Разрабатываем ER-диаграмму и реляционную схему

... на примере высказывания «Сотрудники занимают должности» в реляционном мире.

Сущности на ER-диаграммах будем представлять в виде прямоугольников, атрибуты сущностей — прямоугольниками со скругленными углами (круглоугольниками). Чтобы не загромождать диаграмму «овалами» или круглоугольниками для каждого отдельного атрибута, в одном таком круглоугольнике при необходимости будем размещать несколько атрибутов). Связи будем обозначать ромбами, обычно пронумерованными, но иногда обозначенными глаголом, семантически связывающим сущности.

В случае, когда на ER-диаграмме не указаны атрибуты сущностей, либо указаны только необходимые из них для обсуждения сути вопроса, будем называть такую ER-диаграмму скелетом (ER-скелет). Из необходимых атрибутов сущностей наиболее часто используется наименование/имя (name), но мы, если это не вносит неопределенности, будем опускать. Проектирование на уровне скелетов позволяет не загромождать диаграмму на начальном этапе малосущественными деталями.

Связи также могут иметь атрибуты. Связи между экземплярами сущностей в ряде случаев образуются (начинаются) и исчезают (заканчиваются) в определенный момент времени, в связи с чем в качестве атрибута, который необходимо учитывать изначально, для таких связей принимается дата (date). Под термином «дата» в зависимости от контекста следует понимать как, собственно, дату в виде какого-то дня какого-то месяца какого-то года, так и определенный момент времени.

Мы не будем использовать для обозначения типа связи куриные лапки, вместо этого будем проставлять символы у соответствующих углов ромба, сразу учитывая ограничения согласованности между сущностями в реляционном мире[1].

Рассмотрим несколько вариантов реализации высказывания «Сотрудники занимают должности» на концептуальном и реляционном уровнях. Начнем с самого простого — «детского» варианта, который часто можно встретить в курсовых, да и дипломных работах студентов. В данном случае ER-диаграмма будет содержать одну сущность «Сотрудники[2]», обладающую, помимо всего прочего, атрибутом «должность» (рисунок 1), который на логическом и физическом уровнях может быть реализован в виде строки, содержащей название конкретной должности.

Рисунок 1 — ER-диаграмма для «детского» случая.

Положительным свойством такого решения является исключительная простота, понятная даже студенту-троечнику, а также минимализм реализации такого решения на логическом

1 В реляционном мире, например, не может быть связей типа «многие к нолю», что означало бы нарушение ссылочной согласованности.

2 Будем именовать сущности во множественном числе, что, в принципе, необязательно, однако ассоциируя сущность со множеством, это кажется разумным. С единственным число в данном случае ассоциируется экземпляр сущности, как элемент множества.

и физическом уровнях. Еще одним положительным свойством является слабая чувствительность к ошибкам ввода, поскольку «детские» варианты не предусматривают лексического и грамматического контроля, поддерживаемого справочниками. Одним словом, «строковое поле все стерпит». Однако, это положительное свойство одновременно является и отрицательным — согласитесь, что варианты «Генеральный директор» и «Гинеральный директор» лексически близки и, скорее всего, на экране разница останется незамеченной.

Очевидным «техническим» недостатком данного решения является тот факт, что если должность, занимаемая сотрудником, имеет слишком длинное название, внешняя и внутренняя память будет расходоваться крайне нерационально.

Чтобы устранить возможные ошибки ввода, вместо представления должности в виде атрибута, обычно используют ссылку на одну из должностей, перечисленных в соответствующем справочнике, не содержащем ошибок и существующем независимо от сущности «Сотрудники». Одновременно более рационально будет использоваться как внешняя память, так и оперативная память приложения.

Таким образом, в игру вступает еще одна сущность — «Должности». Поскольку некоторую должность могут занимать несколько сотрудников, скелет ER-диаграммы (ER-скелет) высказывания «Сотрудники занимают должности» приобретает следующий вид:

Рисунок 2 — ER-скелет концепции «Сотрудники занимают должности».

Здесь следует сказать об «активной» и «пассивной» точке зрения на связи между сущностями. На рисунке 2 представлена «активная» точка зрения на связь, как «Сотрудники занимают должности». Пассивная точка зрения была бы представлена, как «Должности занимаются сотрудниками».

