КОНСПЕКТ ЗАНЯТИЯ № 2 ВТОРОЙ НЕДЕЛИ КУРСА «БАЗЫ ДАННЫХ»

2.1. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ: ЦЕЛОСТНОСТЬ

В прошлом конспекте была рассмотрена структурная часть реляционной модели, фундаментальным компонентом которой являются отношения.

Тема данного конспекта посвящена следующей части реляционной модели – целостная часть, которая описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Каждое такое ограничение предназначено для обеспечения соответствия информации, содержащейся в базе данных, предметной области: в любой момент времени данные во всех отношениях базы должны быть точными, актуальными и непротиворечивыми.

Непротиворечивость означает невозможность сделать разные выводы об одном и том же объекте или явлении реального мира.

Соответствие действительности можно проверить только до некоторой степени. Например, мы можем заставить СУБД принимать в качестве значения зарплаты только числа в диапазоне от 5000 до 100000, но мы не сможем научить её проверять, действительно ли конкретно у сотрудника Васильчикова зарплата должна быть 50000.

Все ограничения целостности можно условно разделить на три категории:

1) ограничения для поддержки целостности атрибутов[1];

2) ограничения для поддержки целостности отношений (ключи);

3) ограничения для поддержки ссылочной целостности, т.е. связей между отношениями.

[1] В некоторых изданиях первое ограничение явно не выделяют, поскольку есть разногласия по использованию NULL значений. Однако, поскольку Кодд (основатель реляционной модели) ввел понятие NULL значения, то я выделила данное ограничение в обособленном виде.

2.1.1. Ограничения для поддержки целостности атрибутов

Основное назначение баз данных состоит в том, чтобы хранить и предоставлять информацию о реальном мире. Для представления этой информации в базе данных используются привычные для программистов типы данных - строковые, численные, логические и т.п.

Однако в реальном мире часто встречается ситуация, когда данные неизвестны или не полны. Например, место жительства или дата рождения человека могут быть неизвестны (база данных разыскиваемых преступников). Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты? Ответ - пустую дату - не вполне удовлетворителен, т.к. простейший запрос "выдать список людей в порядке возрастания дат рождения" даст заведомо неправильных ответ.

Для того чтобы обойти проблему неполных или неизвестных данных, в базах данных могут использоваться типы данных, пополненные так называемым ***null-значением***. Null-значение - это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно.

Таким образом, в ситуации, когда возможно появление неизвестных или неполных данных, разработчик имеет на выбор два варианта.

Первый вариант состоит в том, чтобы ограничиться использованием обычных типов данных и не использовать null-значения, а вместо неизвестных данных вводить либо нулевые значения, либо значения специального вида - например, договориться, что строка "АДРЕС НЕИЗВЕСТЕН" и есть те данные, которые нужно вводить вместо неизвестного адреса. В любом случае на пользователя (или на разработчика) ложится ответственность на правильную трактовку таких данных. В частности, может потребоваться написание специального программного кода, который в нужных случаях "вылавливал" бы такие данные. Проблемы, возникающие при этом очевидны - не все данные становятся равноправны, требуется дополнительный программный код, "отслеживающий" эту неравноправность, в результате чего усложняется разработка и сопровождение приложений.

Второй вариант состоит в использовании null-значений вместо неизвестных данных. За кажущейся естественностью такого подхода скрываются менее очевидные и более глубокие проблемы. Наиболее бросающейся в глаза проблемой является необходимость использования

трехзначной логики при оперировании с данными, которые могут содержать null-значения. В этом случае при неаккуратном формулировании запросов, даже самые естественные запросы могут давать неправильные ответы. Есть более фундаментальные проблемы, связанные с теоретическим обоснованием корректности введения null-значений, например, непонятно вообще, входят ли null-значения в домены или нет.

Вышеуказанные проблемы привели к двоякому мнению: основоположник реляционного подхода Кодд считал null-значения неотъемлемой частью реляционной модели. К.Дейт, один из крупнейших теоретиков реляционной модели выступает категорически против null-значений.

Практически все реализации современных реляционных СУБД позволяют использовать null-значения, несмотря на их недостаточную теоретическую обоснованность[2], по этой причине необходимо исследовать концепцию трехзначной логики.

Трехзначная логика (3VL)

Т.к. null-значение обозначает на самом деле тот факт, что значение неизвестно, то любые алгебраические операции (сложение, умножение, конкатенация строк и т.д.) должны давать также неизвестное значение. Действительно, если, например, вес детали неизвестен, то неизвестно также, сколько весят 10 таких деталей.

Таким образом, определение истинности логических выражений базируется на ***трехзначной логике*** (***three-valued logic, 3VL***), в которой кроме значений T (true) - ИСТИНА и F (false) - ЛОЖЬ, введено значение U (unknow) - НЕИЗВЕСТНО. Логическое значение U - это то же самое, что и null-значение.

