить Java Development KIT (JDK) не ниже версии 1.6. В случае, если JDK уже установлен на вашем компьютере, приложение вам может предложить ввести расположение каталога с JDK при первом запуске.

Резюме

Логическая модель данных определяет прототип будущей базы данных. Графическим представлением логической модели данных является диаграмма сущность-связь, позволяющая представить компоненты будущей базы данных в виде сущностей, их структуру, используя атрибуты и ограничения целостности данных, благодаря использованию различных типов связей между сущностями.

Для закрепления теоретического материала раздела предусмотрено выполнение домашнего задания. Варианты домашнего задания содержатся в разделе «Принципы проектирования БД» электронного курса дисциплины.

3. Управление данными

3.1. Реляционная модель данных

Логическая модель данных описывает данные, необходимые для реализации бизнес-процессов. Эта модель используется на ранних стадиях разработки проекта, должна быть полностью независимой от любой реализации БД. Логической модель данных используются в качестве основы для реализации любого типа системы управления базой данных (СУБД) или файловой системы.

Логической моделью данных является высокоуровневое представление, которое реализовано с помощью формального языка и которое обеспечивает однозначную трактовку и концептуально "работоспособное" решение. Поэтому необходимо детально прорабатывать логическую модель данных, прежде чем непосредственно реализовать физическую структуры базы данных.

Процесс проектирования баз данных часто называют созданием реляционной модели. Реляционная модель данных – логическая 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» [27]. 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 данных, чем в иерархической или сетевой. В упомянутой статье Е.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". Таким образом, представление данных не зависит от способа их физической организации, а обеспечивается за счет использования математической теории отношений

Реляционная модель строят на стадии проектирования базы данных. Модель обеспечивает описания объектов базы данных, которые должны быть созданы в процессе генерации БД.

На рисунке 3.1 представлена реляционная модель, состоящая из двух объектов: таблица DEPARTMENTS и таблица EMPLOYEES. Каждая таблица содержит первичный ключ. В таблице EMPLOYEES существует два внешних ключа, один на поле EMPLOYEE_ID таблицы EMPLOYEES и один для DEPARTMENT_ID таблицы DEPARTMENTS.

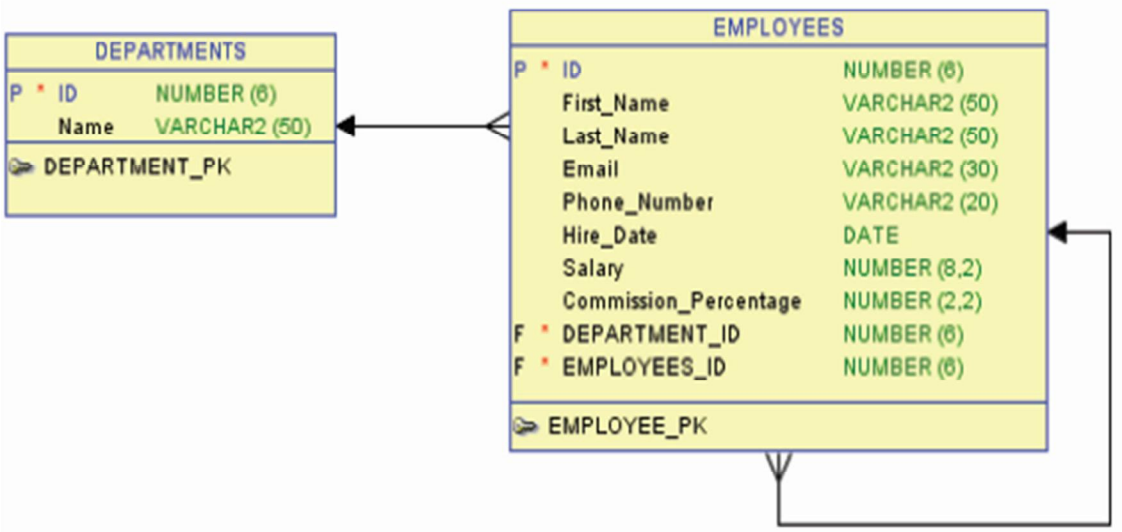


Рисунок 3.1 –Реляционная модель данных

Известному популяризатору идей Ф.Кодда – Кристоферу Дейту принадлежит следующая трактовка реляционной модели: реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

В структурной части модели фиксируется, что единственной родовой структурой данных, используемой в реляционных БД, является нормализованное N -арное отношение.

