Теория проектирования реляционных баз данных

Введение

При проектировании базы данных решаются две основные проблемы:

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области, и было, по возможности, лучшим (эффективным, удобным и т. д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных? Эту проблему обычно называют проблемой физического проектирования баз данных.

В случае реляционных баз данных нет общих рецептов по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Поэтому ограничимся только существенными вопросами логического проектирования реляционных баз данных. Более того, не будем касаться определения ограничений целостности общего вида, а ограничимся ограничениями первичного и внешнего ключей. Будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять базы данных, и какие атрибуты должны быть у этих отношений.

Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс *нормализации* схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы – значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1НФ или 1NF);
- вторая нормальная форма (2НФ или 2NF);
- третья нормальная форма (3НФ или 3NF);
- нормальная форма Бойса-Кодда (НФБК или BCNF);
- четвертая нормальная форма (4НФ или 4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

Основные свойства нормальных форм такие:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели:

- избежать избыточности хранения данных;
- устранить аномалии обновления отношений.

Эти цели являются актуальными для информационных систем оперативной обработки транзакций (On-Line Transaction Processing – OLTP), которым свойственны частые обновления базы данных, и потому аномалии обновления могут сильно вредить эффективности приложения. В информационных системах оперативной аналитической обработки (On-Line Analytical Processing – OLAP), в частности, в системах поддержки принятия решений, базы данных в основном используются для выборки данных. Поэтому аномалиями обновления можно пренебречь. Из этого не следует, что принципы нормализации непригодны при проектировании баз данных OLAP-приложений. Даже если схема такой базы данных должна быть денормализована по соображениям эффективности, то чтобы получить правильную денормализованную схему, нужно сначала понять, как выглядит нормализованная схема.

В основе метода нормализации лежит *декомпозиция* отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы. Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии *функциональной зависимости*.

Функциональные зависимости

Для демонстрации основных идей данной темы, будет использоваться несколько измененная версия отношения поставок SP, содержащая атрибут City, представляющий город соответствующего поставщика. Это измененное отношение будет называться SCP.

Sno	City	Pno	Qty
1	Смоленск	1	100
1	Смоленск	2	100
2	Владимир	1	200
2	Владимир	2	200
3	Владимир	2	300
4	Смоленск	2	400
4	Смоленск	4	400
4	Смоленск	5	400

Следует четко различать:
a) значение этого отношения в определенный момент времени;
b) и набор всех возможных значений, которые данное отношение может принимать в различные моменты времени.

Определение 1. Пусть R - это отношение, а X и Y - произвольные подмножества множества атрибутов отношения R. Тогда Y ***функционально зависимо*** от X, что в символическом виде записывается как X -> Y ⇔ ∀ значение множества X связано в точности с одним значением множества Y.

Примеры ФЗ, которым удовлетворяет отношение SCP в данном состоянии:

{ Sno } -> { City }
{ Sno, Pno } -> { Qty }
{ Sno, Pno } -> { City }
{ Sno, Pno } -> { City, Qty }
{ Sno, Pno } -> { Sno }
{ Sno, Pno } -> { Sno, Pno, City, Qty }
{ Sno } -> { Qty }
{ Qty } -> { Sno }

Левая и правая стороны ФЗ будут называются ***детерминантом*** и ***зависимой частью*** соответственно.

Определение 2. Пусть R является переменной-отношением, а X и Y - произвольными подмножествами множества атрибутов переменной-отношения R. Тогда X -> Y ⇔ ∀ ***допустимого значения отношения R*** ∀ значение X связано в точности с одним значением Y.

Определение 2а. Пусть R(A1, A2, ..., An) - схема отношения. Функциональная зависимость, обозначаемая X -> Y между двумя наборами атрибутов X и Y, которые являются подмножествами R определяет ограничение на возможность существования кортежа в некотором отношении r. Ограничение означает, что для любых двух кортежей t1 и t2 в r, для которых имеет место t1[X] = t2[X], также имеет место t1[Y] = t2[Y].

- Если ограничение на схеме отношения R утверждает, что не может быть более одного кортежа со значением атрибутов X в любом отношении экземпляре отношения r, то X является потенциальным ключом R. Это означает, что X -> Y для любого подмножества атрибутов Y из R. Если X является потенциальным ключ R, то X -> R.
- Если X -> Y в R, это не означает, что Y -> X в R.

Функциональная зависимость является семантическим свойством, т. е. свойством значения атрибутов.

Примеры безотносительных ко времени ФЗ для переменной-отношения SCP:

{ Sno, Pno } -> { Qty }
{ Sno, Pno } -> { City }
{ Sno, Pno } -> { City, Qty }
{ Sno, Pno } -> { Sno }
{ Sno, Pno } -> { Sno, Pno, City, Qty }
{ Sno } -> { City }

Следует обратить внимание на ФЗ, которые выполняются для отношения SCP, но не выполняются «всегда» для переменной-отношения SCP:

{ Sno } -> { Qty }
{ Qty } -> { Sno }

Если X является потенциальным ключом переменной-отношения R, то все атрибуты Y переменной-отношения R должны быть обязательно ФЗ от X (это следует из определения потенциального ключа). Если переменная-отношение R удовлетворяет ФЗ X -> Y и X **не** является потенциальным ключом, то R будет характеризоваться некоторой избыточностью. Например, в случае отношения SCP сведения о том, что каждый данный поставщик находится в данном городе, будут повторяться много раз (это хорошо видно из таблицы).
Эта избыточность приводит к разным ***аномалиям обновления***, получившим такое название по историческим причинам. Под этим понимаются определенные трудности, появляющиеся при выполнении операций обновления INSERT, DELETE и UPDATE. Рассмотрим избыточность, соответствующую ФЗ (Sno -> City). Ниже поясняются проблемы, которые возникнут при выполнении каждой из указанных операций обновления.