[2] Такую ситуацию можно сравнить с ситуацией, сложившейся в начале века с теорией множеств. Почти сразу после создания Кантором теории множеств, в ней были обнаружены внутренние противоречия (антиномии). Были разработаны более строгие теории, позволяющие избежать этих противоречий (конструктивная теория множеств). Однако в реальной работе большинство математиков пользуется классической теорией множеств, т.к. более строгие теории более ограничены и негибки в применении именно в силу своей большей строгости.

Трехзначная логика базируется на следующих таблицах истинности:

Таблица истинности AND

AND	F	T	U
F	F	F	F
T	F	T	U
U	F	U	U

Таблица истинности OR

OR	F	T	U
F	F	T	U
T	T	T	T
U	U	T	U

Таблица истинности NOT

NOT	
F	T
T	F
U	U

При сравнении двух неизвестных значений (NULL=NULL?)или любого известного с неизвестным (NOT NULL=NULL?) мы получаем результат вида U, поскольку по крайней мере одно из сравниваемых значений нам неизвестно.

NULL = NULL? → UNKNOWN
NOT NULL = NULL? → UNKNOWN

2.1.2. Ограничения для поддержки целостности отношений

(В некоторых источниках данное ограничение именуют ограничением сущностей).

Согласно свойствам отношения, все его кортежи должны отличаться друг от друга, т.е. мы не должны дублировать в отношении утверждения об одном и том же объекте (явлении) реального мира. Для этого нам необходимо иметь возможность уникальной идентификации каждого отдельного кортежа отношения по значениям его атрибутов. Помогают нам в этом ограничения-ключи.

Пусть дано отношение R. Подмножество атрибутов K отношения R будем называть потенциальным ключом, если K обладает следующими свойствами:

- Свойством уникальности - в отношении не может быть двух различных кортежей, с одинаковым значением K.
- Свойством неизбыточности - никакое подмножество в K не обладает свойством уникальности.

Любое отношение имеет по крайней мере один потенциальный ключ. Действительно, если никакой атрибут или группа атрибутов не являются

потенциальным ключом, то, в силу уникальности кортежей, все атрибуты вместе образуют потенциальный ключ.

Например, на рисунке 3.1 потенциальным ключом отношения R является множество атрибутов {Атрибут 1, Атрибут 2, Атрибут 3}. Можно заметить, что значения каждого из этих атрибутов в отдельности повторяются, но совокупность всех значений кортежа, входящих в ключ – уникальна (последовательности 10-7-22, 10-22-7 и 15-22-7 – различны).

Атрибут 1	Атрибут 2	Атрибут 3
10	7	22
10	22	7
15	22	7

Рисунок 3.1. Пример состояния отношения R

Отношение может иметь несколько потенциальных ключей. Традиционно, один из потенциальных ключей объявляется **первичным (PK, Primary Key)**, а остальные – **альтернативными (AK, Alternate Key)**.

В базе данных определены соответствующие **ограничения типа PRIMARY KEY** (для первичного ключа) и **UNIQUE** (для альтернативных ключей). Эти ограничения не позволяют совокупным значениям атрибутов, входящих в ключ, повторяться.

При выборе первичного ключа ориентируются на следующие критерии:

- ✓ все атрибуты, входящие в его состав должны быть обязательными, то есть не должны содержать неизвестных значений (так как первичный ключ служит идентификатором объектов предметной области, то значения этих идентификаторов не могут содержать неизвестные значения.);
- ✓ значения атрибутов первичного ключа не должны меняться со временем, так как именно этот ключ будет использоваться в качестве основного критерия для идентификации отдельных кортежей;
- ✓ также значения первичного ключа используются при поиске данных в базе, поэтому в целях увеличения скорости поиска в качестве

первичного ключа стараются выбрать такой, совокупная допустимая длина значений атрибутов которого – минимальна.

Проще всего для соблюдения всех трёх критериев добавить в отношение дополнительный атрибут, значения которого будут браться из некоторой последовательности неповторяющихся чисел, и назначить его первичным ключом. Такие первичные ключи называются **суррогатными** и всегда должны быть поддержаны альтернативными, состоящими из атрибутов, соответствующих естественным характеристикам объекта реального мира. Суррогатные первичные ключи совершенно не пригодны для идентификации реальных объектов, из-за чего их использование в базе данных в дальнейшем может привести к проблемам, особенно если их значения генерируются автоматически.

Например, представим, что в качестве первичного ключа таблицы со сведениями о сотрудниках организации был выбран некоторый идентификатор ID_сотрудника, автоматически генерируемый системой при добавлении новой строки (см. рисунок 3.2). Предположим, что оператору требуется внести данные о новых сотрудниках, и он забыл, что до перерыва на кофе уже внёс информацию о Васильчикове. СУБД позволит добавить строку с данными Васильчикова второй раз, сгенерировав для него новое значение ID_сотрудника и формально посчитав его другим человеком. Если бы для таблицы был определён альтернативный ключ – например, ФИО сотрудника, такого бы не произошло, так как СУБД не пропустила бы повторение ФИО.

