Министерство образования и науки РФ
Севастопольский государственный университет

Проектирование реляционных баз данных. Нормализация отношений

Методические указания
к выполнению лабораторной работы №3
по дисциплине **«Управление данными»**

Для студентов, обучающихся по направлению 09.03.02
«Информационные системы и технологии»
по учебному плану подготовки бакалавров
дневной и заочной форм обучения

Севастополь
2018

УДК 004.65 (075.8)

Проектирование реляционных баз данных. Нормализация отношений. Методические указания к лабораторным занятиям по дисциплине «Управление днными» / Сост. Ю.В. Доронина, А.В. Волкова – Севастополь: Изд-во СевГУ, 2018 – 24 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине «Управление данными». Целью методических указаний является помощь студентам в изучении процесса построения логической модели и модели данных, основанной на ключах. Излагаются теоретические и практические сведения необходимые для выполнения лабораторной работы, требования к содержанию отчета.

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры информационных систем

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

1 ЦЕЛЬ РАБОТЫ

Осуществление исследования и анализа предметной области, построение диаграммы «сущность-связь» и модели данных, основанной на ключах.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основная цель процесса проектирования БД состоит в получении такого проекта, который удовлетворяет следующим требованиям:

1) корректность схемы БД, т.е. база должна быть гомоморфным образом моделируемой предметной области (ПрО), где каждому объекту предметной области соответствуют данные в памяти ЭВМ, а каждому процессу – адекватные процедуры обработки данных;
2) обеспечение ограничений (на объёмы внешней и оперативной памяти и другие ресурсы вычислительной системы);
3) эффективность функционирования (соблюдение ограничений на время реакции системы на запрос и обновление данных);
4) защита данных (от аппаратных и программных сбоев и несанкционированного доступа);
5) простота и удобство эксплуатации;
6) гибкость, т.е. возможность развития и адаптации к изменениям предметной области и/или требований пользователей.

Удовлетворение требований 1-4 обязательно для принятия проекта.

Максимально упростить и формализовать процессы формирования требований и проектирования системы позволяют современные CASE-средства.

2.1 Основы проектирования баз данных

В настоящее время при разработке информационных систем используются специальные программные средства – CASE-средства, реализующие CASE-технологию создания и сопровождения информационных систем. Значение термина CASE охватывает процесс разработки и сопровождения сложных систем в целом.

CASE-технология представляет собой методологию проектирования информационных систем, набор методов, нотаций (установленные способы отображения элементов системы, т.е. графы, таблицы, блок-схемы, формальные и естественные языки) и инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать модель системы на всех этапах разработки и сопровождения системы и разрабатывать приложения в соответствии с информационными потребностями пользователей.

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели. В общем случае процедуру проектирования базы данных разбивают на три этапа, каждый из которых завершается созданием соответствующей информационной модели.

1. Концептуальное проектирование – создание представления (схемы, модели) БД, включающего определение важнейших сущностей (таблиц) и связей между ними, но не зависящего от модели БД (иерархической, сетевой, реляционной и т.д.) и физической реализации (целевой СУБД).

2. Логическое проектирование – развитие концептуального представления БД с учетом принимаемой модели (иерархической, сетевой, реляционной и т.д.).

3. Физическое проектирование – развитие логической модели БД с учетом выбранной целевой СУБД**.**

Концептуальное и логическое проектирование вместе называют также *инфологическим* или *семантическим проектированием*.

В настоящее время для проектирования БД активно используются CASE-средства, в основном ориентированные на использование ERD (Entity – Relationship Diagrams, диаграммы «сущность-связь»), являющейся информационной моделью. С их помощью определяются важные для предметной области объекты (сущности), отношения друг с другом (связи) и их свойства (атрибуты). Средства проектирования ERD в основном ориентированы на реляционные базы данных (РБД), и если существует необходимость проектирования другой системы, например, объектно-ориентированной, то лучше избрать другие методы проектирования.

ERD были впервые предложены П. Ченом в 1976 г. Этапы и правила построения диаграммы «сущность-связь» в нотации П.Чена рассмотрены в п. 3.2.1 методических указаний по выполнению курсовой работы по дисциплине «Управление данными».

При использовании любого CASE-средства вначале строится логическая модель БД в виде диаграммы с указанием сущностей и связей между ними, а тем физическая модель.

Логической моделью данных называется универсальное представление структуры данных, независимое от конечной реализации базы данных и аппаратной платформы. Логическая модель позволяет понять суть проектируемой системы, отражая логические взаимосвязи между сущностями. На основании полученной логической модели переходят к физической модели данных.

Физическая модель представляет собой диаграмму, содержащую всю необходимую информацию для генерации БД для конкретной СУБД или даже конкретной версии СУБД и отражает физические свойства проектируемой БД (типы данных, размер полей, индексы). Параметры физической информационной модели зависят от выбранной СУБД. Если в логической модели не имеет значения, какие идентификаторы носят таблицы и атрибуты, тип данных атрибутов и т.д., то в физической модели должно быть полное описание БД в соответствии с принятым в ней синтаксисом, с указанием типов атрибутов, триггеров, хранимых процедур и т.д. По одной и той же логической модели можно создать несколько физических.

Такой порядок действий называется *прямое проектирование БД (Forward Engineering DB)*. CASE-средства позволяют выполнять также *обратное проектирование БД (Reverse Engineering DB)*, т.е. на основании системно-

го каталога БД или DDL-скрипта (Data Definition Language – язык описания данных) построить физическую и, далее, логическую модель данных.

2.2 Типы структур баз данных

Наиболее распространены три типа структур и моделей данных: иерархическая или древовидная; сетевая; реляционная (плоские файлы).

Если сетевая структура имеет связи типа M:M, то ее называют ***сложной сетевой структурой (ССС)*** в противном случае – ***простой сетевой структурой (ПСС)***.

2.2.1 Преобразование сложной сетевой структуры в простую сетевую структуру

Сложную сетевую структуру можно преобразовать в простую сетевую, введя дополнительный тип записи. Этот тип записи должен содержать ключевые атрибуты объектов, участвующих в связях (см. рисунок 2.1).

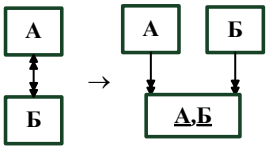


Рисунок 2.1 – Схема преобразования ССС в ПСС

2.2.2 Преобразование простой сетевой структуры в древовидную структуру

Любую простую сетевую структуру можно преобразовать в древовидную, введя избыточность посредством дублирования отношения, входящего в ПСС-связь (1:M) со стороны M. На рисунке 2.2 приведены примеры таких преобразований.