В манипуляционной части модели определяются два фундаментальных механизма манипулирования реляционными БД – реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств, второй – на классическом логическом аппарате исчисления предикатов первого порядка.

В целостной части реляционной модели данных фиксируются два базовых правила целостности, которые должны поддерживаться в любой реляционной СУБД: правило целостности сущности и требование целостности по ссылкам.

Правило целостности сущности – означает, что каждая сущность должна обладать первичным ключом, который полностью идентифицирует данную сущность, поэтому в составе любого значения первичного ключа не допускается наличие неопределенных значений.

Правило целостности по ссылкам требует, чтобы для каждого определенного внешнего ключа должен существовать соответствующий первичный ключ, либо значение внешнего ключа должно быть полностью неопределенным.

Парадигма реляционной модели данных состоит в представлении данных в виде двумерных таблиц. Таблица состоят из нескольких строк, каждая из которых состоит из набора столбцов. В таблице все строки имеют одинаковую структуру, определенную столбцами, однако в некоторых строках может не существовать значений для некоторых столбцов. Поэтому можно выделить основные компоненты реляционной модели:

- таблица – простая структура, обеспечивающая организацию и хранение информации. Таблица состоит из столбцов и строк. Каждый столбец используется для хранения конкретного типа значения. В таблице 3.1 представлена сущность EMPLOYEES предназначенная для хранения информации о сотрудниках в виде таблицы.
- строка – каждая строка описывает нового работника. В таблице 3.1 каждая строка описывает в полном объеме все свойства, определенные в реляционной модели на рисунке 3.1.
- столбец – каждый столбец содержит информацию определенного типа: ID, Name, Address, Birth_date, Dept_ID, 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 3.1 – ID является первичным ключом (то есть каждый сотрудник имеет уникальный идентификационный номер), соответственно в таблице столбец ID также первичный ключ, который однозначно идентифицирует каждую отдельную строку.
- внешний ключ – столбец или сочетание столбцов в одной таблице, которая ссылается на первичный ключ в той же или другой таблице. В примере, Dept_id внешний ключ – идентифицирует отдел в который нанят работник.

Таблица 3.1. 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

3.2. Преобразование ER-модели в реляционную модель

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

1. Каждой сущности, определенной в ER-модели необходимо поставить в соответствие реляционную модель данных (табличное представление). При этом имена сущностей и отношений между ними в реляционной модели могут отличатся, потому что на имена сущностей могут не накладываться дополнительные синтаксические ограничения, кроме уникальности имени в рамках модели. Имена отношений могут быть ограничены требованиями конкретной СУБД, чаще всего эти имена являются идентификаторами в некотором базовом языке, они ограничены

по длине и не должны содержать пробелов и некоторых специальных символов.

2. Каждый атрибут сущности становится атрибутом соответствующего отношения (столбцом таблицы). Переименование атрибутов должно происходить в соответствии с теми же правилами, что и переименование отношений в п.1. Для каждого атрибута задается конкретный допустимый в СУБД тип данных и обязательность или необязательность данного атрибута (то есть допустимость или недопустимость NULL значений для него).
3. Первичный уникальный идентификатор сущности становится первичным ключом сущности – PRIMARY KEY. Атрибуты, входящие в первичный ключ отношения, автоматически получают свойство обязательности (NOT NULL).
4. Вторичные уникальные идентификаторы сущности становятся уникальными ограничением целостности, автоматически получают свойство обязательности (NOT NULL).
5. Отношения между сущностями преобразуются во внешний ключ. Для этого каждой подчиненной сущности, участвующей в отношении, добавляется набор атрибутов основной сущности, являющихся первичным ключом, то есть добавляются дополнительные столбцы в подчиненной таблице для обеспечения организации связи между объектами. Этот набор атрибутов становится внешним ключом – FOREIGN KEY подчиненной сущности.
6. Для моделирования ограничений на физическом уровне у атрибутов, соответствующих внешнему ключу, устанавливается свойство не допустимости неопределенных значений (признак NOT NULL). При не обязательном типе связи атрибуты получают свойство допустимости неопределенных значений (признак NULL). Ограничения целостности БД обязана поддерживать. Некоторые ограничения определяются в проверочных ограничений, другие – сложные правила требуют дополнительного программирования.