ID_сотрудника (PK)	Фамилия	Имя	Отчество	Оклад
…	…	…	…	…
105	Васильчиков	Степан	Игнатьевич	50000
106	Васильчиков	Степан	Игнатьевич	50000

Рисунок 3.2. Пример использования суррогатного первичного ключа без поддержки его альтернативным

Потенциальный ключ, состоящий из одного атрибута, называется **простым.**

Потенциальный ключ, состоящий из нескольких атрибутов, называется **составным.**

2.1.3. Ограничения для поддержки ссылочной целостности (связей между отношениями)

Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны друг с другом. Например, накладная на поставку товара содержит список товаров с количествами и ценами, сотрудник предприятия имеет детей, числится в подразделении и т.д. Термины "содержит", "имеет", "числится" отражают взаимосвязи между понятиями "накладная" и "список товаров", "сотрудник" и "дети", "сотрудник" и "подразделение". Такие взаимосвязи отражаются в реляционных базах данных при помощи внешних ключей, связывающих несколько отношений.

Рассмотрим пример с поставщиками и поставками деталей. Предположим, что нам требуется хранить информацию о наименовании поставщиков, наименовании и количестве поставляемых ими деталей, причем каждый поставщик может поставлять несколько деталей и каждая деталь может поставляться несколькими поставщиками. Можно предложить хранить данные в следующем отношении:

Номер поставщика	**Наименование поставщика**	***Номер детали***	**Наименование детали**	**Поставляемое количество**
1	Иванов	*1*	Болт	100
1	Иванов	*2*	Гайка	200
1	Иванов	*3*	Винт	300
2	Петров	*1*	Болт	150
2	Петров	*2*	Гайка	250
3	Сидоров	*3*	Винт	1000

Таблица 5 Отношение "Поставщики и поставляемые детали"

Потенциальным ключом этого отношения может выступать пара атрибутов {"Номер поставщика", "Номер детали"} - в таблице они выделены курсивом.

Приведенный способ хранения данных обладает рядом недостатков.

Что произойдет, если изменилось наименование поставщика? Т.к. наименование поставщика повторяется во многих кортежах отношения, то это наименование нужно одновременно изменить во всех кортежах, где оно встречается, иначе данные станут противоречивыми. То же самое с

наименованиями деталей. Значит, данные хранятся в нашем отношении с большой избыточностью.

Далее, как отразить факт, что некоторый поставщик, например Петров, временно прекратил поставки деталей? Если мы удалим все кортежи, в которых хранится информация о поставках этого поставщика, то мы потеряем данные о самом Петрове как потенциальном поставщике. Выйти из этого положения, оставив в отношении кортеж типа (2, Петров, NULL, NULL, NULL) мы не можем, т.к. атрибут "Номер детали" входит в состав потенциального ключа и не может содержать null-значений. То же самое произойдет, если некоторая деталь временно не поставляется никаким поставщиком. Получается, что мы не можем хранить информацию о том, что есть некий поставщик, если он не поставляет хотя бы одну деталь, и не можем хранить информацию о том, что есть некоторая деталь, если она никем не поставляется.

Подобные проблемы возникают потому, что мы смешали в одном отношении различные объекты предметной области - и данные о поставщиках, и данные о деталях, и данные о поставках деталей. Говорят, что это отношение плохо нормализовано.

О том, как правильно нормализовать отношения, будет сказано на следующих занятиях, сейчас же предложим разнести данные по трем отношениям - "Поставщики", "Детали", "Поставки" (таблицы 6-8). Для нас важно выяснить, каким образом данные, хранящиеся в этих отношениях взаимосвязаны друг с другом.

Таблица 6 Отношение "Поставщики"

Номер поставщика	**Наименование поставщика**
1	Иванов
2	Петров
3	Сидоров

Таблица 7 Отношение "Детали"

Номер детали	**Наименование детали**
1	Болт
2	Гайка
3	Винт

Таблица 8 Отношение "Поставки"

Номер поставщика	*Номер детали*	Поставляемое количество
1	*1*	100
1	*2*	200
1	*3*	300
2	*1*	150
2	*2*	250
3	*3*	1000

Пример с поставщиками был специально включен в этот раздел, чтобы показать – как важно верно составлять взаимосвязь отношений.

Разберем на другом примере конкретное построение и ссылочность отношений.

Возьмем два отношения (см. рисунок 3.3): одно со сведениями о сотрудниках (номер договора является первичным ключом), другое – со сведениями об отделах, в которых работают сотрудники (номер отдела является первичным ключом).

Сотрудник

Номер договора (PK)	Фамилия	Оклад
10001	Васильчиков	50000.00
10002	Краснин	65000.00
10003	Золотарев	58500.00
10004	Быстроногин	50000.00

Отдел

Номер отдела (PK)	Название отдела
1	Маркетинг
2	Программные технологии

Рисунок 3.3. Пример состояния отношений Сотрудник и Отдел

Для определения типа связи необходимо для каждого из отношений проверить, сколько кортежей второго отношения потенциально может быть связано с одним (любым) кортежем проверяемого. В рассматриваемом примере один сотрудник может быть приписан

максимум к одному отделу, а в одном отделе могут работать несколько сотрудников. Другими словами, не найдётся ни одного сотрудника приписанного более чем к одному отделу, но может существовать такой отдел, в котором работает несколько сотрудников (см. рисунок 3.4). Связь такого типа называется связью один-ко-многим и наиболее часто встречается.