- Операция INSERT. Нельзя поместить в переменную-отношение SCP информацию о том, что некоторый поставщик находится в определенном городе, не указав сведения хотя бы об одной детали, поставляемой этим поставщиком.
- Операция DELETE. Если из переменной-отношения SCP удалить кортеж, который является единственным для некоторого поставщика, будет удалена не только информация о поставке поставщиком некоторой детали, но также информация о том, что этот поставщик находится в определенном городе. В действительности проблема заключается в том, что в переменной-отношении SCP содержится слишком много собранной в одном месте информации, поэтому при удалении некоторого кортежа теряется слишком много информации.
- Операция UPDATE. Название города для каждого поставщика повторяется в переменной-отношении SCP несколько раз, и эта избыточность приводит к возникновению проблем при обновлении.

Для решения всех этих проблем, как предлагалось выше, необходимо выполнить декомпозицию переменной-отношения SCP на две следующие переменные-отношения.

S' { Sno, City }
SP { Sno, Pno, Qty }

Даже если ограничиться рассмотрением ФЗ, которые выполняются «всегда», множество ФЗ, выполняющихся для всех допустимых значений данного отношения, может быть все еще очень большим. Поэтому встает задача сокращения множества ФЗ до компактных размеров. Важность этой задачи вытекает из того, что ФЗ являются ограничениями целостности. Поэтому при каждом обновлении данных в СУБД все они должны быть проверены. Следовательно, для заданного множества ФЗ S желательно найти такое множество T, которое (в идеальной ситуации) было бы гораздо меньшего размера, чем множество S, причем каждая ФЗ множества S могла бы быть заменена ФЗ множества T. Если бы такое множество T было найдено, то в СУБД достаточно было бы использовать ФЗ из множества T, а ФЗ из множества S подразумевались бы автоматически.

Очевидным способом сокращения размера множества ФЗ было бы исключение ***тривиальных зависимостей***, т. е. таких, которые не могут не выполняться. Примером тривиальной ФЗ для отношения SCP может быть { Sno, Pno } -> { Sno }.

Определение 3. ФЗ (X -> Y) *тривиальная* $\Leftrightarrow Y \subseteq X$.

Одни ФЗ могут подразумевать другие ФЗ. Например, зависимость

{ Sno, Pno } -> { City, Qty }

подразумевает следующие ФЗ

{ Sno, Pno } -> City
{ Sno, Pno } -> Qty

В качестве более сложного примера можно привести переменную-отношение R с атрибутами A, B и C, для которых выполняются ФЗ (A -> B) и (B -> C). В этом случае также выполняется ФЗ (A -> C), которая называется ***транзитивной*** ФЗ.

Определение 4. Множество всех ФЗ, которые задаются данным множеством ФЗ S, называется ***замыканием*** S и обозначается символом S+.

Из сказанного выше становится ясно, что для выполнения сформулированной задачи следует найти способ вычисления S+ на, основе S.

Первая попытка решить эту проблему принадлежит Армстронгу (Armstrong), который предложил набор ***правил вывода*** новых ФЗ на основе заданных (эти правила также называются аксиомами Армстронга).

Пусть в перечисленных ниже правилах A, B и C – произвольные подмножества множества атрибутов заданной переменной-отношения R, а символическая запись AB означает { A, B }. Тогда правила вывода определяются следующим образом.

1. Правило ***рефлексивности***: $(B \subseteq A) \Rightarrow (A \text{ -> } B)$.
2. Правило ***дополнения***: $(A \text{ -> } B) \Rightarrow AC \text{ -> } BC$.
3. Правило ***транзитивности***: $(A \text{ -> } B)$ и $(B \text{ -> } C) \Rightarrow (A \text{ -> } C)$.

Каждое и этих правил может быть непосредственно доказано на основе определения ФЗ. Более того, эти правила являются ***полными*** в том смысле, что для заданного множества ФЗ S минимальный набор ФЗ, которые подразумевают все зависимости из множества S, может быть выведен из S на основе этих правил. Они также являются ***исчерпывающими***, поскольку никакие дополнительные ФЗ (т.е. ФЗ, которые не подразумеваются ФЗ множества S) с их помощью не могут быть выведены. Иначе говоря, эти правила могут быть использованы для получения замыкания S+.

Из трех описанных выше правил для упрощения задачи практического вычисления замыкания S+ можно вывести несколько дополнительных правил. (Примем, что D - это другое произвольное подмножество множества атрибутов R.)

4. Правило ***самоопределения***: A -> A.
5. Правило ***декомпозиции***: $(A \text{ -> } BC) \Rightarrow (A \text{ -> } B)$ и $(A \text{ -> } C)$.
6. Правило ***объединения***: $(A \text{ -> } B)$ и $(A \text{ -> } C) \Rightarrow (A \text{ -> } BC)$.
7. Правило ***композиции***: $(A \text{ -> } B)$ и $(C \text{ -> } D) \Rightarrow (AC \text{ -> } BD)$.

Кроме того, Дарвен (Darwen) доказал следующее правило, которое назвал ***общей теоремой объединения***:
8. $(A \text{ -> } B)$ и $(C \text{ -> } D) \Rightarrow (A\ (C–B) \text{ -> } BD)$.

Упражнение. Пусть дана некоторая переменная-отношение R с атрибутами A, B, C, D, E, F и следующими ФЗ:

A -> BC
B -> E
CD -> EF

Показать, что для переменной-отношения R также выполняется ФЗ (AD -> F), которая вследствие этого принадлежит замыканию заданного множества ФЗ.

1. A -> BC (дано)
2. A -> C (следует из п. 1 согласно правилу декомпозиции)
3. AD -> CD (следует из п. 2 согласно правилу дополнения)
4. CD -> EF (дано)
5. AD -> EF (следует из п. 3 и 4 согласно правилу транзитивности)
6. AD -> F (следует из п. 5 согласно правилу декомпозиции) // Конец упр.

Хотя эффективный алгоритм для вычисления замыкания S+ на основе множества ФЗ S так и не сформулирован, можно описать алгоритм, который определяет, будет ли данная ФЗ находиться в данном замыкании. Для этого, прежде всего, следует дать определение понятию ***суперключ***.

Определение 5. *Суперключ* переменной-отношения R - это множество атрибутов переменной-отношения R, которое содержит в виде подмножества (но не обязательно собственного подмножества), по крайней мере, один *потенциальный* ключ.

Из этого определения следует, что суперключи для данной переменной-отношения R - это такие подмножества K множества атрибутов переменной-отношения R, что ФЗ (K -> A) истинна для каждого атрибута A переменной-отношения R.