3.3. Нормализация данных

Нормализация является реляционной концепцией базы данных. Ели вы разработали корректно ERD, то таблицы базы данных, созданные с помощью пакета Oracle SQL Developer Data Modeler, будут соответствовать правилам нормализации [23].

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 (BCNF) – любой атрибут, от которого полностью функционально зависит некоторый другой атрибут, может являться ключом;
- четвертая нормальная форма (4NF) – 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 (5NF или PJ/NF) – любая зависимость соединения в отношении следует из существования некоторого возможного ключа в отношении.

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 у вас есть один атрибут в одном месте, с одним именем, с одним значением, в любой момент времени.

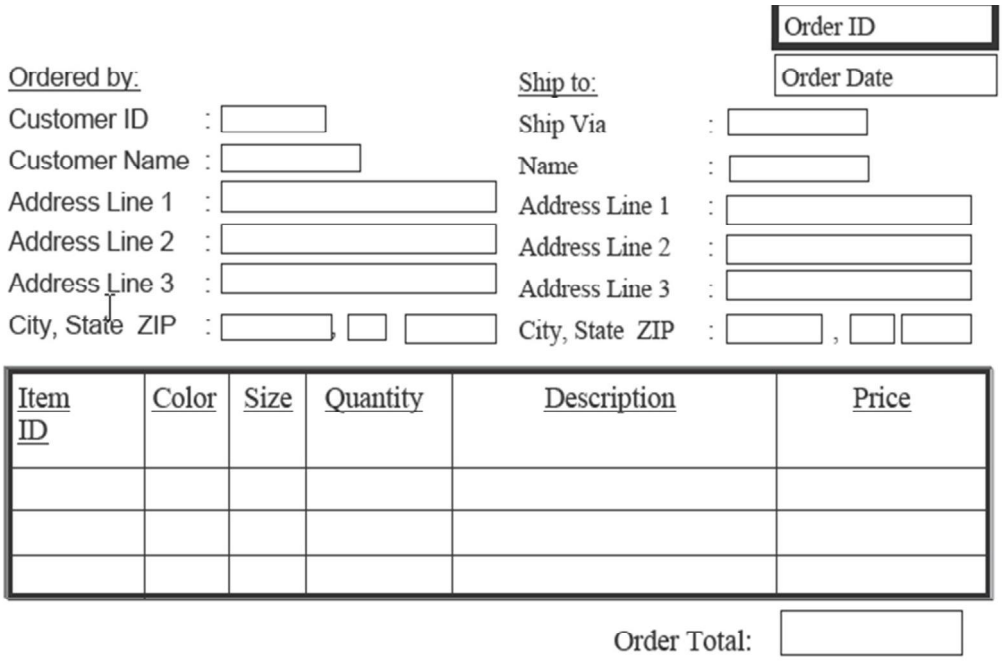


Рисунок 3.2 – Бланк заказа товара

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 3.2.

3.4. Первая нормальная форма

Первая нормальная форма гарантирует, что каждый атрибут имеет одно значение для каждого экземпляра сущности [22]. Не должно быть атрибутов, которые имеют повторяющееся значение. В примере, представленном на рисунке 3.2, 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.

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.