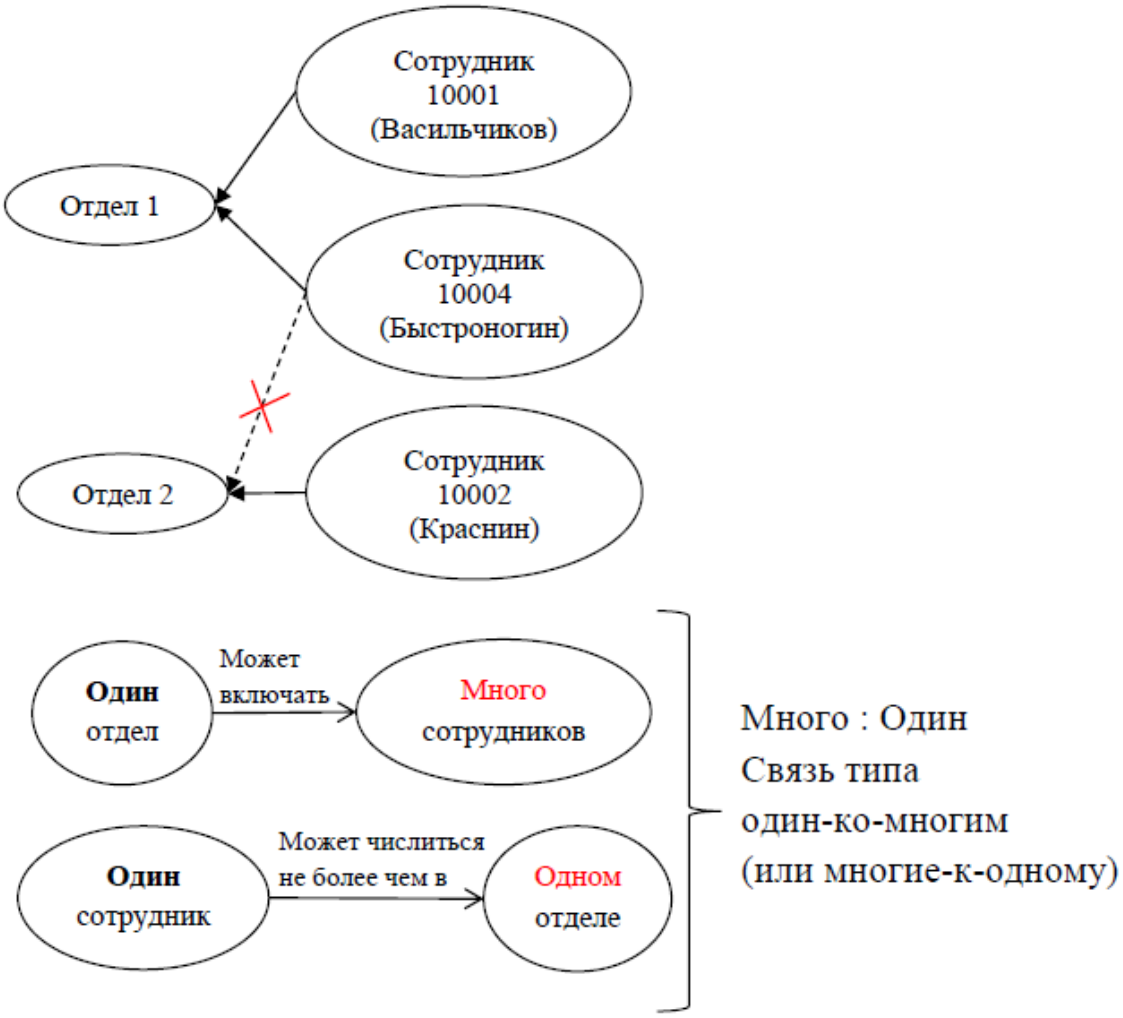

Рисунок 3.4. Связь типа один-ко-многим

Если бы в каждом отделе мог бы работать только один сотрудник, связь называлась бы *один-к-одному*, а если бы сотрудники могли работать в нескольких отделах, связь стала бы *многие-ко-многим* (см. рисунок 3.5).

Связи типа *многие-ко-многим* напрямую в реляционной модели не поддерживаются, т.к. приводят к нарушению свойства уникальности кортежей отношения, что будет проиллюстрировано ниже.