Предположим, что известны некоторые ФЗ, выполняющиеся для данной переменной-отношения, и требуется определить потенциальные ключи этой переменной-отношения. По определению потенциальными ключами называются неприводимые суперключи. Таким образом, выясняя, является ли данное множество атрибутов K суперключом, можно в значительной степени продвинуться к выяснению вопроса, является ли K потенциальным ключом. Для этого нужно определить, будет ли набор атрибутов переменной-отношения R множеством всех атрибутов, функционально зависящих от K. Таким образом, для заданного множества зависимостей S, которые выполняются для переменной-отношения R, необходимо найти способ определения множества всех атрибутов переменной-отношения R, которые функционально

зависимы от K, т.е. так называемое замыкание K^+ множества K для S. Простой алгоритм вычисления этого замыкания имеет вид.

```
function Closure(K: SetOfAttr; S: SetOfFD): SetOfAttr;
var
        Jold, Jnew: SetOfAttr;
begin
        Jnew := K;
        repeat
                Jold := Jnew;
                foreach ((X -> Y) in S) do
                        if (X ⊆ Jnew) then Jnew=Jnew ∪ Y;
        until (Jold = Jnew);
        return Jold;
end; // of Closure
```

Упражнение. Пусть дана переменная-отношение R с атрибутами A, B, C, D, E и F и следующими ФЗ:

A -> BC
E -> CF
B -> E
CD -> EF

Вычислить замыкание $\{ A, B \}^+$ множества атрибутов { A, B }, исходя из заданного множества ФЗ.

1. Присвоим Jold и Jnew начальное значение - множество {A,B}.
2. Выполним внутренний цикл четыре раза - по одному разу для каждой заданной ФЗ. На первой итерации (для зависимости A -> BC) будет обнаружено, что левая часть действительно является подмножеством замыкания Jnew. Таким образом, к Jnew можно добавить атрибуты B и C. Jnew теперь представляет собой множество {A,B,C}.
3. На второй итерации (для зависимости E -> CF) обнаруживается, что левая часть не является подмножеством полученного до этого момента результата, который, таким образом, остается неизменным.
4. На третьей итерации (для зависимости B -> E) к Jnew будет добавлено множество E, которое теперь будет иметь вид {A, B, C, E}.
5. На четвертой итерации (для зависимости CD -> EF) Jnew останется неизменным.
6. Далее внутренний цикл выполняется еще четыре раза. На первой итерации результат останется прежним, на второй он будет расширен до {A, B, C, E, F}, а на третьей и четвертой - снова не изменится.
7. Наконец после еще одного четырехкратного прохождения цикла замыкание Jnew останется неизменным и весь процесс завершится с результатом $\{A, B\}^+ = \{A, B, C, E, F\}$. // Конец упр.

Выводы.

1. Для заданного множества ФЗ S легко можно указать, будет ли заданная ФЗ (X -> Y) следовать из S, поскольку это возможно тогда и только тогда, когда множество Y является подмножеством замыкания X^+ множества X для заданного множества S. Таким образом, представлен простой способ определения, будет ли данная ФЗ (X->Y) включена в замыкание S^+ множества S.
2. Напомним определение понятия суперключа. Суперключ переменной-отношения R - это множество атрибутов переменной-отношения R, которое в виде подмножества (но необязательно собственного подмножества) содержит по крайней мере один потенциальный ключ. Из этого определения прямо следует, что суперключи для данной пеменной-отношения R - это такие подмножества K множества атрибутов переменной-отношения R, что ФЗ (K -> A) будет истинна для каждого атрибута A переменной-отношения R. Другими словами, множество K является суперключ тогда и только тогда, когда замыкание K^+ для множества K в пределах заданного мноства ФЗ является множеством абсолютно всех атрибутов переменной-отношения R. (Кроме того, множество K является потенциальным ключом тогда и только тогда, когда оно является неприводимым суперключом).

Определение 6. Два множества ФЗ S1 и S2 ***эквивалентны*** тогда и только тогда, когда они ***являются покрытиями*** друг для друга, т. е. S1+ = S2+.

[Пример

Consider the following two sets of functional dependencies:
F= {A ->C, AC ->D,E ->AD, E ->H} and G = {A ->CD, E ->AH}.
Check whether or not they are equivalent.
Answer:
To show equivalence, we prove that G is covered by F and F is covered by G.

Proof that G is covered by F:
{A}+ = {A, C, D} (with respect to F), which covers A ->CD in G
{E}+ = {E, A, D, H, C} (with respect to F), which covers E ->AH in G
Proof that F is covered by G:
{A}+ = {A, C, D} (with respect to G), which covers A ->C in F
{A, C}+= {A, C, D} (with respect to G), which covers AC ->D in F
{E}+ = {E, A, H, C, D} (with respect to G), which covers E ->AD and E ->H in F

Конец примера]

Каждое множество ФЗ эквивалентно, по крайней мере, одному ***неприводимому*** множеству.

Определение 7. Множество ФЗ является ***неприводимым*** тогда и только тогда, когда оно обладает всеми перечисленными ниже свойствами.

- Каждая ФЗ этого множества имеет одноэлементную правую часть.
- Ни одна ФЗ множества не может быть устранена без изменения замыкания этого множества.
- Ни один атрибут не может быть устранен из левой части любой ФЗ данного множества без изменения замыкания множества.

Если I является неприводимым множеством, которое эквивалентно множеству S, то проверка выполнения ФЗ из множества I автоматически обеспечит выполнение ФЗ из множества S.

Пример. Рассмотрим переменную-отношение деталей P с функциональными зависимостями, перечисленными ниже.

Pno -> Pname
Pno -> Color
Pno -> Weight
Pno -> City

Нетрудно заметить, что это множество функциональных зависимостей является неприводимым:

- правая часть каждой зависимости содержит только один атрибут,
- левая часть, очевидно, является неприводимой,
- ни одна из функциональных зависимостей не может быть опущена без изменения замыкания множества (т. е. без утраты некоторой информации).

В противоположность этому приведенные ниже множества функциональных зависимостей не являются неприводимыми.

1. Pno -> { Pname, Color }
   Pno -> Weight
   Pno -> City

(Правая часть первой ФЗ не является одноэлементным множеством.)