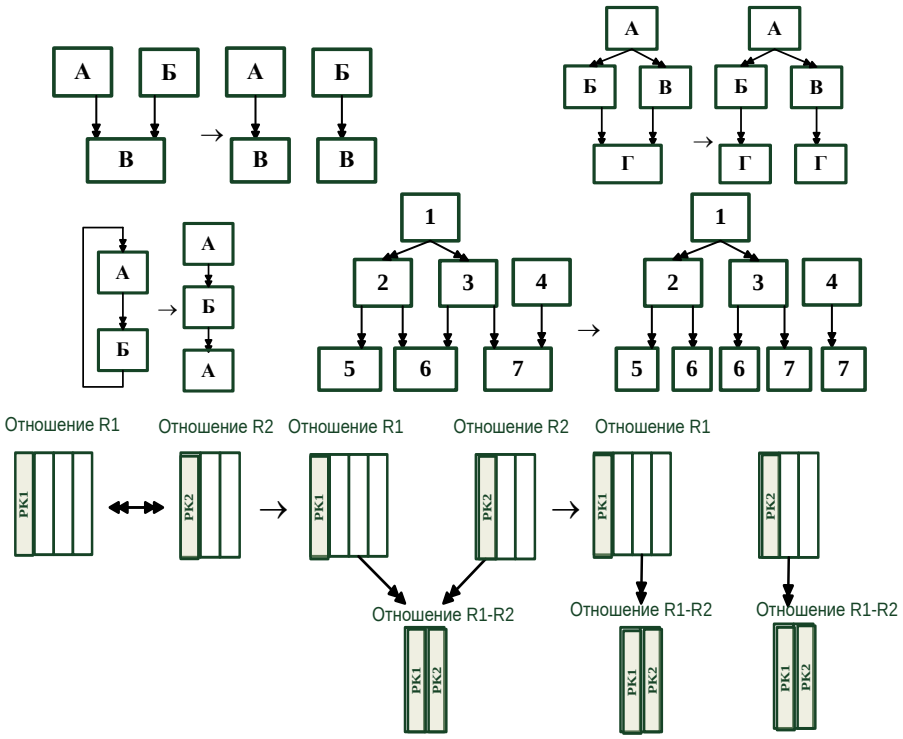


Рисунок 2.2 – Примеры преобразования ПСС в древовидные структуры

После составления концептуальной (логической) схемы БД необходимо проверить её на отсутствие аномалий модификации данных. При неправильно спроектированной схеме БД могут возникнуть аномалии выполнения операций модификации данных. Эти аномалии обусловлены ограниченностью структуры реляционной модели данных.

Аномалия – такая ситуация в таблице БД, которая приводит к противоречию в БД, либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Различают три вида аномалий: аномалии обновления, удаления и добавления. Аномалия обновления может возникнуть в том случае, когда информация дублируется. Другие аномалии возникают тогда, когда две и более сущности объединены в одно отношение.

Виды аномалий модификации данных

– аномалии вставки (INSERT) – возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы (например, нельзя добавить информацию о поставщике, который не поставил ни одного товара; или нельзя добавить информацию о преподавателе, если данный предмет не выбрал ни один студент и т.п.);

– аномалии обновления (UPDATE) – проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы (например, эта проблема возникнет в том случае, если необходимо переместить поставщика из одного города в другой; или чтобы изменить товар «сода» на «сода пищевая», нужно просмотреть всю таблицу);

– аномалии удаления (DELETE) – при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью (например, с удалением поставки исчезнет информация о поставщике).

Для решения проблемы аномалии модификации данных при проектировании реляционной БД проводится нормализация отношений.

2.2.3 Табличные структуры данных (реляционная модель)

Понятие РМД (реляционной модели БД) основывается на:

- целостности отношений – связей по внешним ключам;
- нормализации отношений – процессе оптимизации схемы с целью повышения эффективности манипулирования данными.

Процесс приведения древовидной структуры (ДС) к табличной форме называется ***нормализацией***. Это понятие ввел Кодд.

Задача преобразования древовидной структуры к табличной форме состоит в ведении избыточности. Существуют 7НФ. Приведение к каждой из последующих нормальных форм связано с устранением избыточности посредством анализа зависимостей отношения.

Примечание: *введение избыточности в процессе формирования РБД из ДС, а затем ее устранение в процессе нормализации не может быть описано операциями сложения и вычитания. В данном случае используются реляционные операции соединения и проекции, а это иные операции.*

2.3 Нормализация отношений

Под ***нормализацией*** отношения подразумевается процесс приведения отношения к одной из так называемых ***нормальных форм*** (или в дальнейшем НФ). Но сначала определим зачем же нужна нормализация.

База данных – это не просто хранилище фактов (с этой задачей способны справиться и незатейливые плоские файлы). При проектировании баз данных упор в первую очередь делается на достоверность и непротиворечивость хранимых данных, причем эти свойства не должны утрачиваться в процессе работы с данными, т.е. после многочисленных изменений, удалений и дополнений данных по отношению к первоначальному состоянию БД.

Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщенное название *средств поддержки целостности*. Эти механизмы применяются как статически (на этапе проектирования БД), так и динамически (в процессе работы с БД). Рассмотрим ограничения, которым должна удовлетворять БД в процессе создания, независимо от ее наполнения данными. Приведение структуры БД в соответствие этим ограничениям – это и есть нормализация.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести (и на практике непременно приводит, как только проект приобретает реальную сложность) к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации.

Впрочем, для устранения всяческих неопределенностей и неоднозначностей традиционно используется формальный язык математики. Итак, наша цель – приведение отношений к НФ. Следует заметить, что в процессе нормализации постоянно встречается ситуация, когда отношение приходится разложить на несколько других отношений. Поэтому более корректно было бы говорить о нормализации не отдельных отношений, а всей их совокупности в БД. В примерах для простоты рассмотрены отдельные отношения, нормализация же реальных баз данных – гораздо более трудоемкий процесс.

Цель нормализации – получение такого проекта базы данных, в котором каждый факт появляется лишь в одном месте (исключена избыточность информации).

Нормализация может не представлять такую уж проблему, если БД проектируется сразу по определенным канонам. Другими словами, можно сначала сделать БД как попало, а потом нормализовать ее, или же с самого начала строить ее по правилам, чтобы в дальнейшем не пришлось переделывать. Сейчас мы пойдем первым путем, т.е. возьмем в качестве примеров ни-

куда не годные БД и попытаемся привести их в порядок. При этом будем иметь в виду, что при профессиональном подходе к проектированию БД используется одна из хорошо проработанных технологий (например, IDEF1X), и, возможно, какие-либо инструментальные средства, поддерживающие эту технологию.

Что же представляет собой процесс нормализации? Фактически это не что иное, как последовательное преобразование исходной БД к НФ, при этом каждая следующая НФ обязательно включает в себя предыдущую, т.е. при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются (что, собственно, и позволяет разбить процесс на этапы и производить его однократно, не возвращаясь к предыдущим этапам). Всего в реляционной теории насчитывается 7 НФ: 1-я НФ (обычно обозначается также 1НФ), 2НФ, 3НФ, 4НФ, Бойса-Кодда (НФБК), 4НФ, 5НФ.

На практике, как правило, ограничиваются 3НФ, ее оказывается вполне достаточно для создания надежной схемы БД. НФ более высокого порядка представляют скорее академический интерес из-за чрезмерной сложности. Более того, при реализации абстрактной схемы БД в виде реальной базы иногда разработчики вынуждены сделать шаг назад – провести денормализацию с целью повышения эффективности, ибо идеальная с точки зрения теории структура может оказаться слишком накладной на практике.

2.3.1 Первая нормальная форма (1НФ)

Схема отношения **R** находится в 1НФ, если значения **dom(A)** являются атомарными для каждого атрибута **A** в **R**.

Другими словами, *каждый атрибут отношения должен хранить одно-единственное значение и не являться ни списком, ни множеством значений*.