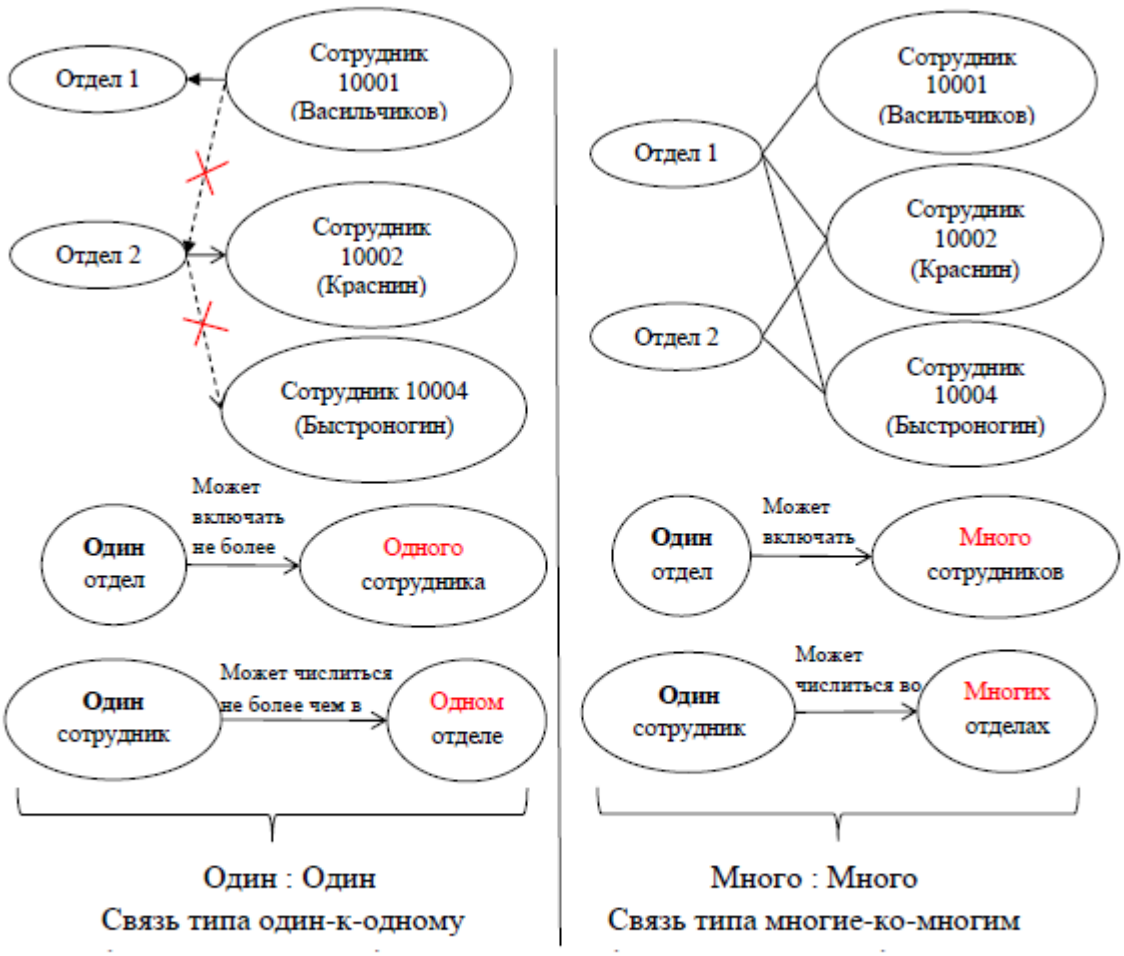

Рисунок 3.5. Связи типа один-к-одному (слева)
и многие-ко-многим (справа)

Вернёмся к исходному примеру: в одном отделе могут работать несколько сотрудников, каждый сотрудник приписан только к одному отделу (или ни к какому). Пусть сотрудники Васильчиков и Быстроногин приписаны к первому отделу, сотрудник Краснин – ко второму, а сотрудник Золотарев работает по договору подряда и в отделах не числится.

Как сохранить информацию о том, какие сотрудники к какому отделу относятся? Очевидно, что нужно в отношение, находящееся в связи со стороны “много” (с информацией о сотрудниках) добавить дополнительный атрибут, содержащий уникальные идентификаторы отделов (см. рисунок 3.6). Уникальными идентификаторами всегда являются значения некоторого потенциального ключа, в данном случае, первичного ключа отношения с информацией об отделах.