Символы 'N' и '1' слева и справа от ромба «занимают» представляют собой тип связи. В данном случае они выражают тот факт, что одна и та же должность может быть занята несколькими сотрудниками, а сотрудник может занимать только одну из должностей. Например, в столовой могут работать несколько сотрудников в должности «повар», но сотрудник не может занимать более одной должности, тем не менее на свете много таких предприятий и организаций, где именно так все и организовано.

Тип связи обозначается как 1:1 — связь типа «один к одному», N:1 — связь типа «многие к одному», N:M — связь типа «многие ко многим». Слово «многие» в данном контексте означает ноль, один или больше одного.

Так же есть понятие «валентность связи». Оно определяет сколько сущностей объединяется связью. В данном случае валентность связи равна двум.

Казалось бы, вот оно решение, пусть и не всеобъемлющее. Однако реальная жизнь сложнее и через некоторое время реализация ИС в соответствии с нашей диаграммой перестает учитывать тот факт, что сегодня сотрудник «повар», а завтра его *повысили*. Это и есть вышеупомянутая зависимость связи от времени — она возникает между экземплярами сущностей в некоторый момент времени (назначение на должность) и в некоторый следующий за ним исчезает (увольнение с должности).

Сущности — это абстрактные модели реальных субъектов и объектов, соответственно они существуют не только как «вещи в себе[3]», а также в пространстве и времени. Как минимум, во времени. Соответственно, ER-связь должна отражать не только свое сиюминутное

3 Кант, И. Критика чистого разума. Пер. Н. М. Соколова. В 2 вып. — СПб., М. В. Попов, 1896—1897.

состояние, но и прошлое, а также предусматривать ее изменения в будущем (хотя бы в ее текущей конфигурации). Иными словами, даже если сотрудник никогда не будет иметь возможности занимать несколько должностей одновременно, должен учитываться тот факт, что завтра его могут назначить на новую должность. И если новая должность в ИС просто заменит старую, история занятости сотрудника, его послужной список, теряется, что никого не устраивает.

Также отметим, что одновременное занятие нескольких должностей одним сотрудником весьма распространено, например, нянечка в детском садике может работать на пол- или четверть ставки в качестве технического персонала, в связи с чем ER-диаграмма должна учитывать и такую возможность. Итак, мы приходим к следующей схеме связи между сотрудниками и занимаемыми должностями:

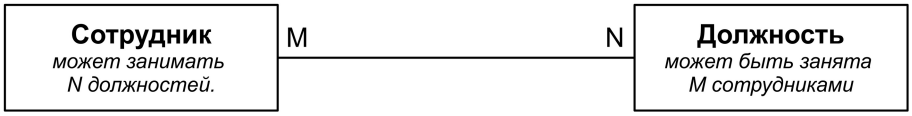

Рисунок 3 — Схема связи между сотрудниками и занимаемыми должностями.

Соответствующий данной схеме скелет ER-диаграммы будет иметь следующий вид:

Рисунок 4 — Скелет ER-диаграммы в случае, более приближенном к реальности.

В принципе, этого уже достаточно для создания рабочей реляционной схемы концепции «Сотрудники занимают должности» и мы далее создадим ее *скелет*, то есть только ту часть, которая отражает связи и, возможно, минимально необходимые атрибуты, уточняющие некоторые моменты, связанные с представлением сущностей в пользовательском интерфейсе.

Прежде чем начать преобразование, уточним некоторые термины и как они соотносятся с друг с другом, а также форму представления реляционной схемы. Реляционные системы хранения и управления данными хранят свои данные в виде двумерных таблиц — прямоугольных массивов, состоящих из элементов, каждый из которых принадлежит конкретной строке и конкретному столбцу. Все столбцы в таблице однородные — элементы одного столбца имеют одинаковую природу. Столбцам однозначно присвоены имена. В таблице нет двух одинаковых строк — все строки уникальны и могут быть однозначно идентифицированы.

В реляционной теории таблицы называются отношениями, строки кортежами, столбцы атрибутами. Также вместо кортежа можно встретить слово «запись», а вместо атрибута слово «колонка» или «поле». Таким образом, смешивая терминологию, таблица состоит из кортежей. Если таблица имеет n столбцов, кортежи являются n-мерными. Каждый из n элементов кортежа представляет собой упорядоченную тройку в форме $< A_i, T_i, v_i >$, где A_i — имя атрибута, T_i — имя типа и v_i — значение типа. Упорядоченная пара $< A_i, T_i >$ представляет атрибут и однозначно определяется его именем A_i, т.е. имена атрибутов (столбцов) в рамках таблицы являются уникальными. Тип T_i — это соответствующий тип атрибута, никак не связанный с типом представления его значений. Например, в рамках одного кортежа может существовать два атрибута «дата начала» и «дата завершения» с типами «тип_даты», значения которых могут иметь разные форматы хранения на физическом уровне[4]. В литературе можно встретить классическую нотацию для реляционного продолжения ER-диаграмм в виде набо-

4 Козе известно, что лучше всего даты и время хранить в виде модифицированной юлианской даты.