Тем не менее, несмотря на внешнюю строгость данного определения, однозначно определить понятие атомарности зачастую оказывается довольно затруднительно, если заранее неизвестны семантика атрибута и его роль в обработке хранимых данных. Атрибут, который является атомарным в одном приложении, может оказаться составным в другом.

Простейший пример: в БД отдела кадров предприятия в таблице, хранящей личные сведения о сотрудниках, имеется атрибут «домашний-адрес», в котором адрес хранится в формате: город, улица, дом[/корпус], [квартира] (следуя общепринятой нотации, здесь в квадратных скобках указаны опциональные фрагменты адреса, которые могут отсутствовать). В данном случае адрес хранится в виде единой текстовой строки, поскольку маловероятно, чтобы потребовалось выбрать сотрудников, скажем, по номеру квартиры. Таким образом, в контексте БД отдела кадров адрес является атомарным понятием, и его деление на составные части не имеет смысла, т.к. только внесет в БД излишнюю громоздкость.

Однако тот же адрес для приложения, предназначенного для сортировки почты в почтовом отделении, атомарным не является, поскольку желательно сгруппировать конверты в отдельные стопки по улицам, так как каждую улицу обслуживает свой почтальон. Кроме того, с целью оптимизации

перемещений почтальона в пределах улицы, каждую стопку желательно отсортировать по номерам домов, чтобы сделать возможным разнести почту за один проход по улице без возврата.

Итак, очевидно, что в отрыве от контекста затруднительно дать строгое определение атомарности, за исключением самых простых случаев (например, домен натуральных чисел). В большинстве случаев необходимо располагать сведениями о предметной области, а также о том, каким образом планируется обрабатывать хранимые в БД данные.

Приведение отношения к 1НФ – довольно простая операция. Необходимо просмотреть схему отношения и разделить составные атрибуты на различные строки/столбцы. Возможно, эту операцию придется повторить несколько раз до тех пор, пока каждый из атрибутов не станет атомарным.

Чтобы отношение соответствовало 1НФ необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы (одинаковые кортежи);

3) привести поля отношения к атомарности.

Основой этого этапа нормализации является определение ключей.

На рисунке 2.3 приведена схема определения ключевых атрибутов в древовидной форме, которая называется *правилом распространения ключей.*

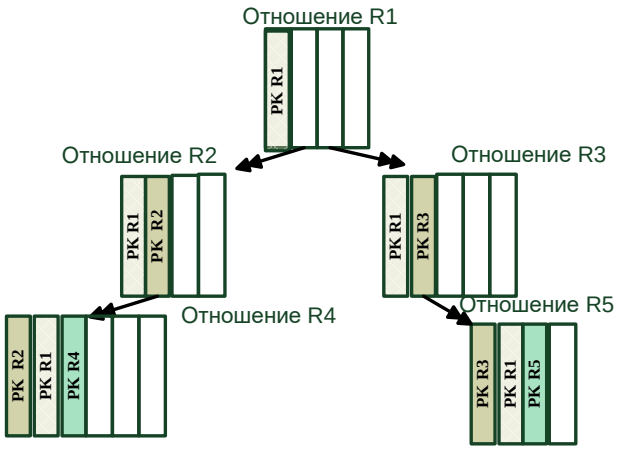


Рисунок 2.3 – Схема распространения ключей в древовидной структуре: PK Ri –первичный ключ отношения i

2.3.2 Вторая нормальная форма (2НФ)

Перед тем, как дать определение 2НФ, необходимо познакомиться с понятием *полной* и *частичной* функциональной зависимостей (ФЗ).

Пусть **F** – множество функциональных зависимостей, при этом ФЗ **X→Y** **F+**. Множество **Y** называется *частично зависимым* от **X** относительно **F**, если ФЗ **X→Y** не является редуцированной слева (т.е. существует собственное подмножество **X** множества **X**, такое, что ФЗ **X'→Y** **F+**. Если же ФЗ **X→Y** является редуцированной слева, то множество **Y** называется *полностью зависимым* от **X** относительно **F.** Множество атрибутов **X** называется *детерминантом функциональной зависимости*, а множество атрибутов **Y** называется *зависимой частью*.

Атрибут B отношения ***функционально*** *зависит от атрибута A того же отношения (атрибуты могут быть составными) в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений атрибута A обязательно существует только одно из различных значений атрибута B.*

Атрибут B находится в ***полной функциональной зависимости*** *от составного атрибута A, если он функционально зависит от A и не зависит функционально от любого подмножества атрибута A.*

Для изображения ФЗ используется графическое изображение ФЗ, так называемая *диаграмма ФЗ (или схема ФЗ)*, например, см. рисунок 2.4.

Рисунок 2.4 – Диаграмма функциональной зависимости (или схема ФЗ)

Обозначение функциональной зависимости: $f : A \rightarrow B \Rightarrow A$ определяет B.

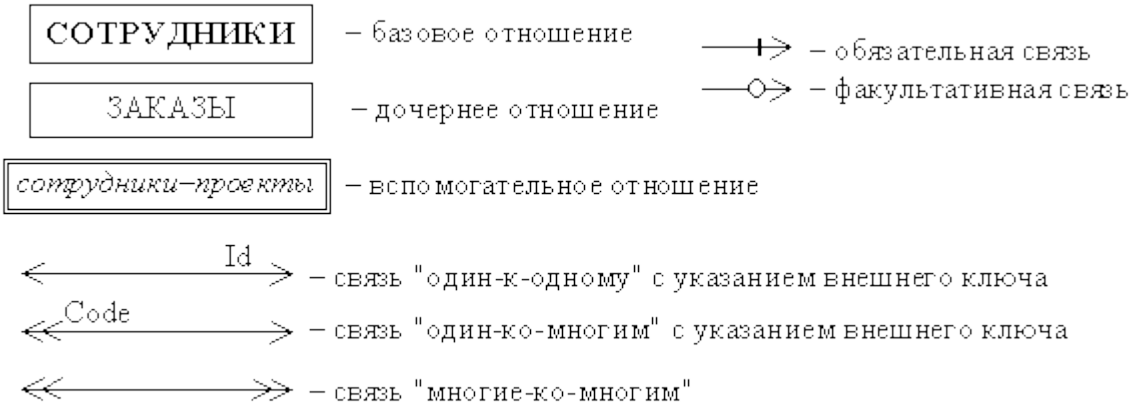


Схема отношения **R** находится во 2НФ относительно множества функциональных зависимостей **F**, если она находится в 1НФ и каждый неключевой атрибут полностью зависит от каждого ключа для **R.**

Другими словами, *отношение находится во 2НФ, если оно находится в 1НФ, и каждый неключевой атрибут характеризуется полной функциональной зависимостью от первичного ключа (т.е. все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части).*

Например, таблица «Оценки по экзаменам» характеризуется следующим набором атрибутов: Номер зачетной книжки, Дисциплина, Дата сдачи, ФИО студента, № группы, Оценка. Очевидно, что первичным ключом является набор: Номер зачетной книжки, Дисциплина, Дата сдачи. Полной функциональной зависимостью обладает только один неключевой атрибут «Оцен-

ка». Атрибуты «ФИО студента» и «№ группы» могут быть однозначно определены по части первичного ключа – «Номер зачетной книжки». Таким образом, требуется разбиение исходной таблицы на две (см. рисунок 2.5).