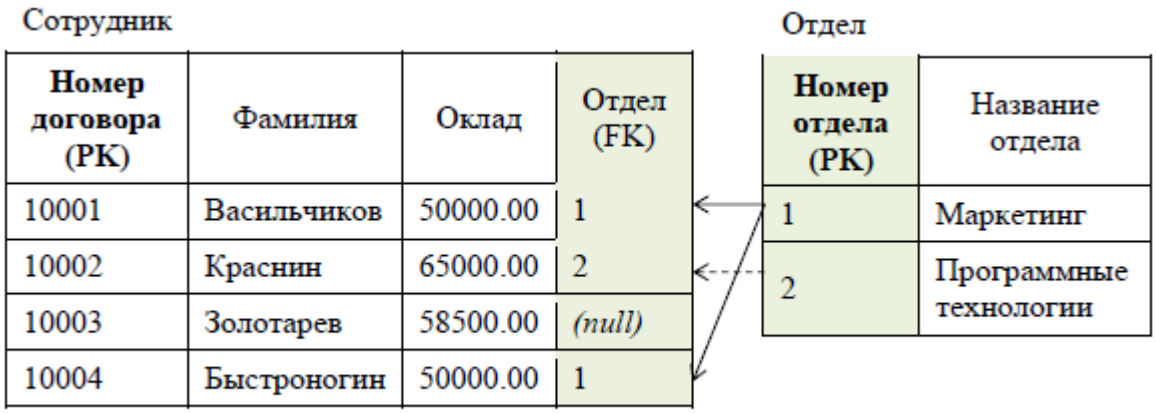

Рисунок 3.6. Добавление атрибута в отношение Сотрудник

Отметим, что противоположная ситуация – добавление атрибута с номерами договоров сотрудников в отношение с информацией об отделах – невозможна. В такой ситуации мы столкнёмся с необходимостью продублировать всю информацию об отделе для каждого из сотрудников, в том числе и номер отдела, являющийся в этом отношении первичным ключом и дублирования значений не допускающий (см. рисунок 3.7).

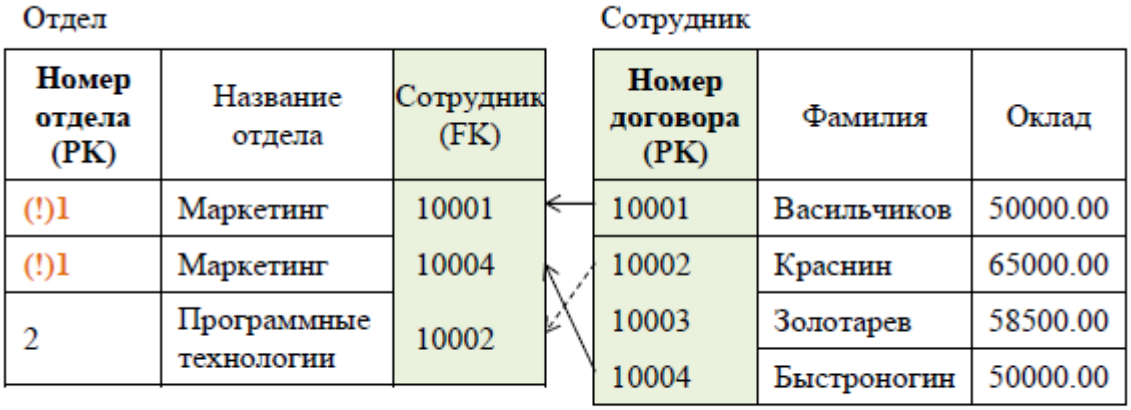

Рисунок 3.7. Добавление атрибута в отношение Отдел

По этой же причине невозможно напрямую реализовать между реляционными отношениями связь многие-ко-многим: дополнительные атрибуты придётся добавлять в оба отношения, и хотя бы в одном из них это приведёт к дублированию значений потенциального ключа.

Для устранения этой проблемы вводят дополнительное отношение, связанное с исходными связями типа один-ко-многим («один» со стороны исходного отношения, «много» – со стороны нового). Добавленное отношение обязательно должно включать атрибуты, содержащие значения потенциальных ключей исходных отношений – совокупность этих атрибутов становится его потенциальным ключом. Также новое отношение может иметь какие-то другие атрибуты, дополнительно

характеризующие связь между исходными отношениями (если вернуться к примеру с поставщиками, то Вы можете заметить эту взаимосвязь).

Рассмотрим более «близкий» к нам пример: реализация связи многие-ко-многим между отношениями Студент и Дисциплина (рисунок 3.8). Связь возникла в результате необходимости хранения сведений об истории сдачи студентами экзаменов: каждый студент может сдать несколько дисциплин, каждая дисциплина может быть сдана несколькими студентами.

В качестве дополнительных характеристик связи в новое отношение были добавлены атрибуты Дата (сдачи экзамена) и Оценка.

Рисунок 3.8. Пример реализации связи многие-ко-многим

Связи в реляционной модели данных поддерживаются с помощью **внешнего ключа**, которому в базе данных соответствует **ограничение типа FOREIGN KEY**. Это ограничение накладывается на вновь добавленный столбец (атрибут) или столбцы (атрибуты) и по сути превращает их известные значения в ссылки.

В примере на рисунке 3.8 благодаря ограничениям внешнего ключа, наложенным на атрибуты Номер зачётной книжки и Индекс дисциплины в отношении Результат экзамена, в базу невозможно будет внести

информацию о студенте, сдавшем несуществующую дисциплину или об экзамене, сданном «левым» студентом.

Итак, подмножество атрибутов FK некоторого отношения R будем называть внешним ключом, если:

- существует отношение S с потенциальным (первичным или альтернативным) ключом K
- каждое значение FK в отношении R всегда совпадает со значением K для некоторого кортежа из S, либо является NULL.