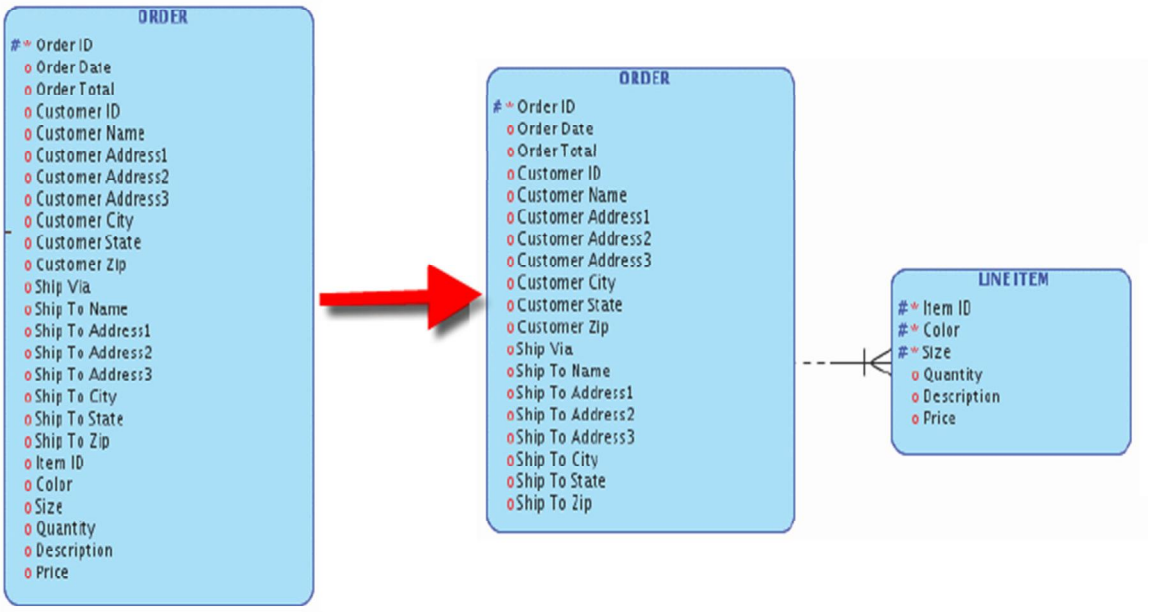


Рисунок 3.3 –Первая нормальная форма

Но такая таблица еще не является первой нормальной формой, так как содержит дублирующие записи. Для того чтобы преобразовать данную сущность к первой нормальной форме, необходимо выполнить следующие действия (рисунок 3.3):

- создать другую 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.

3.5. 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 [24]. Первичный уникальный идентификатор – является первичным ключом. Первичный ключ может включать в себя несколько атрибутов, в таком случае он называется составным первичным ключом. Не ключевые 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.

Для преобразования модели ко второй нормальной форме (2NF), 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.

Для преобразования модели из рисунка 3.3 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 не ключевые атрибуты. Определить эти атрибуты как первичный ключ для новой сущности.

На рисунке 3.4 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».