2. { Pno, Pname } -> Color
   Pno -> Pname
   Pno -> Weight
   Pno -> City

(Первую ФЗ можно упростить, опустив атрибут Pname в левой части без изменения замыкания, т. е. она не является неприводимой слева.)

3. Pno -> Pno
   Pno -> Pname
   Pno -> Color
   Pno -> Weight
   Pno -> City

(Здесь первую ФЗ можно опустить без изменения замыкания.) // Конец прим.

Можно сделать утверждение, что для любого множества ФЗ существует, по крайней мере, одно эквивалентное множество, которое является неприводимым. Это достаточно легко продемонстрировать на следующем примере. Пусть дано исходное множество ФЗ S. Тогда благодаря правилу декомпозиции можно без утраты общности предположить, что каждая ФЗ в этом множестве S имеет одноэлементную правую часть. Далее для каждой ФЗ f из этого множества S следует проверить каждый атрибут A в левой части зависимости f. Если множество S и множество зависимостей,

полученное в результате устранения атрибута A в левой части зависимости f, эквивалентны, значит этот атрибут следует удалить. Затем для каждой оставшейся во множестве S зависимости f, если множества S и S \ {f} эквивалентны, следует удалить зависимость f из множества S. Получившееся в результате таких действий множество S является неприводимым и эквивалентно исходному множеству S.

Упражнение. Пусть дана переменная-отношение R с атрибутами A, B, C, D и следующими функциональными зависимостями.

A -> BC
B -> C
A -> B
AB -> C
AC -> D

Найти неприводимое множество функциональных зависимостей эквивалентное данному множеству.

1. Прежде всего, следует переписать заданные ФЗ таким образом, чтобы каждая из них имела одноэлементную правую часть.

A -> B
A -> C
B -> C
A -> B
AB -> C
AC -> D

Нетрудно заметить, что зависимость A -> B записана дважды, так что одну из них можно удалить.

2. Затем в левой части зависимости AC -> D может быть опущен атрибут C, поскольку дана зависимость A -> C, из которой по правилу дополнения можно получить зависимость A -> AC. Кроме того, дана зависимость AC -> D, из которой по правилу транзитивности можно получить зависимость A -> D. Таким образом, атрибут C в левой части исходной зависимости AC -> D является избыточным.
3. Далее можно заметить, что зависимость AB -> C может быть исключена, поскольку дана зависимость A -> C, из которой по правилу дополнения можно получить зависимость AB -> CB, а затем по правилу декомпозиции - зависимость AB -> C.
4. Наконец зависимость A -> C подразумевается зависимостями A -> B и B -> C, так что она также может быть отброшена. В результате получается неприводимое множество зависимостей.

A -> B
B -> C
A -> D // Конец упр.

Множество ФЗ I, которое неприводимо и эквивалентно другому множеству ФЗ S, называется неприводимым покрытием множества S. Таким образом, с тем же успехом в системе вместо исходного множества ФЗ S может использоваться его неприводимое покрытие I (здесь следует повторить, что для вычисления неприводимого эквивалентного покрытия I необязательно вычислять замыкание S+). Однако необходимо отметить, что для заданного множества ФЗ не всегда существует уникальное неприводимое покрытие.

[Дополнения к ФЗ

Проектирование базы данных может быть выполнено с использованием двух подходов: снизу вверх (bottom-up) или сверху вниз (top-down).

- Методология проектирования bottom-up (также называемая методологией синтеза) рассматривает основные связи между отдельными атрибутами в качестве отправной точки и использует их, чтобы построить схемы отношений схем. Этот подход не пользуется популярностью на практике, потому что она страдает от проблемы того, чтобы собрать большое число бинарных связей между атрибутами в качестве отправной точки. На пактике это сделать почти невозможно.
- Методология проектирования top-down (также называемая методологией анализа) начинается с некоторого набора отношений, состоящих из атрибутов. Затем отношения анализируются отдельно или совместно, в результате чего происходит их декомпозиция до тех пор, пока не будут достигнуты все желаемые свойства.

Неявными целями обеих методологий являются сохранение информации и минимизация избыточности.

Процесс нормализации схем отношений, основанный на операции декомпозии, должен обладать следующими свойства:

- неаддитивностью JOIN или JOIN без потерь информации (NJP) , которая гарантирует, что в результате декомпозиции не появятся лишние кортежи.
- свойством сохранения зависимостей (DPP), которое гарантирует, что каждая функциональная зависимость будет представлена в каком-либо отдельном отношении после декомпозиции.

Свойство NJP является очень критическим и должно быть достигнуто любой ценой. Свойство DPP желательно, но не всегда достижимо.

Определение. Суперключ в схеме отношения R(A1, A2, ... , An) - это набор атрибутов S из R со свойством, что ни для каких двух кортежей t1 и t2 в любом отношении над схемой R не будет выполняться равенство t1[S] = t2[S].

Определение. Потенциальный ключ K - это суперключ с дополнительным свойством: удаление любого атрибута из K приведет к тому, что K перестанет быть суперключом.

Определение. Атрибут в схеме отношения R называется первичным атрибутом R, если он является членом некоторого потенциального ключа R. Атрибут называется непервичным, если он не является первичным атрибутом, то есть, если он не является членом какого-либо потенциального ключа.

Алгоритм поиска минимального покрытия F для множества функциональных зависимостей E

Вход: Множество функциональных зависимостей E.
1. Пусть F: = E.
2. Заменить каждую функциональную зависимость X -> {A1, A2, ..., An} из F на n функциональных зависимостей X -> A1, X -> A2, ..., X -> An.
3. Для каждой функциональной зависимости X -> A из F
    Для каждого атрибута B, который является элементом X
        если { {F – {X -> A} } ∪ { (X – {B}) -> A} } эквивалентно F
            заменить X -> A на (X – {B}) -> A в F.
4. Для каждой оставшейся функциональной зависимости X -> A из F
    если {F - {X -> A}} эквивалентно F,
        удалить X -> A из F.

Пример.

Let the given set of FDs be E : {B>A, D>A, AB>D}.We have to find the minimal cover of E.