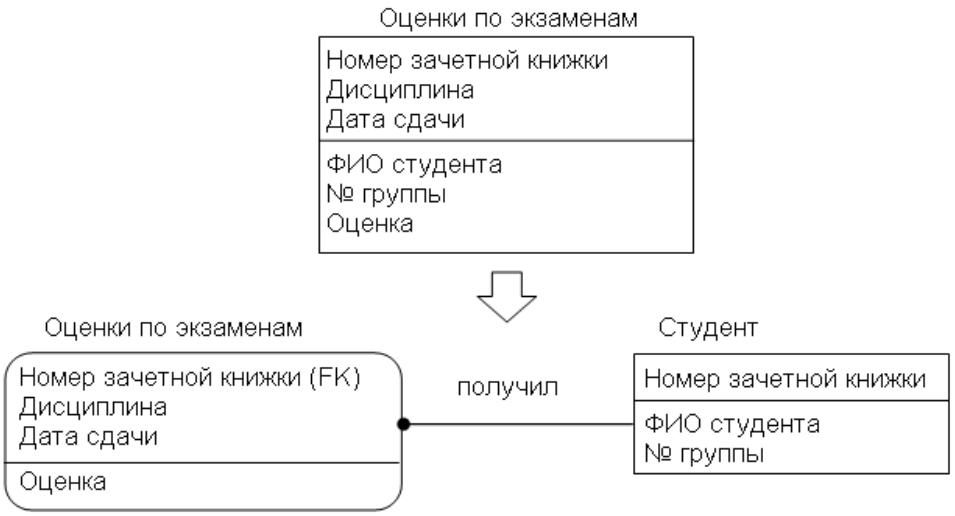


Рисунок 2.5 – Обеспечение полной функциональной зависимости

2.3.3 Третья нормальная форма (3НФ)

Для того, чтобы формально определить 3НФ, необходимо предварительно познакомиться с понятием *транзитивной зависимости* атрибутов, от которой нужно будет попытаться избавиться на этом этапе.

Обозначим: **R** – схема отношения, **X** – подмножество **R**, **A** – атрибут в **R**, **F** – множество функциональных зависимостей. **A** называется *транзитивно зависимым* от **X** в **R**, если существует такое **Y**, являющееся подмножеством **R**, что:

- $X \rightarrow Y$
- $\sim(Y \rightarrow X)$
- $Y \rightarrow A$
- $\sim(A \ XY)$

Атрибут B отношения ***транзитивно функционально*** *зависит от атрибута A того же отношения (атрибуты могут быть составными) в том и только в том случае, если существует такой атрибут C, что имеются функциональные зависимости между атрибутами A и C, а также между B и C.*

Схема отношения **R** находится в 3НФ относительно множества функциональных зависимостей **F**, если она находится во 2НФ и ни один из неключевых атрибутов в **R** не является транзитивно зависимым от ключа для **R**.

Другим словами, *отношение находится в 3НФ, если оно находится во 2НФ и никакой неключевой атрибут функционально не зависит от другого неключевого атрибута, т.е. нет транзитивных зависимостей.*

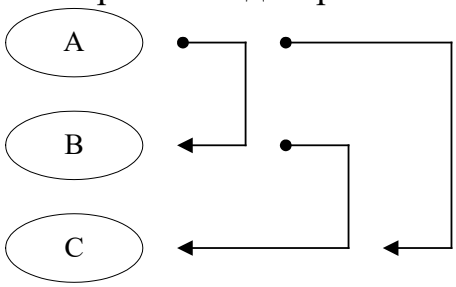
В реляционной теории имеется лемма, которая гласит, что любая схема отношения, находящаяся в 3НФ относительно **F**, находится в 2НФ относительно **F**.

Теорема Хита дает возможность выполнить декомпозицию отношений без потерь информации:

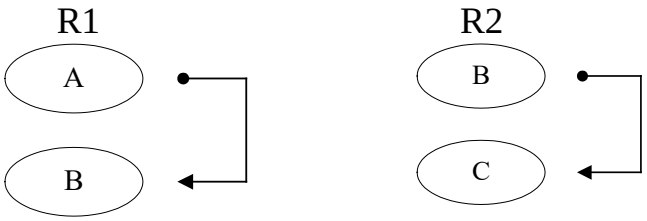
– исходное отношение с ФЗ преобразуется в другие отношения, в каждом из которых атрибуты минимально зависят от первичного ключа;

– атрибут С минимально зависит от атрибута А, если выполняется минимальная слева ФЗ В→С.

Пусть А, В, С – три атрибута или три набора атрибутов отношения R. Зависимости между ними изобразим на диаграмме.



Если А → В и В → С, но В !→ А (В не является ключом), то А → С. В этом случае С транзитивно зависит от А. Преобразование в 3НФ состоит в декомпозиции исходного отношения на два отношения.



Например, таблица «Работник» характеризуется набором атрибутов: Табельный номер, Фамилия, Имя, Отчество, Должность, Зарплата, …, первичный ключ – Табельный номер. В этой таблице от первичного ключа («Табельный номер») зависит неключевой атрибут «Должность», а от «Должности» другой неключевой атрибут «Зарплата». Для приведения к 3NF необходимо добавить новую таблицу (см. рисунок 2.6).

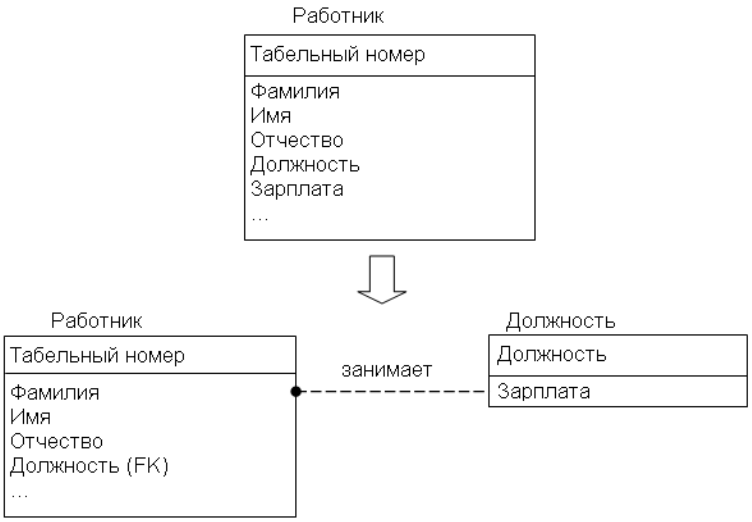


Рисунок 2.6 – Устранение транзитивной зависимости

Таким образом, чтобы привести отношение к 3НФ, необходимо устранить функциональные зависимости между неключевыми атрибутами отношения. То есть, факты, хранимые в таблице, должны зависеть только от ключа.

2.3.4 Нормальная форма Бойса-Кодда (НФБК)

Нормальная форма Бойса-Кодда считается уточнением 3НФ. Она учитывает все потенциальные ключи, которые входят в отношения. Если отношение имеет единственный потенциальный ключ, то 3НФ и НФБК – эквивалентны. Считается, что отношение, находящееся в НФБК, если каждый его детерминант является потенциальным ключом. Чтобы убедиться, что отношение находится в НФБК необходимо отыскать все его детерминанты и убедиться, что они являются потенциальными ключами.