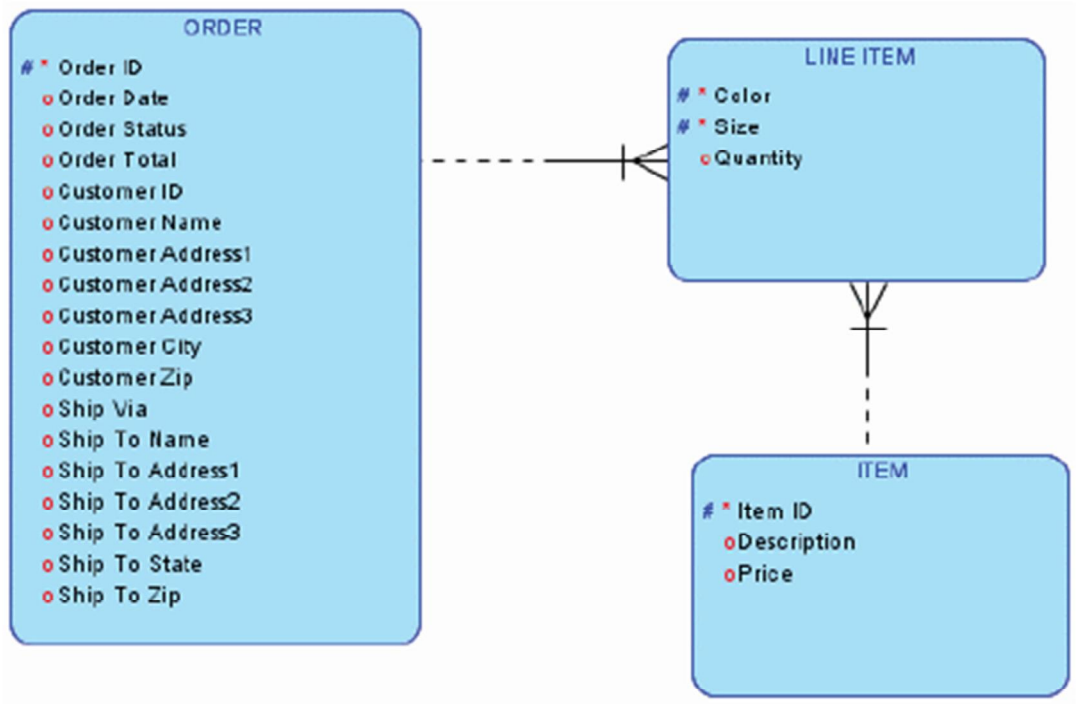


Рисунок 3.4 – Вторая нормальная форма

3.6. Третья нормальная форма

Третья нормальная форма гарантирует, что каждый атрибут зависит только от уникального идентификатора своей сущности [23].

Для преобразования модели к третьей нормальной форме (3NF), 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.

В примере на рисунке 3.5 сформирована новая сущность «CUSTOMER», содержащая атрибуты «Customer ID», «Name», «Address1», «Address2», «Address3», «City», «State», «Zip». Эти атрибуты не зависели на прямую 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, которые еще не оформлял заказ, было невозможно. В качестве первичного ключа для новой сущности выбран атрибут «Customer ID».

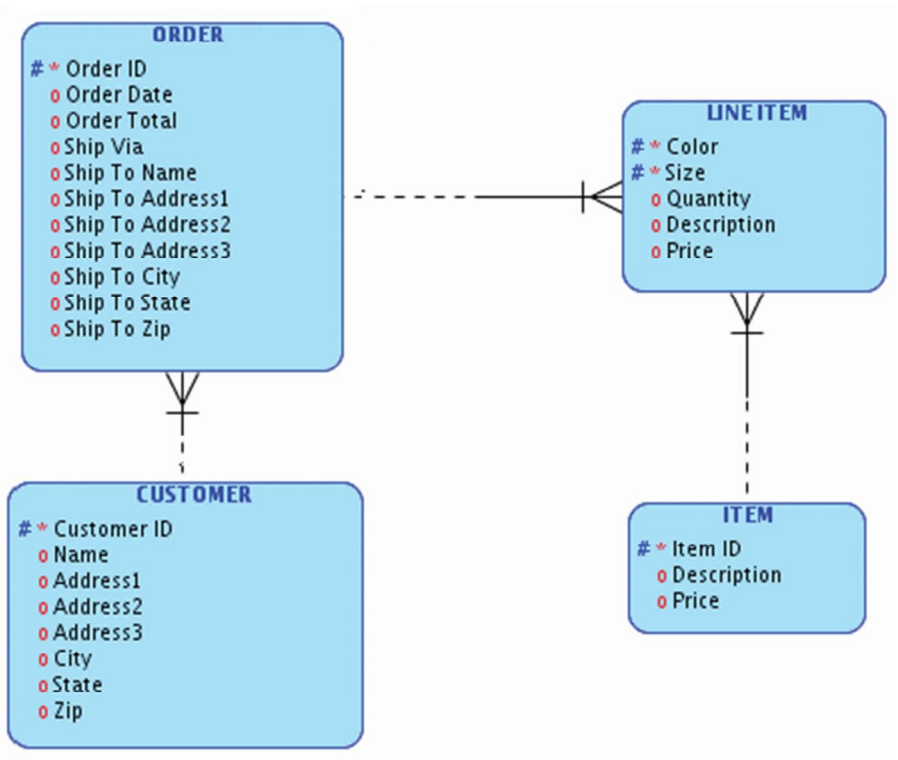


Рисунок 3.5 – Третья нормальная форма

3.7. 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 [25]. В этом случае используют декомпозицию, основанную на многозначных зависимостях.