ра строк типа `TUPLE{A1:T1, A2:T2, ..., An:Tn}`, однако мы будем использовать графическую форму, представленную на рисунке 5, и называть ее таблицей для реляционной схемы или просто таблицей[5].

Наша табличная форма состоит из двух столбцов и имеет общую для них верхнюю строку. Текст в этой строке, набранный полужирным шрифтом, является именем таблицы. Строки, расположенные ниже, в правом столбце содержат имена атрибутов реляционной модели и отображаются на столбцы таблицы в безе данных. Типы атрибутов на реляционной схеме не указываются, чтобы ее не загромождать[6], и приводятся вместе с подробным описанием в пояснительной записке.

employee	
P	id
S	name
S	no
	birthday
	class

Рисунок 5 — Графическое представление таблицы для реляционной схемы.

Все имена на реляционной схеме, включая имя таблицы, задаются в том виде, в котором они появятся в декларациях и операторах доступа. В данном случае это слова английского языка в нижнем регистре, являющиеся переводом русских слов, обозначающих сущности и атрибуты. Подробнее о том, как выбирать имена таблиц и столбцов, см. ниже.

Ячейки, расположенные под ячейкой с именем таблицы разбиты на две каждая, образуя два столбца — один с именем поля, другой содержит флаг семантики поля. В данном случае флаг семантики поля представлен символами `'P'`, `'F'` и `'S'`, где `'P'` — первичный ключ (Primary key), `'F'` — внешний ключ (Foreign key) и `'S'` — вторичный ключ (Secondary key).

Первичный ключ однозначно идентифицирует запись и используется для прямого доступа к ней. Внешний ключ ссылается на первичный ключ другой таблицы (содержит значение того первичного ключа) и используется для связи таблиц друг с другом. Обычно используется для последовательного доступа к группе записей при итерировании по дубликатам данного ключа. Вторичный ключ используется для упорядочения последовательного доступа к данным в порядке, устанавливаемом данным ключом (при диапазонном доступе). Например, для вывода сотрудников в алфавитном порядке их ФИО. Для такого ключа создается отдельный индекс. В распространенной литературе вместо `P`, `F` можно встретить обозначения `PK`, `FK`, однако односимвольных флажков для указания семантики столбца более чем достаточно.

Итак, вернемся к преобразованию ER-диаграммы в реляционную схему. Правила такого преобразования достаточно просты и мы здесь кратко их приведем. Пояснять правила будем, используя ER-диаграммы на рисунках 2 и 4.

1) Все сущности ER-диаграммы преобразуются в отдельные таблицы. В случае, приведенном на рисунке 2, сущности «Сотрудники» и «Должности» преобразуются в таблицы «`emplyee`» и «`position`», как представлено на рисунке .

5 Слово «таблица» является перегруженным — в зависимости от контекста оно может обозначать таблицу в базе данных (отношение), таблицу на реляционной схеме (графическое отображение таблицы в БД), таблицу-виджет пользовательского интерфейса и другие регулярные формы.

6 В большинстве реальных случаев реляционная схема слишком велика, чтобы разместиться на формате А3.

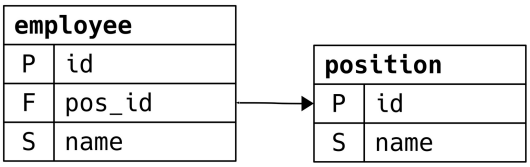

Рисунок 6 — Скелет реляционной схемы для ER-диаграммы на рисунке 2.

2) Каждый из атрибутов сущности превращается в строку таблицы реляционной схемы. В частности, атрибут «Имя» сущности «Сотрудники» и атрибут «Наименование» сущности «Должности» превращается в строку «`name`» в обеих таблицах в соответствии с правилами именования, представленными в приложении 1 Как выбирать имена таблиц и столбцов.