Можно считать БКНФ как особый случай 3НФ. На самом деле большинство таблиц соответствуют требованиям БКНФ, если они соответствуют требованиям 3НФ. Однако рассмотрим случай, если неключевой атрибут является детерминантом ключевого атрибута. Такое обстоятельство не нарушает условий 3НФ, но не отвечает правилам БКНФ, поскольку БКНФ требует, чтобы каждый детерминант в таблице был потенциальным ключом.

Описанную выше ситуацию, (таблица, приведенная к 3НФ, не соответствует требованиям БКНФ), проще объяснить с помощью рисунка 2.7, на котором прослеживаются следующие функциональные зависимости:

$$A + B \rightarrow C, D$$

$$C \rightarrow B$$

В структуре сущности, представленной на рисунке 2.7, нет ни частичных зависимостей, ни транзитивных зависимостей (условие C → B указывает на то, что неключевой атрибут определяет часть первичного ключа – а эта зависимость не транзитивна!). Очевидно, что структура сущности, представленная на рисунке 2.7 отвечает требованиям 3НФ. А вот условие C → B нарушает требования БКНФ.

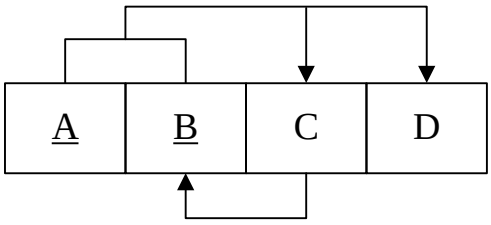


Рисунок 2.7 – Структура сущности, приведенной к 3НФ, но не к БКНФ

Чтобы преобразовать структуру, представленную на рисунке 2.7, в сущность, отвечающую требованиям как 3НФ, так и БКНФ, прежде всего, необходимо изменить первичный ключ на A + C. Это целесообразно сделать, поскольку зависимость C → B означает, что C, по сути дела, является подмножеством B. После этого можно считать, что сущность приведена к 2НФ, поскольку в ней имеется частичная зависимость C → B. Затем, с помощью знакомых процедур декомпозиции, приведем сущность к виду, представленному на рисунке 2.8.

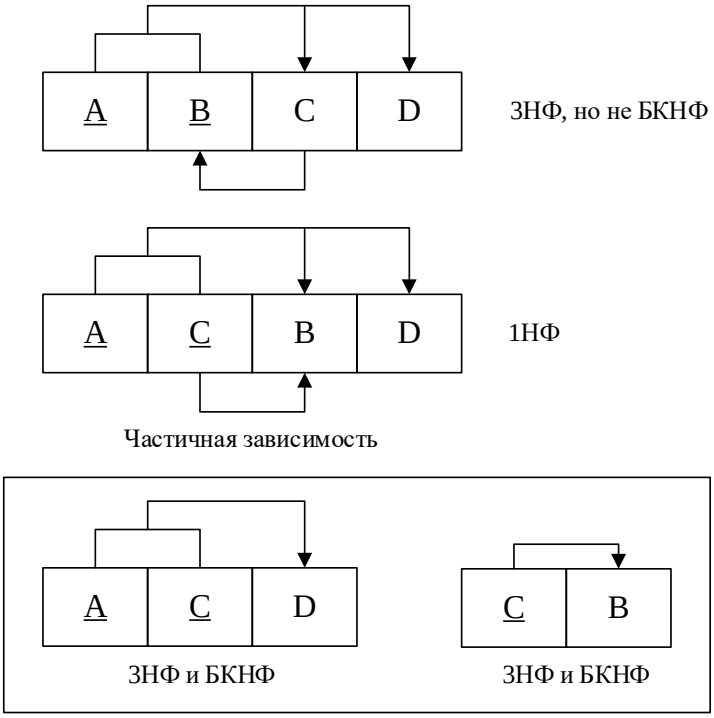


Рисунок 2.8 – Декомпозиция структуры таблицы для выполнения требований БКНФ

Примечание: для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.

2.3.5 Четвертая нормальная форма (4НФ)

В ходе проектирования БД выявлен один тип зависимости – ***многозначная зависимость***. Многозначные зависимости выявляет проблемы и избыточностью данных.

Отделение – Сотрудник – Клиент

N_отдела	ФИО_сотрудника	ФИО_клиента
011	Кот	Чижик
012	Крот	Лебедев
011	Кот	Гусев
012	Крот	Тупик

В данном отношении имеются многозначные зависимости типа «один ко многим» (1:N).

N_отдела – ФИО_клиента (1:N)

N_отдела – ФИО_сотрудника (1:N)

Если для каждого атрибута A имеется набор атрибутов B и C. Хотя атрибут B и C не зависят друг от друга.

Многозначная зависимость A → B и A → C.

Многозначная зависимость подразделяется на *тривиальную* и *нетривиальную* зависимости. Многозначная зависимость A и B, определенных на некотором отношении R, называется ***тривиальной***, если атрибут B является

подмножеством атрибут А. В противном случае тривиальная зависимость является не тривиальной.

Отношение находится в 4НФ, если оно удовлетворяет НФБК и не содержит многозначных нетривиальных зависимостей.

R1: Отделение – Сотрудник (N_отделения, ФИО_сотрудника)
R2: Отделение – Клиент (N_отделения, ФИО_клиента)

Бывают случаи, когда необходимо выполнять декомпозицию на более чем два отношения. В этом случае необходимо учитывать зависимость соединения. Зависимость соединения – это свойство декомпозиции, которая вызывает генерацию ложных строк при обратном соединении декомпозированных отношений. Чтобы не возникало зависимостей соединения, необходимо отношение приводить к 5НФ.

Примечание: приведение базы данных к 4НФ сокращает дублирование данных, но появление новых отношений усложняет схему базы данных.

2.3.6 Пятая нормальная форма (5НФ)

Отношение в 5НФ – это отношение без зависимостей соединения.
Например:

Объект – Мебель – Поставщик

N_объекта	Мебель	N_поставщика
31	Стол	P1
31	Стул	P2
52	Стул	P3
52	Кровать	P1

Для того, чтобы отношение удовлетворяло 5-ой НФ, необходимо его разбить на следующие отношения:

Объект – Мебель (N_объекта, Мебель)
Поставщик – Мебель (N_поставщика, Мебель)
Объекта - Поставщик (N_объекта, N_поставщика)

2.3.7 Этапы нормализации базы данных

Шаг 1 (приведение к 1НФ). Задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (приведение к 2НФ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений согласно процедуре приведения ко 2НФ.

Шаг 3 (приведение к 3НФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов других неключевых атрибутов, то проводим декомпозицию этих отношений согласно процедуре приведения к 3НФ.

2.4 Денормализация отношений

Создание нормализованных отношений является важнейшей частью проектирования БД. В хорошем проекте БД должны учитываться различные требования к обработке информации. По мере того как сущности преобразуются в соответствии с требованиями нормализации, их число растет. Большее число сущностей влечет за собой увеличение количества операций обмена данными с диском (операции ввода/вывода) и увеличение времени на обработку логических связей, что приводит к снижению скорости всей системы в целом. Поэтому противоречие между эффективностью проекта, информационными потребностями и скоростью обработки информации чаще всего разрешается поиском компромисса, который, как правило, состоит в некоторой денормализации. Денормализация приводит к понижению уровня формы (т.е. в процессе денормализации 3НФ конвертируется в 2НФ). Однако необходимо помнить, что повышение производительности посредством денормализации приведет к повышению уровня избыточности данных.