- All above dependencies are in canonical form (that is, they have only one attribute on the right-hand side), so we have completed step 1 of Algorithm and can proceed to step 2. In step 2 we need to determine if AB > D has any redundant attribute on the left-hand side; that is, can it be replaced by B > D or A > D?
- Since B > A, by augmenting with B on both sides (IR2), we have BB > AB, or B > AB (i). However, AB>D as given (ii).
- Hence by the transitive rule (IR3), we get from (i) and (ii), B > D. Thus AB > D may be replaced by B > D.
- We now have a set equivalent to original E, say E': {B>A, D>A, B>D}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.
- In step 3 we look for a redundant FD in E'. By using the transitive rule on B > D and D > A, we derive B > A. Hence B > A is redundant in E' and can be eliminated.
- Therefore, the minimal cover of E is {B > D, D > A}.

Этот алгоритм может быть использован для синтеза отношения из заданного множества зависимостй E.

Алгоритм нахождения ключа K схемы отношения R для заданного множество функциональных зависимостей F

Вход: Схема отношения R и множество функциональных зависимостей F на атрибутах R.
1. Пусть K := R.
2. Для каждого атрибута из K
    {
        вычислить Closure({K - A), F);
        если замыкание содержит все атрибуты из R, то установите K: = K - {A}
    };

Заметьте, что алгоритм определяет только один ключ из множества возможных ключей на R; возвращаемый ключ зависит от порядка, в котором атрибуты удаляются из R на шаге 2.

Конец дополнений к ФЗ]

Нормальные формы, основанные на функциональных зависимостях

Определение 1. Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

Первая нормальная форма - это условие, согласно которому каждый компонент каждого кортежа является атомарным значением. В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение.

Определение 2. Функциональная зависимость R.X -> R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение 3. Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z -> R.X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Определение 4. Неключевым атрибутом называется любой атрибут отношения, не входящий в состав потенциального ключа (в частности, первичного).

Определение 5. Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Вторая нормальная форма

Определение 6. (В этом определении предполагается, что единственным ключом отношения является первичный ключ.) Отношение R находится во второй нормальной форме (2НФ) в том и только в том случае, когда оно находится в 1НФ, и каждый неключевой атрибут полностью зависит от первичного ключа.

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Определение 6а. Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1НФ, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Здесь и далее мы не будем приводить примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

Третья нормальная форма

Определение 7. (Снова определение дается в предположении существования единственного ключа.) Отношение R находится в третьей нормальной форме (3НФ) в том и только в том случае, если оно находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Определение 7а. Отношение R находится в третьей нормальной форме (3НФ) в том и только в том случае, если оно находится в 2НФ, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

[Дополнения к 2НФ и 3НФ

Exercise 1. Consider the universal relation R = {A, B, C, D, E, F, G, H, I} and the set of functional dependencies
F = { {A, B} -> {C}, {A} -> {D, E}, {B} -> {F}, {F} -> {G, H}, {D} -> {I, J} }.
What is the key for R? Decompose R into 2NF, then 3NF relations.
Answer:
A minimal set of attributes whose closure includes all the attributes in R is a key. Since the closure {A, B}+ = R, one key of R is {A, B} (in this case, it is the only key).
To normalize R intuitively into 2NF then 3NF, we take the following steps:
First, identify partial dependencies that violate 2NF. These are attributes that are functionally dependent on either parts of the key, {A} or {B}, alone. We can calculate the closures {A}+ and {B}+ to determine partially dependent attributes:

o {A}+ = {A, D, E, I, J}, hence {A} -> {D, E, I, J} ({A} -> {A} is a trivial dependency)
o {B}+ = {B, F, G, H}, hence {B} -> {F, G, H} ({B} -> {B} is a trivial dependency)
To normalize into 2NF, we remove the attributes that are functionally dependent on part of the key (A or B) from R and place them in separate relations R1 and R2, along with the part of the key they depend on (A or B), which are copied into each of these relations but also remains in the original relation, which we call R3 below:
o R1 = {A, D, E, I, J},
o R2 = {B, F, G, H},
o R3 = {A, B, C}
The new keys for R1, R2, R3 are underlined.
Next, we look for transitive dependencies in R1, R2, R3. The relation R1 has the transitive dependency {A} -> {D} -> {I, J}, so we remove the transitively dependent attributes {I, J} from R1 into a relation R11 and copy the attribute D they are dependent on into R11. The remaining attributes are kept in a relation R12. Hence, R1 is decomposed into R11 and R12 as follows:
o R11 = {D, I, J},
o R12 = {A, D, E}
The relation R2 is similarly decomposed into R21 and R22 based on the transitive dependency {B} -> {F} -> {G, H}:
o R21 = {F, G, H},
o R22 = {B, F}
The final set of relations in 3NF are {R11, R12, R21, R22, R3}

Exercise 2. Repeat exercise 1for the following different set of functional dependencies
G = { {A, B} -> {C}, {B, D} -> {E, F}, {A, D} -> {G, H}, {A} -> {I}, {H} -> {J} }.
Answer:
To help in solving this problem systematically, we can first find the closures of all single attributes to see if any is a key on its own as follows:
o {A}+ = {A, I},
o {B}+ = {B},
o {C}+ = {C},
o {D}+ = {D},
o {E}+ = {E},
o {F}+ = {F},
o {G}+ = {G},
o {H}+ = {H, J},
o {I}+ = {I},
o {J}+ = {J}
Since none of the single attributes is a key, we next calculate the closures of pairs of attributes that are possible keys:
o {A, B}+ = {A, B, C, I},
o {B, D}+ = {B, D, E, F},
o {A, D}+ = {A, D, G, H, I, J}
None of these pairs are keys either since none of the closures includes all attributes. But the union of the three closures includes all the attributes:
o {A, B, D}+ = {A, B, C, D, E, F, G, H, I}
Hence, {A, B, D} is a key.
Based on the above analysis, we decompose as follows, in a similar manner to **Exercise 1**, starting with R = {A, B, D, C, E, F, G, H, I}.
The first-level partial dependencies on the key (which violate 2NF) are:
o {A, B} -> {C, I},
o {B, D} -> {E, F},
o {A, D} -> {G, H, I, J}
Hence, R is decomposed into R1, R2, R3, R4 (keys are underlined):
o R1 = {A, B, C, I},
o R2 = {B, D, E, F},
o R3 = {A, D, G, H, I, J},
o R4 = {A, B, D}
Additional partial dependencies exist in R1 and R3 because {A} -> {I}. Hence, we remove {I} into R5, so the following relations are the result of 2NF decomposition:
o R1 = {A, B, C},
o R2 = {B, D, E, F},
o R3 = {A, D, G, H, J},
o R4 = {A, B, D},
o R5 = {A, I}
Next, we check for transitive dependencies in each of the relations (which violate 3NF). Only R3 has a transitive dependency {A, D} -> {H} -> {J}, so it is decomposed into R31 and R32 as follows:
o R31 = {H, J},
o R32 = {A, D, G, H}