Замечания относительно FK:

- отношение S (всегда находится со стороны “один” в связи) называется родительским отношением, отношение R – дочерним;
- внешний ключ должен состоять ровно из такого же количества атрибутов, как и потенциальный, т.е. тоже может быть простым и составным;
- внешний ключ должен быть определен на тех же доменах, что и соответствующий потенциальный ключ родительского отношения, т.е. множества их допустимых значений должны совпадать;
- внешний ключ, как правило, не обладает свойством уникальности. Так и должно быть, т.к. в дочернем отношении может быть несколько кортежей, ссылающихся на один и тот же кортеж родительского отношения. Это, собственно, и дает тип отношения "один-ко-многим".
- Null для значений атрибутов FK допустимы только в том случае, когда атрибуты FK не входят в состав первичного ключа. Например, в отношении Результат экзамена на рисунке 3.8 ни один из атрибутов внешнего ключа не может содержать NULL (мы обязательно должны знать кто сдал экзамен, и за экзамен по какой дисциплине студент в определённую дату получил оценку).

2.2. ЗАМЕЧАНИЯ К ПРАВИЛАМ ЦЕЛОСТНОСТИ

На самом деле приведенные правила целостности сущностей и внешних ключей прямо следуют из определений понятий "потенциальный ключ" и "внешний ключ".

Действительно, в определении потенциального ключа требуется, чтобы потенциальный ключ обладал свойством уникальности. Это фактически означает, что мы должны уметь различать значения потенциальных ключей, т.е. при сравнении двух значений потенциального ключа мы всегда должны получать значения либо ИСТИНА, либо ЛОЖЬ. Но любое сравнение, в которое входит null-значение, принимает значение U - НЕИЗВЕСТНО, откуда следует, что атрибуты потенциального ключа не могут содержать null-значений.

Тем не менее, явная формулировка правил целостности имеет определенный практический смысл. В большинстве серьезных СУБД за выполнением этих ограничений следит сама СУБД, если, конечно, пользователь явно объявил потенциальные и внешние ключи. Но, во-первых, для некоторых систем можно допустить, чтобы эти ограничения не выполнялись, а во-вторых, некоторые системы просто не поддерживают понятия целостности, например, некоторые "настольные" СУБД типа FoxPro 2.5. В этих случаях за целостностью данных должен следить сам пользователь, или программист, разрабатывающий приложение для пользователя.

Явная формулировка правил целостности помогает четко понять, какие опасности несет в себе пренебрежение этими правилами.

Ссылочная целостность может нарушиться в результате операций, изменяющих состояние базы данных. Таких операций три - вставка, обновление и удаление кортежей в отношениях. Т.к. в определении ссылочной целостности участвуют два отношения - родительское и дочернее, а в каждом из них возможны три операции - вставка, обновление, удаление, то нужно рассмотреть шесть различных вариантов.

2.2.1. Для родительского отношения

- *Вставка кортежа в родительском отношении*. При вставке кортежа в родительское отношение возникает новое значение потенциального ключа. Т.к. допустимо существование кортежей в

родительском отношении, на которые нет ссылок из дочернего отношения, то вставка кортежей в родительское отношение *не нарушает ссылочной целостности*.

- *Обновление кортежа в родительском отношении*. При обновлении кортежа в родительском отношении может измениться значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на обновляемый кортеж, то значения их внешних ключей станут некорректными. Обновление кортежа в родительском отношении *может привести к нарушению ссылочной целостности*, если это обновление затрагивает значение потенциального ключа.
- *Удаление кортежа в родительском отношении*. При удалении кортежа в родительском отношении удаляется значение потенциального ключа. Если есть кортежи в дочернем отношении, ссылающиеся на удаляемый кортеж, то значения их внешних ключей станут некорректными. Удаление кортежей в родительском отношении *может привести к нарушению ссылочной целостности*.

2.2.2. Для дочернего отношения

- *Вставка кортежа в дочернее отношение*. Нельзя вставить кортеж в дочернее отношение, если вставляемое значение внешнего ключа некорректно. Вставка кортежа в дочернее отношение *привести к нарушению ссылочной целостности*.
- *Обновление кортежа в дочернем отношении*. При обновлении кортежа в дочернем отношении можно попытаться некорректно изменить значение внешнего ключа. Обновление кортежа в дочернем отношении *может привести к нарушению ссылочной целостности*.
- *Удаление кортежа в дочернем отношении*. При удалении кортежа в дочернем отношении *ссылочная целостность не нарушается*.

Таким образом, ссылочная целостность в принципе может быть нарушена при выполнении одной из четырех операций:

- Обновление кортежа в родительском отношении.
- Удаление кортежа в родительском отношении.
- Вставка кортежа в дочернее отношение.
- Обновление кортежа в дочернем отношении.