Нормальные формы низких уровней имеют место в специализированных БД, называемых информационными хранилищами (data warehouses). В таких специализированных БД находят отражение все возрастающие потребности широты охвата и глубины поиска информации, на которых основаны системы поддержки решений. В информационных хранилищах, как правило, используются структуры 2НФ в условиях сложной, многоуровневой информационной среды, питающейся от множества источников.

Это не означает, что сущности рабочих баз данных, в которых содержатся частичные или транзитивные зависимости, можно просто оставить в таком виде. Ненормализованные сущности в рабочей БД, кроме возможных аномалий данных, имеют и другие недостатки:

– невысокая эффективность обновления информации, поскольку программы, которые считывают и обновляют данные, работают с сущностями больших размеров;

– затруднен процесс индексирования. Просто непрактично формировать все индексы для множества атрибутов, расположенных в единственной ненормализованной сущности;

– ненормализованные сущности затрудняют стратегии создания представлений.

Таким образом, увеличивая путём денормализации скорость обработки данных, снижается эффективность обновления сущностей (они становятся больше), индексирование усложняется и появляется угроза избыточности данных, которая может стать причиной аномалий данных. Вывод – денормализацию следует применять с осторожностью.

Обоснованием денормализации может служить требование обеспечения определённой производительности для критических запросов.

Денормализация бывает нескольких видов:

– восходящая – подразумевает перенос некоторой информации из подчинённого отношения в родительское;

– нисходящая – в этом случае информация переносится из родительского отношения в подчинённое.

Обычно денормализация применяется в случае, когда запись имеет большую длину за счёт наличия атрибутов большого объёма (графические данные, текстовые описания и проч.). Если данные этих атрибутов редко используются, то можно выделить в отдельное отношение атрибуты большого объёма. Для связи с исходным отношением вводится уникальный внешний ключ. А для получения исходного отношения создаётся представление (view), которое является соединением двух полученных отношений.

3 ПРИМЕР НОРМАЛИЗАЦИИ ОТНОШЕНИЙ

В качестве примера реализации процесса нормализации, рассмотрим простейшее бизнес-приложение. В данном примере будет рассматриваться работа с упрощенной базой данных компании по разработке ПО. У каждого проекта имеются свой уникальный номер, название, штат сотрудников и т.д. У каждого сотрудника есть идентификационный номер и специальность, например, инженер-программист или постановщик задач.

Для учёта трудозатрат на разработку ПО компания периодически составляет отчеты, представляющие собой расходы на заработную плату сотрудников по каждому из проектов. Для этого учитываются человеко-часы, затраченные каждым работником на выполнение данного контракта, которые умножаются на стоимость одного часа работы соответствующего специалиста.

Рассмотрим сущность, содержимое которой соответствовало бы форме самого отчета (Таблица 2.1). Причём, общий итог в Таблица 2.1 – это производный атрибут, который получается умножением отработанных часов на стоимость часа работы, и поэтому он не хранится в БД.

Структура Таблица 2.1 не соответствует требованиям к реляционным базам данных, по причинам, представленным ниже.

1. Номер проекта (PROJ NUM) нельзя использовать в качестве первичного ключа (или части первичного ключа) поскольку в нем имеются пустые места (null).
2. Вид элементов отношения стимулирует противоречивость данных. Например, значение JOB CLASS (специализация) в одном случае введена как «Системный аналитик», в другом «Сист. аналитик».
3. В сущности имеется очевидная избыточность данных, что приводит к следующим аномалиям:

– *аномалии обновления:* изменение JOB_CLASS для сотрудника с номером (EMP_NUM) 102 требует выполнения этого действия в нескольких строках сущности, где встречается EMR_NUM = 102;

– *аномалии включения:* для занесения информации в любую строку необходимо вводить в проект какого-нибудь сотрудника. Если сотрудник еще не включен в какой-либо проект, то для того, чтобы занести в сущность сведения о сотруднике, придется создавать фиктивный проект;

– *аномалии удаления:* если сотрудник с номером 102 увольняется, то необходимо будет удалить все строки, тле EMP_NUM = 102. При этом может быть утеряна и важная информация.

Таблица 2.1 – Периодически составляемый отчёт

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAM	JOB_CLASS	CHG_HOUR	HOURS	
Номер проекта	**Наименование проекта**	**Номер работника**	**Ф.И.О. работника**	**Должность**	**руб/ час**	**трудоза-траты в часах**	**Общие расходы**
15	Alpha Edit	101	Семен Иванов	Программист	200	120	24000
		102	Андрей Петров	Программист	200	100	20000
		110	Антон Сидоров*	Сист. аналитик	300	40	12000
18	Beta Base	103	Федот Антонов	Программист	200	250	50000
		102	Андрей Петров	Программист	200	280	56000
		111	Петр Семенов*	Проектировщик БД	250	80	20000
22	Delta CAD	104	Сидор Федотов	Программист	200	180	36000
		105	Иван Андреев	Программист	200	150	30000
		110	Антон Сидоров*	Системный аналитик	300	60	18000
	Итого:						266000

Может показаться, что структура БД работает правильно, и выполнить отчет очень просто. Однако могут возникнуть некоторые проблемы.

Каждый раз, когда на проект назначается новый сотрудник, некоторые элементы данных (PROJ_NAME, EMP_NAME, CHG_HOUR) приходится вводить без всякой на то необходимости. Представьте себе объем ненужного ввода данных, если потребуется ввести 200-300 элементов сущности. Ведь достаточно ввести соответствующий идентификационный номер сотрудника, чтобы можно было идентифицировать, например, Антона Сидорова, его специализацию и стоимость часа его работы. Поскольку лишь один сотрудник имеет номер 110, его персональные данные (имя, специальность и т.д.) не должны набираться всякий раз при обновлении базы данных. Структура, представленная в Таблица 2.1, не предоставляет такой возможности.

Избыточность данных, имеющая место в Таблица 2.1, может привести к не экономному использованию дискового пространства. К тому же избыточность данных является причиной аномалий. При вводе новых данных, например, может возникнуть такая строка:

15	Alfa Edit	**111**	П.Семёнов	Проектировщик БД	**280**	**0**

На первый взгляд эти данные корректны. Но разве, Alfa Edit это тот же проект что и Alpha Edit? А П. Семёнов – та же личность, что и Петр Семенов? Подобная путаница вызвана проблемой целостности данных, и произошла она потому, что при вводе информации не соблюдалось правило, предписывающее, что все копии избыточных данных должны быть идентичны.

При проектировании БД проблемы целостности данных, которые могут быть вызваны их избыточностью, обязательно должны приниматься во внимание.

3.1 Приведение к первой нормальной форме

В реляционной БД не должно быть повторяющихся групп. Если же такие группы существуют, то от них необходимо избавиться для того, чтобы каждая строка определяла единственное значение сущности. Процедура заключается в добавлении соответствующего компонента, по крайней мере, в столбец (или столбцы) первичного ключа. Это означает, что данные, пред-

ставленные в Таблица 2.1, должны быть преобразованы к виду, представленному в Таблица 2.2. Удалив повторяющиеся группы, мы получаем БД, приведенную к первой нормальной форме.