The final set of 3NF relations is {R1, R2, R31, R32, R4, R5}

Конец дополнений к 2НФ и 3НФ]

Нормальная форма Бойса-Кодда

Определение 3НФ неадекватно при выполнении следующих условий.
1. Переменная-отношение имеет два (или более) потенциальных ключа.
2. Эти потенциальные ключи являются составными.
3. Два или более потенциальных ключей перекрываются (т. е. имеют по крайней мере один общий атрибут).

Поэтому впоследствии исходное определение 3НФ было заменено более строгим определением нормальной формы Бойса-Кодда (НФБК).

Определение 8. Детерминант - любой атрибут (или группа атрибутов), от которого полностью функционально зависит некоторый другой атрибут.

Определение 9. Отношение R находится в нормальной форме Бойса-Кодда (НФБК) в том и только в том случае, если каждый детерминант является потенциальным ключом.

Отношение в НФБК позволяет исключить все виды аномалий обновления, связанные с функциональными зависимостями. Однако отношение в НФБК все еще может быть плохо структурированным и допускать аномалии обновления. В этом случае причиной аномалий обновления является наличие многозначных зависимостей.

[Упражнение.

Дана схема отношения R(A, B, C, D, E, F, G, H, I, J), для которой выполняется множество функциональных зависимостей S={AB->C, A->DE, B->F, B->G, F->G, D->HIJ}.
a) Выполняются ли функциональные зависимости AF->BI и AB->DJ для R? Ответ пояснить.
b) Найти все потенциальные ключи для R.
c) Показать этапы преобразования R в BCNF.
Решение:
a)
- AF->BI: AF->ADEF->ADEFG->ADEFGHIJ not
- AB->DJ: AB->ABC->ABCDE->ABCDEF->ABCDEFG->ABCDEFGHIJ yes
b)
The only key is AB:
- super-key: see above
- minimal: A->ADHIJ, B->BFG
A and B do not occur on any right side of a FD. Therefore, they must be part of any key. This shows that AB is the only key.
c)
A->DE violates BCNF:
R1 = (ADE, {A->DE})
R2 = (ABCFGHIJ, {AB->C, B->F, B->G, F->G})
R2 is not in BCNF, since B->FG violates BCNF:
R21 = (BFG, {B->F, B->G, F->G})
R22 = (ABCHIJ, AB->C)
R21 is not in BCNF, since F->G violates BCNF.
R211 = (FG, {F->G})
R212 = (BF, {B->F})
decomposition R={R1, R211, R212, R22} lost dependencies: D->HIJ, B->G

Конец упражнения]

[Дополнение ко всему

Properties of Relational Decompositions

Dependency Preservation Property of a Decomposition

It would be useful if each functional dependency *X -> Y* specified in *F* either appeared directly in one of the relation schemas *Ri* in the decomposition *D* or could be inferred from the dependencies that appear in some *Ri*. Informally, this is the *dependency preservation condition.* We want to preserve the dependencies because each dependency in *F* represents a constraint on the database. If one of the dependencies is not represented in some individual relation *Ri* of the decomposition, we cannot enforce this constraint by dealing with an individual relation.We may have to join multiple relations so as to include all attributes involved in that dependency.

It is not necessary that the exact dependencies specified in *F* appear themselves in individual relations of the decomposition *D*. It is sufficient that the union of the dependencies that hold on the individual relations in *D* be equivalent to *F*.We now define these concepts more formally.

Definition. Given a set of dependencies *F* on *R*, the **projection** of *F* on *Ri*, denoted by $\pi Ri(F)$ where *Ri* is a subset of *R*, is the set of dependencies $X \rightarrow Y$ in *F*+ such that the attributes in $X \cup Y$ are all contained in *Ri*. Hence, the projection of *F* on each relation schema *Ri* in the decomposition *D* is the set of functional dependencies in *F*+, the closure of *F*, such that all their left- and right-hand-side attributes are in *Ri*. We say that a decomposition *D* = {*R*1, *R*2, ..., *Rm*} of *R* is **dependency-preserving** with respect to *F* if the union of the projections of *F* on each *Ri* in *D* is equivalent to *F*; that is, $((\pi R1(F)) \cup ... \cup (\pi Rm(F)))+ = F+$.

If a decomposition is not dependency-preserving, some dependency is **lost** in the decomposition. To check that a lost dependency holds, we must take the JOIN of two or more relations in the decomposition to get a relation that includes all leftand right-hand-side attributes of the lost dependency, and then check that the dependency holds on the result of the JOIN — an option that is not practical.

Claim 1. It is always possible to find a dependency-preserving decomposition *D* with respect to *F* such that each relation *Ri* in *D* is in 3NF.

Nonadditive (Lossless) Join Property of a Decomposition

Another property that a decomposition *D* should possess is the nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.

Definition. Formally, a decomposition *D* = {*R*1, *R*2, ..., *Rm*} of *R* has the **lossless (nonadditive) join property** with respect to the set of dependencies *F* on *R* if, for *every* relation state *r* of *R* that satisfies *F*, the following holds, where * is the NATURAL JOIN of all the relations in *D*: $*(\pi R1(r), ..., \pi Rm(r)) = r$.

The word loss in *lossless* refers to *loss of information*, not to loss of tuples. If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous or invalid information. We prefer the term
nonadditive join because it describes the situation more accurately. Although the term *lossless join* has been popular in the literature, *we will henceforth use the term nonadditive join*, which is self-explanatory and unambiguous. The nonadditive join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations. We may, however, sometimes use the term **lossy design** to refer to a design that represents a loss of information.