3) В каждую таблицу реляционной схемы, сгенерированной из сущности, добавляется столбец первичного ключа. В частности, в упомянутые выше две таблицы добавляются строки, поименованные как `"id"`. Ссылки в коде на данные столбцы будут иметь вид `"employee.id"` и `"position.id"`, соответственно.

3) Связь типа N:1 превращается в поле внешнего ключа (`pos_id`) в таблице, соответствующей сущности со стороны связи, указанной, как N. Такая связь чаще всего является ссылкой на справочник или таблицу поиска (Lookup Table — LUT). Содержимое поля внешнего ключа в таблице базы данных в момент установления связи устанавливается в значение первичного ключа таблицы-справочника. На реляционной диаграмме такая связь указывается линией, начинающейся на поле внешнего ключа, и завершающейся на поле первичного ключа таблицы-справочника. Такая линия может завершаться стрелкой, обозначая направление ссылки. Сущность-справочник всегда обозначается символом `'1'`, поскольку «висящих» ссылок типа N:0 на ER-диаграммах реляционного мира не допускается.

Теперь приступим к преобразованию ER-диаграммы рисунка 4 в реляционную схему. Все сущности (у нас их две), как и в случае ER-диаграммы на рисунке 2, преобразуются в таблицы сущностей «`emplyee`» и «`position`», в которые добавляются поля первичных ключей `"id"`. Со связью немного интереснее — связь на рисунке 4 отличается тем, что она является связью типа N:M («многие ко многим»), в отличие от связи от связи на рисунке 2, которая является связью типа N:1 («многие к одному»).

4) Связи типа «многие ко многим» преобразуются в отдельные таблицы, содержащие внешние ключи, ссылающиеся на первичные ключи таблиц связываемых сущностей. Это показано на рисунке 7.

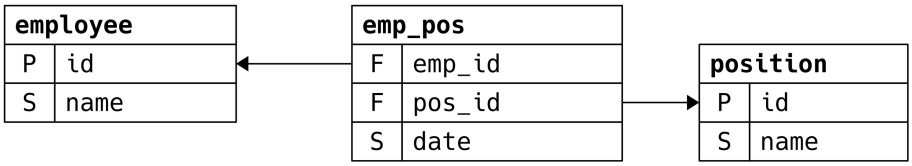

Рисунок 7 — Скелет реляционной схемы для ER-диаграммы в случае, более приближенном к реальности.

Связь «занимают» типа N:M преобразуется в таблицу «`emp_pos`» с парой полей внешних ключей, содержащих ссылки на первичные ключи в таблицах «employee» и «`position`». Содержимое этих полей — это значения первичных ключей, на которые они ссылаются.

Все связи из ER-диаграмм реляционного мира, какого бы типа и валентности они бы не были, при их преобразовании в реляционную схему, в конечном счете, распадаются на связи типа N:1, где $N \geqslant 0$, которые на реляционных схемах представляются линиями, соединяющими поля таблиц. Эти линии исходят из полей, содержащих внешние ключи, и заканчивающимися на полях с первичными ключами. Иногда (как в данном случае) их снабжают стрелками, указывающими направление ссылки.

Надо понимать, что связи между сущностями ER-моделей реляционного мира не образуют какой-либо иерархии наследования или зависимости, как это имеет место между объектами в объектном мире, согласно OO-парадигме[7], поэтому поэтому стрелки можно опускать, если соблюдаются правила именования полей, однозначно идентифицирующие поля первичных и внешних ключей. По этой же причине мы не используем иконографику UML и сам UML, поскольку это разработано для объектного мира и не годится для описания и проектирования реляционного (они банально не пересекаются). Все попытки привязать UML для описания реляционного мира имеют под собой исключительно коммерческий интерес.

Так же нужно отметить, что между проекциями сущностей на физическую реализацию в реляционном мире нет физической связи, как этот имеет место в объектном (физические указатели (ссылки) на экземпляры), а только тот факт, что содержимое полей первичного и ссылающихся на него внешних ключей содержат одно и то же значение. Иными словами, в базе данных никакой информации о том, что сущность А каким-то образом связана с сущностью Б, не хранится. Эта связь обеспечивается прикладным кодом, который использует либо механизм JOIN, либо явные алгоритмы для ее проявления.