По изменениям, представленным в Таблица 2.2 видно, что атрибут PROJ_NUM нельзя использовать в качестве первичного ключа, поскольку номер проекта не идентифицирует уникально все остальные атрибуты сущности (строки). Например, значение PROJ_NUM, равное 15, может определять одного из пяти сотрудников. Первичный ключ, уникально определяющий любое значение атрибута, должен быть комбинацией атрибутов PROJ_NUM и EMP_NUM.

Таблица 2.2 – Организация данных: первая нормальная форма (1НФ)

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Alpha Edit	101	Семен Иванов	Программист	200	120
15	Alpha Edit	102	Андрей Петров	Программист	200	100
15	Alpha Edit	110	Антон Сидоров*	Сист. аналитик	300	40
18	Betta Prog	103	Федот Антонов	Прграммист	200	250
18	Beta Prog	102	Андрей Петров	Программист	200	280
18	Beta Prog	111	Петр Семенов*	Проектировщик БД	250	80
22	Delta CAD	104	Сидор Федотов	Программист	200	180
22	Delta CAD	105	Иван Андреев	Программист	200	150
22	Delta CAD	110	Антон Сидоров*	Системный аналитик	300	60

Зависимости можно установить с помощью *диаграммы зависимостей*, представленной на рисунке 2.9.

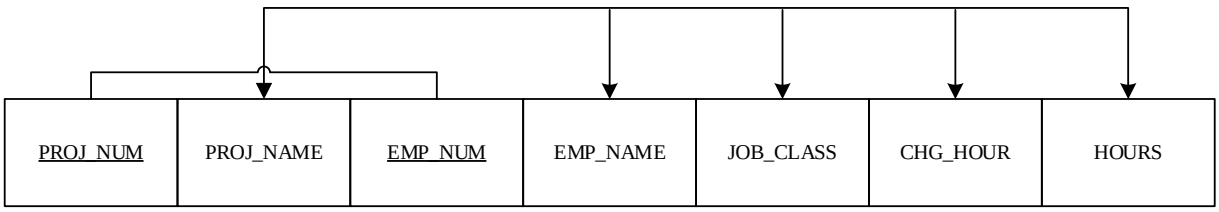


Рисунок 2.9 – Диаграмма зависимостей, основанных на первичном ключе

На рисунке 2.9 необходимо обратить внимание на следующее:

1) атрибуты первичного ключа подчеркнуты;

2) линии со стрелками над сущностями указывают все возможные желательные зависимости, т.е. зависимости, основанные на первичном ключе.

В БД имеется составной ключ, в который входят атрибуты PROJ_NUM и EMP_NUM. Любой атрибут, который, по меньшей мере, является частью ключа, называется *первичным атрибутом* или *ключевым атрибутом.* Следовательно, и PROJ_NUM, и EMP_NUM являются первичными (ключевыми) атрибутами. Соответственно, непервичный атрибут (или неключевой атрибут) не является частью ключа.

Говорят, что БД приведена к первой нормальной форме (1НФ), если в ней:

− определены все ключевые атрибуты;

− отсутствуют повторяющиеся группы. Иначе говоря, на пересечении каждого столбца и каждой строки содержится только одно значение (см. свойства реляционных таблиц), а не множество значений;

− все атрибуты зависят от первичного ключа.

Таким образом, сущность, представленная в Таблица 2.2 удовлетворяет требованиям, предъявляемым к 1НФ.

3.2 Приведение ко второй нормальной форме

В сущности, представленной на рисунке 2.9 имеются необязательные зависимости. Эти зависимости называются *частичными* и указаны линиями со стрелками в нижней части диаграммы зависимостей, представленной на рисунке 2.10.

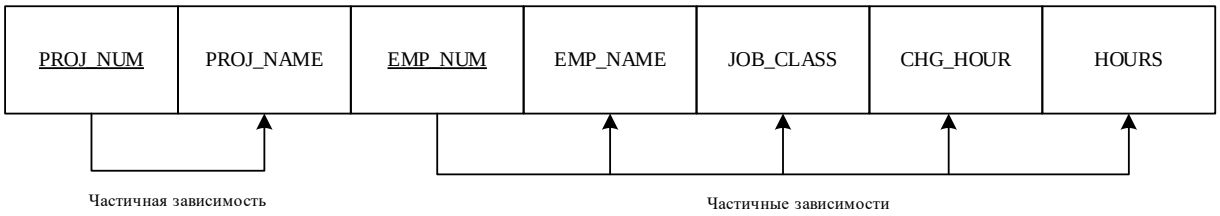


Рисунок 2.10 – Диаграмма частичных зависимостей

Необходимо знать только значение PROJ_NUM для определения PROJ_NAME, т.е. PROJ_NAME зависит только от части первичною ключа. А для определения EMP_NAME, JQB_CLASS и CHG_HOUR необходимо знать только EMP_NUM. Зависимость, определяемая только частью составного первичного ключа, называется частичной зависимостью;

Для преобразования 1НФ в 2НФ необходимо взять за основу форму 1НФ, представленную на рисунке 2.9 и проделать следующее:

− записать каждый ключевой компонент и отдельной строке, а затем в последней строке записать исходный (составной) ключ:

PROJ_NUM

EMP_NUM

PROJ_NUM EMP_NUM

− каждый компонент станет ключом в новой сущности. Иначе говоря, исходную сущность необходимо разделить на три сущности. Назовем эти таблицы PROJECT (проект), EMPLOYEE (сотрудник) и ASSIGN (назначение) соответственно;

− после каждого нового ключа записать зависимые атрибуты (для определения зависимости атрибутов используйте рисунок 2.10). Зависимости для компонентов исходного ключа можно выявить, прослеживая линии со стрелками в нижней части диаграммы зависимостей, представленной на ри-

сунке 2.10. Другими словами, три новые сущности PROJECT, EMPLOYEE и ASSIGN могут быть описаны следующим образом:

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGN (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

– поскольку количество часов, затраченных на каждый проект каждым сотрудником, зависит в сущности ASSIGN как от атрибута PROJ_NUM, так и от атрибута EMP_NUM, мы поместим их в сущность ASSIGN под названием ASSIGN_HOURS.

Результат этой операции представлен на рисунке 2.11. Теперь большинство аномалий, которые мы ранее обсуждали, устранено. Например, если мы захотим добавить/изменить/удалить запись в проекте PROJECT, то необходимо лишь обратиться к сущности PROJECT и добавить/изменить/удалить только одну строку.

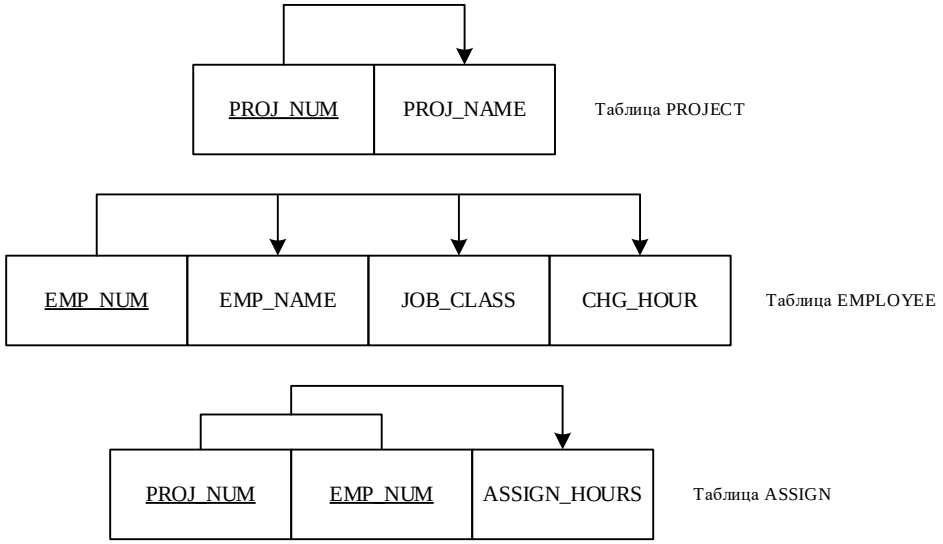


Рисунок 2.11 – Результат приведения ко второй нормальной форме (2НФ)

Говорят, что таблица приведена ко второй нормальной форме (2НФ), если:

– она приведена к первой нормальной форме;

в ней нет частичных зависимостей, т.е. нет атрибутов, зависящих от части первичного ключа.