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

В качестве примера можно рассмотреть сущность «ПРЕПОДАВАТЕЛЬ», атрибутами которой являются «Имя», «Курс», «Учебное_пособие». Из сущности «ПРЕПОДАВАТЕЛЬ» можно получить информацию о курсах, читаемых преподавателем, и написанных им учебниках. Пусть профессор N читает курсы "Теория упругости" и "Теория колебаний" и имеет соответствующие учебные пособия, а профессор K читает курс "Теория удара" и является автором учебников "Теория удара" и "Теоретическая механика". Тогда наше отношение будет иметь вид, представленный в таблице 3.2.

Таблица 3.2. Экземпляры сущности «ПРЕПОДАВАТЕЛЬ»

Имя	Курс	Учебное_пособие
N	Теория упругости	Теория упругости
N	Теория колебаний	Теория упругости

N	Теория упругости	Теория колебаний
N	Теория колебаний	Теория колебаний
K	Теория удара	Теория удара
K	Теория удара	Теоретическая механика
добавление информации		
K	Теория упругости	Теория удара
K	Теория упругости	Теоретическая механика

Экземпляры сущности имеют значительную избыточность, что приводит к возникновению аномалии обновления. Например, добавление информации о том, что профессор K будет также читать лекции по курсу "Теория упругости" приводит к необходимости добавить 2 строки в таблицу, по одной для каждого написанного им учебника, вместо одной.

Указанные аномалии исчезают при замене сущности «ПРЕПОДАВАТЕЛЬ» двумя сущностями: «ПРЕПОДАВАТЕЛЬ_КУРС» (таблица 3.3) и «ПРЕПОДАВАТЕЛЬ_ПОСОБИЕ» (таблица 3.4).

Таблица 3.3. Отношение «ПРЕПОДАВАТЕЛЬ_КУРС»

Имя	Курс
N	Теория упругости
N	Теория колебаний
K	Теория удара
K	Теория упругости

Таблица 3.4. Отношение «ПРЕПОДАВАТЕЛЬ_ПОСОБИЕ»

Имя	Учебное_пособие
N	Теория упругости
N	Теория колебаний
K	Теоретическая механика
K	Теория удара

Аномалия обновления возникает в данном случае потому, что в сущности «ПРЕПОДАВАТЕЛЬ» имеются:

- зависимость множества значений атрибута «Курс» от множества значений атрибута «Имя»;
- зависимость множества значений атрибута «Учебное_пособие» от множества значений атрибута «Имя».

3.8. Пятая нормальная форма

Во всех предыдущих формах единственной операцией, необходимой для устранения избыточности сущности, была декомпозиция ее на две сущности.

Однако существуют отношения, для которых нельзя выполнить декомпозицию без потерь на две сущности, но которые можно подвергнуть декомпозиции без потерь на три (или более) сущностей. Этот факт получил название зависимости по соединению, а такие сущности называют 3-декомпозируемые сущности.

Зависимость по соединению является обобщением многозначной зависимости. Сущности, в которых имеются зависимости по соединению, не являющиеся одновременно ни многозначными, ни функциональными, также характеризуются аномалиями обновления. Поэтому, вводится понятие пятой нормальной формы.

Сущность находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в ней определяется только его возможными ключами [26]. То есть каждая новая сущность, полученная при декомпозиции, содержит не менее одного возможного ключа и не менее одного неключевого атрибута.