Testing Binary Decompositions for the Nonadditive Join Property

There is a special case of a decomposition called a **binary decomposition** — decomposition of a relation *R* into two relations.

Property NJB (Nonadditive Join Test for Binary Decompositions).

A decomposition *D* = {*R*1, *R*2} of *R* has the lossless (nonadditive) join property with respect to a set of functional dependencies *F* on *R if and only if* either

- The FD $((R1 \cap R2) \rightarrow (R1 - R2))$ is in *F*+, or
- The FD $((R1 \cap R2) \rightarrow (R2 - R1))$ is in *F*+

Successive Nonadditive Join Decompositions

Claim 2 (Preservation of Nonadditivity in Successive Decompositions).
If a decomposition *D* = {*R*1, *R*2, ..., *Rm*} of *R* has the nonadditive (lossless) join property with respect to a set of functional dependencies *F* on *R*, and if a decomposition *Di* = {*Q*1, *Q*2, ..., *Qk*} of *Ri* has the nonadditive join property with respect to the projection of *F* on *Ri*, then the decomposition *D*2 = {*R*1, *R*2, ..., *Ri* -1, *Q*1, *Q*2, ..., *Qk*, *Ri*+1, ..., *Rm*} of *R* has the nonadditive join property with respect to *F*.

Algorithms for Relational Database Schema Design

We now give three algorithms for creating a relational decomposition from a universal relation. Each algorithm has specific properties, as we discuss next.

Dependency-Preserving Decomposition into 3NF Schemas

Algorithm creates a dependency-preserving decomposition *D* = {*R*1, *R*2, ..., *Rm*} of a universal relation *R* based on a set of functional dependencies *F*, such that each *Ri* in *D* is in 3NF. It guarantees only the dependency-preserving property; it does *not* guarantee the nonadditive join property. The first step of Algorithm is to find a minimal cover *G* for *F*. Note that multiple minimal covers may exist for a given set *F*. In such cases the algorithms can potentially yield multiple alternative designs.

Algorithm. Relational Synthesis into 3NF with Dependency Preservation
Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.
1. Find a minimal cover *G* for *F*;
2. For each left-hand-side *X* of a functional dependency that appears in *G*, create a relation schema in *D* with attributes $\{X \cup \{A1\} \cup \{A2\} ... \cup \{Ak\}\}$, where $X \rightarrow A1$, $X \rightarrow A2$, ..., $X \rightarrow Ak$ are the only dependencies in *G* with *X* as the left-hand-side (*X* is the key of this relation);
3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

Example of Algorithm. Consider the following universal relation:
U(Emp_ssn, Pno, Esal, Ephone, Dno, Pname, Plocation)
Emp_ssn, Esal, Ephone refer to the Social Security number, salary, and phone number of the employee.
Pno, Pname, and Plocation refer to the number, name, and location of the project.
Dno is department number.
The following dependencies are present:
FD1: Emp_ssn $\rightarrow$ {Esal, Ephone, Dno}
FD2: Pno $\rightarrow$ { Pname, Plocation}
FD3: {Emp_ssn, Pno} $\rightarrow$ {Esal, Ephone, Dno, Pname, Plocation}

By virtue of FD3, the attribute set {Emp_ssn, Pno} represents a key of the universal relation. Hence *F*, the set of given FDs includes {Emp_ssn –> {Esal, Ephone, Dno}; Pno –> {Pname, Plocation}; {Emp_ssn, Pno} –> {Esal, Ephone, Dno, Pname, Plocation}}.

By applying the minimal cover Algorithm, in step 3 we see that Pno is a redundant attribute in Emp_ssn, Pno –> {Esal, Ephone, Dno}. Moreover, Emp_ssn is redundant in {Emp_ssn, Pno} –> {Pname, Plocation}. Hence the minimal cover consists of FD1 and FD2 only (FD3 being completely redundant) as follows (if we group attributes with the same left-hand side into one FD):
Minimal cover *G*: {Emp_ssn –> {Esal, Ephone, Dno}; Pno –> {Pname, Plocation}}

By applying Algorithm to the above Minimal cover *G*, we get a 3NF design consisting of two relations with keys Emp_ssn and Pno as follows:
*R*1 (Emp_ssn, Esal, Ephone, Dno)
*R*2 (Pno, Pname, Plocation)
An observant reader would notice easily that these two relations have lost the original information contained in the key of the universal relation U (namely, that there are certain employees working on certain projects in a many-to-many relationship).

Thus, while the algorithm does preserve the original dependencies, it makes no guarantee of preserving all of the information.Hence, the resulting design is a *lossy* design.

Claim 3. Every relation schema created by Algorithm is in 3NF. (We will not provide a formal proof here;6 the proof depends on *G* being a minimal set of dependencies.)

Algorithm is called a **relational synthesis algorithm**, because each relation schema *Ri* in the decomposition is synthesized (constructed) from the set of functional dependencies in *G* with the same left-hand-side *X*.

Nonadditive Join Decomposition into BCNF Schemas

The next algorithm decomposes a universal relation schema *R* = {*A*1, *A*2, ..., *An*} into a decomposition *D* = {*R*1, *R*2, ..., *Rm*} such that each *Ri* is in BCNF *and* the decomposition *D* has the lossless join property with respect to *F*. Algorithm utilizes Property NJB and Claim 2 (preservation of nonadditivity in successive decompositions)
to create a nonadditive join decomposition *D* = {*R*1, *R*2, ..., *Rm*} of a universal relation *R* based on a set of functional dependencies *F*, such that each *Ri* in *D* is in BCNF.

Algorithm. Relational Decomposition into BCNF with Nonadditive Join Property
Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.
1. Set *D* := {*R*} ;
2. While there is a relation schema *Q* in *D* that is not in BCNF do
{
choose a relation schema *Q* in *D* that is not in BCNF;
find a functional dependency *X* -> *Y* in *Q* that violates BCNF;
replace *Q* in *D* by two relation schemas (*Q* – *Y*) and (*X* ∪ *Y*);
} ;

Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

By now we know that it is *not possible to have all three of the following:*
(1) guaranteed nonlossy design,
(2) guaranteed dependency preservation, and
(3) all relations in BCNF.