3.3 Приведение к третьей нормальной форме

На рисунке 2.11 сущности EMPLOYEE видна транзитивная зависимость (см. рисунок 2.12), которая может стать причиной аномалий. Например, если стоимость часа работы для данной специализации, которую имеют несколько сотрудников, изменится, то это изменение необходимо будет проделать для *каждого* такого сотрудника. Если вы забудете обновить какие-то записи, связанные с изменением стоимости часа, то разные сотрудники, имеющие одну и ту же специализацию, получат разную зарплату.

Рисунок 2.12 – Диаграмма транзитивных зависимостей

На рисунке 2.12 видно, что CHG_HOUR зависит от JOB_CLASS. Поскольку ни CHG_HOUR, ни JOB_CLASS не являются первичными атрибутами – т е. ни один из этих атрибутов не является, по крайней мере, частью ключа – в этом случае говорят, что имеет место транзитивная зависимость. Проблема здесь состоит в том, что такие зависимости тоже могут стать причиной аномалии данных.

Аномалии данных, созданные в базе данных, представленной на рисунке 2.12, можно устранить, разбив фрагменты или фрагмент, на который указывает линия со стрелкой транзитивной зависимости (в нижней части диаграммы зависимостей) и поместив их в отдельную сущность. Однако атрибут JOB_CLASS должен остаться в исходной сущности (имеющей 2НФ) в качестве внешнего ключа для того чтобы, установить связь между исходной сущностью и вновь созданной. Иначе говоря, после завершения преобразования в базе данных должно быть четыре сущности (см. рисунок 2.13):

PROJECT (PROJ_NUM, PROJ_NAME)
ASSIGN (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)
EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)
JOB (JOB_CLASS, CHG_HOUR)

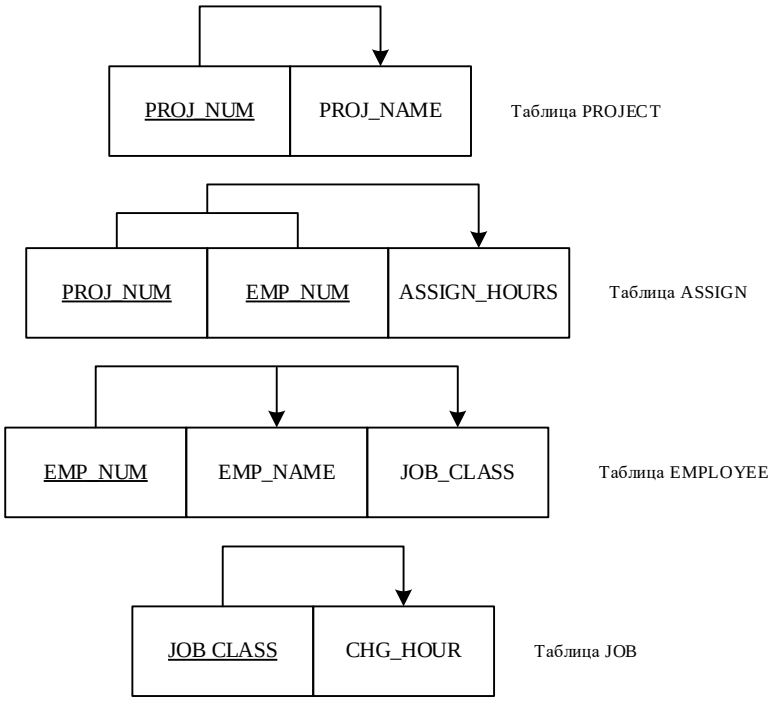


Рисунок 2.13 – Результат приведения к третьей нормальной форме (3НФ)

Выполнив это преобразование, мы устранили транзитивную зависимость оригинальной таблицы EMPLOYEE, и теперь таблица приведена к третьей нормальной форме (3НФ).

Говорят, что сущность приведена к *третьей нормальной форме (3НФ),* если:

- она приведена ко второй нормальной форме (2НФ);
- в ней отсутствуют транзитивные зависимости.

Таким образом, можно сделать вывод, что база данных находится в 3НФ.

4 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Произвести краткое описание предметной области (предметная область лабораторной работы соответствует варианту предметной области из курсового проекта). Подробное описание предметной области включить в раздел «Анализ предметной области» курсового проекта. Для выполнения этого этапа необходимо:
   - проанализировать информационные потребности пользователей;
   - сформировать состав документов, подлежащих включению в БД;
   - разработать состав и форму представления информации по каждому документу;
   - создать кодификаторы для упорядочения данных в БД;
   - определить задачи и функции системы.
2. Разработать первые два уровня логической модели базы данных:
   - диаграмму сущность-связь (ERD) в нотации П.Чена;
   - модель данных, основанную на ключах (KB) по методологии IDEF1.
3. Нормализовать отношения в базе данных до третьей и четвертой нормальной формы.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Этапы проектирования баз данных.
2. Инфологическое проектирование.
3. Задачи, решаемые на этапе инфологического проектирования.
4. Задачи, решаемые на этапе логического проектирования.
5. Задачи, решаемые на этапе физического проектирования
6. Сущности. Отличие понятия типа сущности и элемента сущности. Способы представления сущности.
7. Для чего предназначена диаграмма «сущность-связь»?
8. Как представляется диаграмма «сущность-связь» в нотации П. Чена?
9. Какие существуют типы связей между сущностей и чем они отличаются?
10. Какова цель модель, основанная на ключах. В чем ее особенность.

11. Атрибуты и их типы. Правила атрибутов. Классификация атрибутов.

12. Связи. Понятие безусловной, условной, биусловной, рекурсивной связи. Формализация связи. Фундаментальные виды связей. Формализация связей 1:1, 1:M, M:N.

13. Нормальные формы. Общие свойства нормальных форм. Виды нормальных форм.

14. Функциональная зависимость. Виды функциональных зависимостей. Функционально полная зависимость.

15. Функционально полная и частичная зависимости неключевого атрибута от составного ключа.

16. Определение транзитивной зависимости.

17. Условия нахождения отношений в первой нормальной форме.

18. Негативные последствия нахождение отношения лишь в первой нормальной форме.

19. Условия нахождения отношений во второй нормальной форме.

20. Условия нахождения отношений в третьей нормальной форме.

21. Условия нахождения отношений в усиленной третьей нормальной форме.

22. Многозначные зависимости.

23. Условия нахождения отношений в четвертой нормальной форме.

24. Условия нахождения отношений в пятой нормальной форме проекции-соединения.