3.9. Преобразование отношений

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

Для преобразования M:M отношения необходимо определить атрибуты которые могут быть включены в новую сущность, являющуюся пересечением существующих. В примере на рисунке 3.6 отношение M:M между сущностью «ORDER» (заказ) и «PRODUCT» (продукт) может быть преобразовано, если создать новую сущность «ORDER ITEM», определяющую позицию заказа, уникальным идентификатором в которой 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 SQL Developer Data Modeler у сущностей, имеющих M:M отношение, 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.

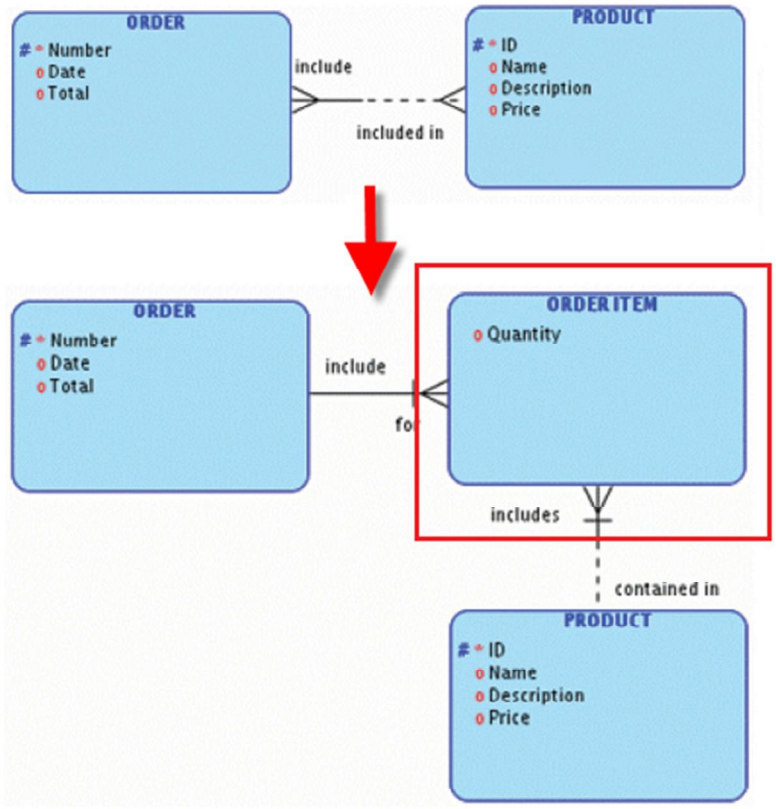


Рисунок 3.6 – Преобразование отношения многие-ко-многим

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 3.7 представлена рекурсивная связь для иерархических данных, используя сущность «ORGANIZATION». Рекурсивная связь не может быть обязательной. Если каждая организация должна входить в другую организацию, иерархия организаций должна была бы быть бесконечно большой. Чтобы этого не происходило, рекурсивное отношение должно быть необязательными 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.