Итак, если решено использовать линии со стрелками, такая линия начинается на поле внешнего ключа и завершается на поле первичного — это, кстати, отображает направление допустимого прямого доступа от записи одной таблицы к записи другой (в обратном направлении из-за того, что внешние ключи не являются уникальными, прямой доступ невозможен).

О вторичных ключах

Вторичный ключ используется для последовательного доступа или вывода полей, не являющимися первичными ключами, в указанном порядке предшествования. Флажок `'S'` вторичного ключа обычно является указанием на то, что данное поле должно индексироваться для последовательного доступа (второе значение символа 'S' — сортировка). К таким полям относятся все те, которые в приложении выводятся в виде скроллируемых списков и `name` – это такое поле чуть менее, чем всегда. К таблицам, содержащим такие поля (Lookup Tables) относятся справочники, на строки которых ссылаются по их первичному ключу, используя операторы прямого доступа или объединения.

Обычно, если скроллируемый объем достаточно мал, чтобы полностью разместиться в памяти приложения целиком, соответствующий индекс для такого поля не строится, вместо этого полученный массив сортируется в памяти для вывода в нужном порядке. Однако, когда скроллируемый объем записей превышает возможности приложения в отношении его размещения, приходится явно использовать курсоры или строить индекс для сохранения приемлемой отзывчивости базы данных.

Итак, на скелете реляционной схемы (рисунок 8) мы видим три таблицы, две из которых, `employee` и `position`, соответствуют нашим сущностям, и третью, `emp_pos`, реализующую связь многие ко многим. Фактически таблица `emp_pos` представляет собой внутренность примитивной «трудовой книжки» предприятия.

Еще ближе к реальности

Для полноты изложения следует рассмотреть более сложную связь между более чем двумя сущностями (валентности > 2), которая обозначается ромбом с более чем двумя соединениями. Попробуем развить наш ER-скелет с рисунка 4, учитывая то, что любые назначения на должности и увольнения с них прежде всего отражаются в приказах руководства.

7 В реляционном мире нет ни объектов, ни иерархии наследования, ни чего бы там ни было вообще, что есть в ОО мире, а лексические совпадения терминов из обоих миров следует считать случайными.

Приказ из существенных атрибутов имеет номер, форматируемый по установленным правилам, и дату. На ER-диаграмме (рисунок) мы их опустим, но на реляционной схеме приведем.

Добавим в ER-диаграмму сущность «Приказы», как это показано на рисунке 9. Связь, обозначенная номером 1, говорит нам о том, что сотрудники могут занимать несколько должностей, несколько сотрудников могут занимать одну и ту же должность, а также что в одном приказе могут быть несколько записей о назначении/увольнении.

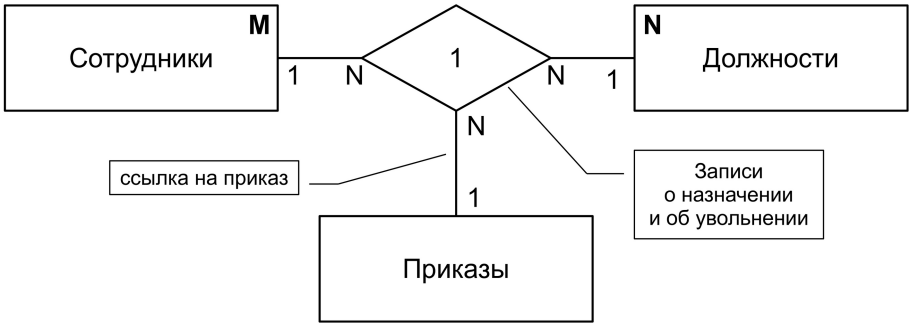

Рисунок 9 — Скелет ER-диаграммы в случае, еще более приближенном к реальности.

Процессе преобразования ER-диаграммы в реляционную схему заключается в создании таблицы `"order"` для сущности «Приказы» и добавления в таблицу `"emp_pos"` поля `"ord_id"` для внешнего ключа, ссылающегося на таблицу `"order"`. Результат показан на рисунке 10.