2.2.3. Стратегии поддержания ссылочной целостности

Существуют **две основные стратегии поддержания ссылочной целостности**:

- ***RESTRICT (ОГРАНИЧИТЬ)*** - не разрешать выполнение операции, приводящей к нарушению ссылочной целостности. Это самая простая стратегия, требующая только проверки, имеются ли кортежи в дочернем отношении, связанные с некоторым кортежем в родительском отношении.
- ***CASCADE (КАСКАДИРОВАТЬ)*** - разрешить выполнение требуемой операции, но внести при этом необходимые поправки в других отношениях так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Изменение начинается в родительском отношении и каскадно выполняется в дочернем отношении. В реализации этой стратегии имеется одна тонкость, заключающаяся в том, что дочернее отношение само может быть родительским для некоторого третьего отношения. При этом может дополнительно потребоваться выполнение какой-либо стратегии и для этой связи и т.д. Если при этом какая-либо из каскадных операций (любого уровня) не может быть выполнена, то необходимо отказаться от первоначальной операции и вернуть базу данных в исходное состояние. Это самая сложная стратегия, но она хороша тем, что при этом не нарушается связь между кортежами родительского и дочернего отношений.

Эти стратегии являются стандартными и присутствуют во всех СУБД, в которых имеется поддержка ссылочной целостности.

Можно рассмотреть ***дополнительные стратегии поддержания ссылочной целостности***:

- ***SET NULL (УСТАНОВИТЬ В NULL)*** - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на null-значения. Эта стратегия имеет два недостатка. Во-первых, для нее требуется допустить использование null-значений. Во-вторых, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с

каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.

- ***SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ)*** - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию. Достоинство этой стратегии по сравнению с предыдущей в том, что она позволяет не пользоваться null-значениями. Недостатки заключаются в следующем. Во-первых, в родительском отношении должен быть некий кортеж, потенциальный ключ которого принят как значение по умолчанию для внешних ключей. В качестве такого "кортежа по умолчанию" обычно принимают специальный кортеж, заполненный нулевыми значениями (не null-значениями!). Этот кортеж *нельзя удалять* из родительского отношения, и в этом кортеже *нельзя изменять* значение потенциального ключа. Таким образом, не все кортежи родительского отношения становятся равнозначными, поэтому приходится прилагать дополнительные усилия для отслеживания этой неравнозначности. Это плата за отказ от использования null-значений. Во-вторых, как и в предыдущем случае, кортежи дочернего отношения теряют всякую связь с кортежами родительского отношения. Установить, с каким кортежем родительского отношения были связаны измененные кортежи дочернего отношения, после выполнения операции уже нельзя.

В некоторых реализация СУБД рассматривается ***еще одна стратегия поддержания ссылочной целостности***:

- ***IGNORE (ИГНОРИРОВАТЬ)*** - выполнять операции, не обращая внимания на нарушения ссылочной целостности.

Конечно, это не стратегия, а отказ от поддержки ссылочной целостности. В этом случае в дочернем отношении могут появляться некорректные значения внешних ключей, и вся ответственность за целостность базы данных ложится на пользователя.

Выводы

Современные СУБД допускают использование ***null-значений***, т.к. данные часто бывают неполными или неизвестными. Споры о допустимости использования null-значений ведутся до сих пор. Использование null-значения связано с применением ***трехзначной логики*** (***three-valued logic, 3VL***).

Средством, позволяющим однозначно идентифицировать кортежи отношения, являются ***потенциальные ключи отношения***.

Потенциальный ключ отношения - это набор атрибутов отношения, обладающий свойствами ***уникальности*** и ***неизбыточности***. Доступ к конкретному кортежу можно получить, лишь зная значение потенциального ключа для этого кортежа.

Традиционно один из потенциальных ключей объявляется ***первичным ключом***, остальные - ***альтернативными ключами***.

Потенциальный ключ, состоящий из одного атрибута, называется ***простым***. Потенциальный ключ, состоящий из нескольких атрибутов, называется ***составным***.

Отношения связываются друг с другом при помощи ***внешних ключей***.

Внешний ключ отношения - это набор атрибутов отношения, содержащий ссылки на потенциальный ключ другого (или того же самого) отношения. Отношение, содержащее потенциальный ключ, на который ссылается некоторый внешний ключ, называется ***родительским отношением***. Отношение, содержащее внешний ключ, называется ***дочерним отношением***.

Внешний ключ не обязан обладать свойством уникальности. Поэтому, одному кортежу родительского отношения может соответствовать несколько кортежей дочернего отношения. Такой тип связи между отношениями называется "***один-ко-многим***".

Связи типа "***много-ко-многим***" реализуются использованием нескольких отношений типа "один-ко-многим".

В любой реляционной базе данных должны выполняться два ограничения:

- Целостность сущностей
- Целостность внешних ключей

Правило ***целостности сущностей*** *состоит в том, что а*трибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

Правило ***целостности внешних ключей*** состоит в том, что внешние ключи не должны ссылаться на отсутствующие в родительском отношении кортежи, т.е. внешние ключи должны быть корректны.

Ссылочную целостность могут нарушить операции, изменяющие состояние базы данных. Такими операциями являются операции вставки, обновления и удаления кортежей.