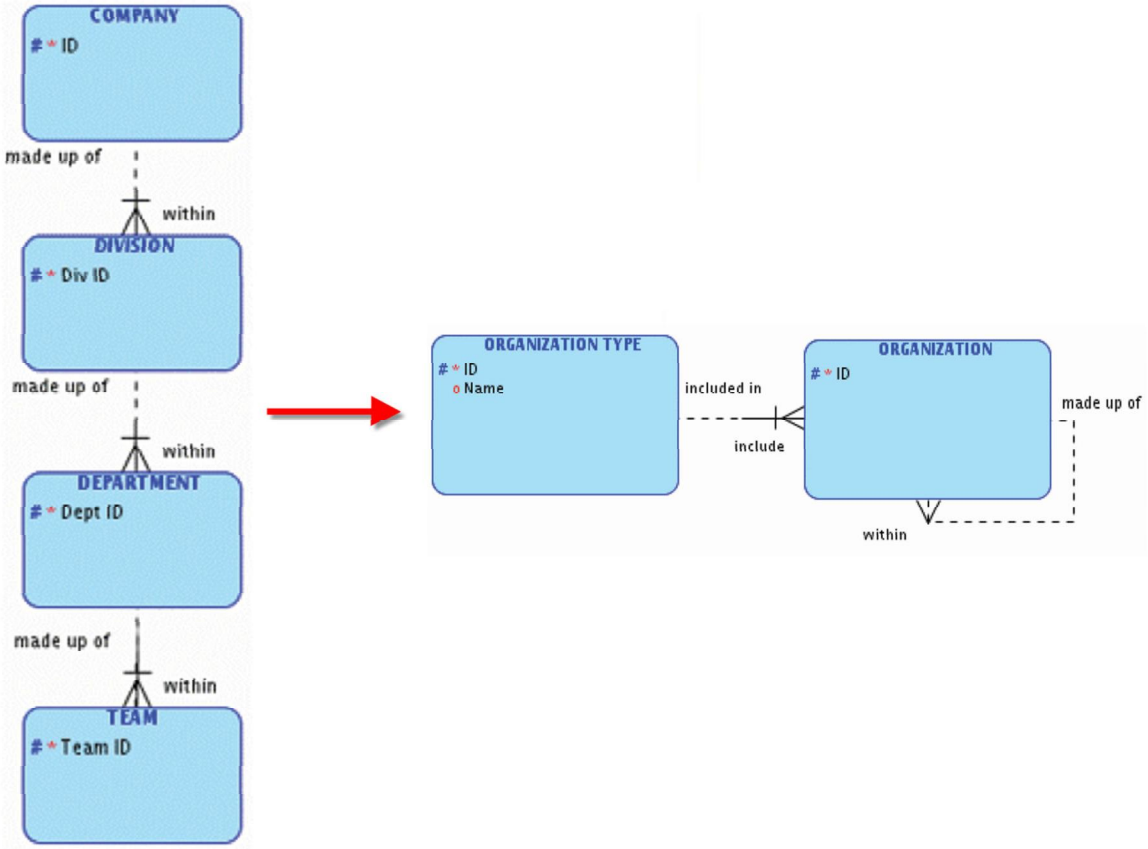


Рисунок 3.7 – Рекурсивная связь для иерархических данных

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

На рисунке 3.8 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.

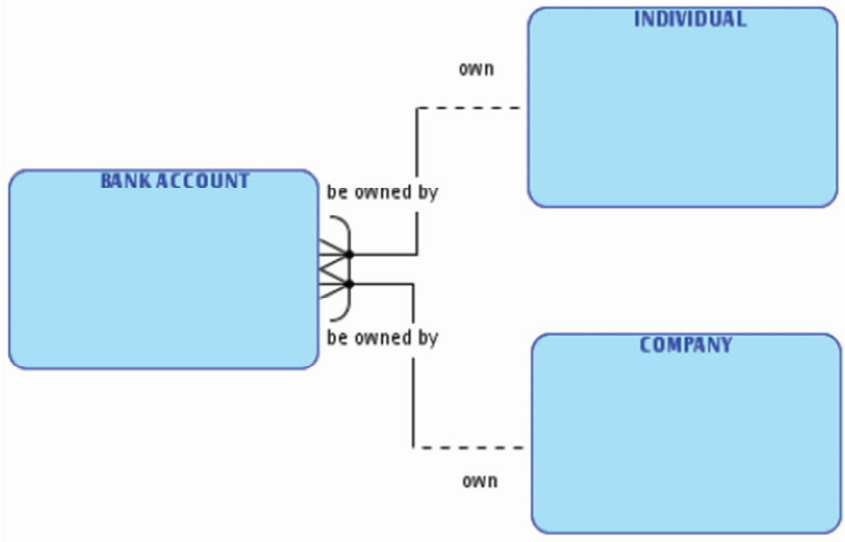


Рисунок 3.8 – Взаимоисключающие отношения

Резюме

В базе данных данные хранятся 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, 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. На практике в реляционной базе данных также не используются отношения многие-ко-многим, они заменяются двумя отношениями многие к одному, и иерархические отношения, которые заменяются рекурсивной связью.

Для закрепления теоретического материала раздела предусмотрено выполнение лабораторных работ №1 и №2. Описание и варианты лабораторных работ содержатся в разделе «Управление данными» электронного курса дисциплины.