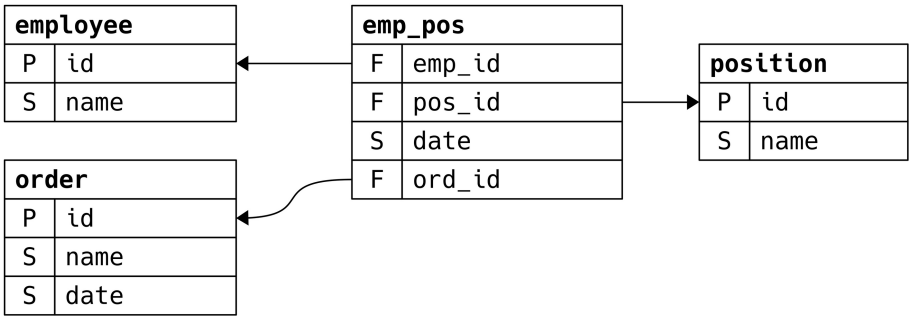

Рисунок 10 — реляционная схема для ER-модели с рисунка 9.

Разложение связи N:M на на связи N:1

Разложение связей N:M между сущностями на связи N:1 можно выполнить до преобразования ER-диаграммы в реляционную схему. Для этой цели необходимо создать отдельную сущность, которая будет конкретизировать связь типа N:M между исходными сущностями. Такой сущностью может быть факт возникновения частных связи между экземплярами сущностей в некоторый момент времени.

Для случая диаграммы на рисунке 9 такой сущностью может быть множество фактов возникновения частной связи между сотрудником, должностью и приказом в некоторый момент времени. Пусть это будет сущность «Записи в трудовой книжке», подразумевающая записи о конкретном назначении или увольнении с должности, согласно приказу (рисунок 11).

Данная ER-диаграмма стандартным образом преобразуется в реляционную схему на рисунке 10.

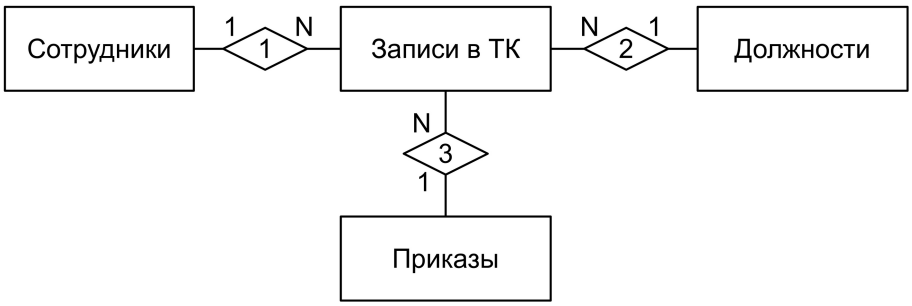

Рисунок 11 — Скелет детализированной ER-диаграммы с рисунка 9.

Связи с валентностью больше двух

В случае, когда валентность связи превышает два, ее можно разложить на связи валентности два, выделив из многовалентной связи некоторую в двухвалентную связь в отдельную сущность.

1 Как выбирать имена таблиц и столбцов

О том, как выбирать имена таблиц и столбцов, как при создании схемы таблицы для реляционной схемы, так и при создании таблицы в базе данных написано много страниц, рекламируя тот или иной подход. Мы тоже внесем свою лепту в этот процесс и коснемся двух мало различающихся по сути вариантов, приведя некоторые обоснования относительно их выбора.

1) все имена на реляционной схеме, включая имя таблицы, задаются в том виде, в котором ни появятся в декларациях и прочих операторах. Именовать столбцы следует словами английского языка, которые являются семантически корректным переводом русских слов, обозначающих сущности и атрибуты. Почему не используем кириллицу? Потому, что некоторые языки не поддерживают в именах символы, не входящих в первую половину ASCII-таблицы. Те же, которые поддерживают, требует специального экранирования. Почему тогда не используем транслитерацию? Потому, что гладиолус.

2) регистр всех наших имен, включая имена таблиц и атрибутов наших атрибутов либо нижний, либо верхний без смешивания.

Дело в том, что ключевые слова специальных языков, предназначенных для работы с данными, а также имена их стандартных процедур и функций допускают либо только верхний регистр, либо только нижний, либо вовсе регистра не различают. В последнем случае настоятельно рекомендуется выбрать регистр и его использовать для ключевых лексем, не смешивая. Таким образом регистр всех остальных лексем, как то имена таблиц и атрибутов (полей) набирается в оппозитном регистре. Т.е. если ключевые слова в верхнем, все наши имена в нижнем и наоборот. Такой подход существенно облегчает распознавание классов лексем человеком и является зарекомендовавшей себя практикой.

3) именовать таблицы следует в единственном числе в отличие от именования сущностей на ER-диаграммах, где используется множественное число;

Дело в том, что имя таблицы постоянно используется в коде для уточнения целевых столбцов и использование его во множественном числе в выражениях нарушает семантическое и отчасти синтаксическое согласование (сравните «`должности.наименование = повар`» против «`должность.наименование = повар`»).

4) для таблиц помимо их имени следует создавать короткие но уникальные в рамках базы данных алиасы;

Некоторые языки поддерживают перманентные алиасы, которые могут назначаться именам таблиц как на постоянной основе, так и только на время сеанса. Даже если применения в коде алиасов не предусматривается, короткие имена могут использоваться в именах внешних ключей (например, имя внешнего ключа, ссылающегося на первичный ключ таблицы `employee`, имеющей алиас `emp`, может иметь имя `emp_id` вместо `employee_id`, что укорачивает строки кода, препятствуя их вынужденному переламыванию, а также сокращает и усилия по набору).

5) компоненты составных имен разделяются символом подчеркивания (`part_no`, `employee_id`), при этом количества компонент более трех следует избегать;

6) по возможности всегда следует выбирать короткие имена, без избыточности связывающие семантику атрибута и его имя;

Например, вместо имени столбца `birthday_date` в большинстве случаев достаточно использовать `birthday`. То же самое касается именования столбцов, содержащих наименования и/или имена — в подавляющем большинстве все они могут быть названы `"name"`.

Также нет никакого смысла в имени столбца упоминать имя его таблицы, поскольку имя таблицы в составе ссылки на столбец в любом случае в коде будет либо присутствовать явно, либо однозначно подразумеваться («`employee.name`» vs «`employee.employee_name`»).

7) в качестве имен всех полей, являющихся первичным ключом, следует использовать одно и то же имя для всех таблиц, например, `"id"`.

Данное поле никогда не появляется перед пользователями, поскольку заполняется автоматически при создании записи и никогда более не изменяется. Также данное имя визуально идентифицирует поле как ключевое.

8) имя внешнего ключа включает имя таблицы первичного ключа или ее алиаса и суффикс, сформированный из имени первичного ключа `"_id"` (`employee_id`, `emp_id`).

2 Как выбирать, что у нас будет в качестве первичного ключа

Классическая теория баз данных, которая разрабатывалась в те времена, когда оперативную память приложений считали в байтах и внешнюю в метрах магнитной ленты, в отношении выбора атрибутов (полей) для первичных ключей кортежа (записи) сгенерировала ряд правил:

1) кортеж должен однозначно определяться значением ключа (однозначная идентификация записи);

2) никакой атрибут нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации (отсутствие избыточности);

Для записей одной таблицы может существовать несколько наборов полей, удовлетворяющих вышеприведенным свойствам. Такие наборы называются возможными ключами. Тот ключ, который фактически будет использоваться для идентификации записей, называется первичным ключом.

Однако практика показала, что для таблиц, которые являются отображением сущностей, со всех сторон выгодно ввести отдельное поле, которое служит исключительно в качестве первичного ключа, невзирая на уникальность каких-либо полей и/или их наборов. В пользу отдельного поля и против набора информационных полей для первичного ключа можно привести следующие соображения. Представим, что для учета сотрудников в некоторой организации используется некоторый код, содержащий код предприятия и код сотрудника в формате, установленном теми или иными руководящими документами. Данный код содержится в поле таблицы сотрудников и принят в качестве первичного ключа. При изменении этих руководящих документов в отношении правил кодирования все операции с персоналом будут приостановлены на время, которое потребуется для приведения базы данных в новое состояние ссылочной целостности. Использование в качестве первичного ключа отдельного поля никак не повлияет на ссылочную целостность базы, доступ к которой придется остановить всего-лишь на небольшое время, необходимое для перезаписи поля, содержащих этот самый номер.

Это же относится и к таблицам, которые являются отображением связей типа «многие ко многим», где иногда связку нескольких внешних ключей трактуют в качестве первичного. В принципе, ничего страшного в этом нет, поскольку, обеспечивая ссылочную целостность, данные ключи измениться не могут, однако нарушается другой принцип — первичные ключи используются только для прямого доступа, внешние — в качестве селекторов диапазонного поиск и в качестве ссылок на первичные ключи.

Итак, в качестве первичных ключей в таблицах, проектируемых на сущности, используем отдельные поля с соответствующими первичным ключам свойствами, тем более, что в распространенных языках для этой цели предусмотрены специальные средства.