Algorithm. Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property
Input: A universal relation *R* and a set of functional dependencies *F* on the attributes of *R*.
1. Find a minimal cover *G* for *F*.
2. For each left-hand-side *X* of a functional dependency that appears in *G*, create a relation schema in *D* with attributes {*X* ∪ {*A*1} ∪ {*A*2} ... ∪ {*Ak*} }, where *X* –> *A*1, *X* -> *A*2, ..., *X* -> *Ak* are the only dependencies in *G* with *X* as left-hand-side (*X* is the key of this relation).
3. If none of the relation schemas in *D* contains a key of *R*, then create one more relation schema in *D* that contains attributes that form a key of *R*.
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation *R* is considered redundant if *R* is a projection of another relation *S* in the schema; alternately, *R* is subsumed by *S*.

Example of Algorithm. Let us revisit the example given earlier. The minimal cover *G* holds as before. The second step produces relations *R*1 and *R*2 as before. However, now in step 3, we will generate a relation corresponding to the key {Emp_ssn, Pno}. Hence, the resulting design contains:
*R*1 (Emp_ssn , Esal, Ephone, Dno)
*R*2 (Pno, Pname, Plocation)
*R*3 (Emp_ssn, Pno)
This design achieves both the desirable properties of dependency preservation and nonadditive join.

Конец дополнения ко всему]

Многозначные зависимости и четвертая нормальная форма

В качестве примера рассмотрим переменную-отношение **HCTX**, содержащую информацию о курсах обучения, преподавателях и учебниках. В этой переменной-отношении атрибуты, описывающие преподавателей и учебники, принимают в качестве значений *отношения*. Каждый кортеж переменной-отношения **HCTX** состоит из атрибутов названия курса (**COURSE**), а также атрибута-отношения с именами преподавателей (**TEACHERS**) и атрибута-отношения с названиями учебников (**TEXTS**). Смысл каждого кортежа состоит в том, что соответствующий курс может преподаваться любым из указанных преподавателей с использованием всех указанных учебников. Предположим, что для заданного курса может быть определено произвольное количество соответствующих преподавателей и учебников. Более

того, допустим, что преподаватели и рекомендуемые учебники совершенно независимы друг от друга. Это значит, что независимо от того, кто преподает данный курс, всегда используется один и тот же набор учебников. Наконец, допустим, что определенный преподаватель или определенный учебник может быть связан с любым количеством курсов.

```
<HCTX>
        <Course>
                <Name>Физика</Name>
                <Teachers>
                        <Teacher>Проф. Фейнман</Teacher>
                        <Teacher>Проф. Арнольд</Teacher>
                </Teachers>
                <Texts>
                        <Text>Механика</Text>
                        <Text>Оптика</Text>
                </Texts>
        </Course>
        <Course>
                <Name>Математика</Name>
                <Teachers>
                        <Teacher>Проф. Арнольд</Teacher>
                </Teachers>
                <Texts>
                        <Text>Механика</Text>
                        <Text>Анализ</Text>
                        <Text>Алгебра</Text>
                </Texts>
        </Course>
</HCTX>
```

Пусть необходимо исключить атрибуты, принимающие в качестве значений отношения. Один из способов заключается в простой замене переменной-отношения **HCTX** переменной-отношением **CTX** с тремя скалярными атрибутами { **Course**, **Teacher** и **Text** }, как показано в таблице.

CTX

Course	**Teacher**	**Text**
Физика	Проф. Фейнман	Механика
Физика	Проф. Фейнман	Оптика
Физика	Проф. Арнольд	Механика
Физика	Проф. Арнольд	Оптика
Математика	Проф. Арнольд	Механика
Математика	Проф. Арнольд	Анализ
Математика	Проф. Арнольд	Алгебра

Очевидно, что переменная-отношение **CTX** характеризуется значительной **избыточностью**, вследствие чего возникнут **аномалии обновления**. Например, для добавления информации о том, что курс ‘Физика’ может читаться новым преподавателем, необходимо создать два новых кортежа, по одному для каждого используемого учебника. Как можно избежать появления таких проблем? Ситуацию можно улучшить, если выполнить декомпозицию переменной-отношения **CTX** на две проекции (например, с именами **CT** и **CX**) с атрибутами { **Course**, **Teacher** } и { **Course**, **Text** } соответственно. При этом переменная-отношение **CTX** может быть восстановлена путем обратного соединения проекций **CT** и **CX**, и поэтому данная декомпозиция была выполнена без потерь.

CT

Course	**Teacher**
Физика	Проф. Фейнман
Физика	Проф. Арнольд
Математика	Проф. Арнольд

CX

Course	**Text**
Физика	Механика
Физика	Оптика
Математика	Механика
Математика	Анализ

С неформальной точки зрения, очевидно, что переменная-отношение **CTX** спроектирована неудачно и ее декомпозиция на проекции **CT** и **CX** является более удачным решением. Но с формальной точки зрения это совсем неочевидно. Переменная-отношение **CTX** вообще не имеет функциональных зависимостей (за исключением таких тривиальных, как **Course** -> **Course**). Фактически переменная-отношение **CTX** находится в **НФБК**, поскольку все ее атрибуты входят в состав ее ключа, а любая, подобная переменная-отношение обязательно находится в **НФБК**. Проекции **CT** и **CX** также являются полностью ключевыми, а потому находятся в **НФБК.**

Способы разрешения «проблем», которые связаны с переменными-отношениями в **НФБК**, подобными переменной-отношению **CTX**, были сформулированы Фейгином (Fagin) в строгом теоретическом виде с использованием понятия **многозначной зависимости** (**МЗ**), которую можно считать обобщением **ФЗ** в том смысле, что каждая **ФЗ** также является **МЗ**. Точнее говоря, **ФЗ** - это **МЗ**, в которой множество зависимых значений, соответствующее заданному значению детерминанта, всегда является одноэлементным множеством. В переменной-отношении **CTX** есть две **МЗ**, которые не являются **ФЗ**:

Course ->> **Teacher**
Course ->> **Text**

Формальное определение МЗ. Пусть **A**, **B** и **C** являются произвольными подмножествами множества атрибутов переменной-отношения **R**. Тогда подмножество **B многозначно зависит** от подмножества **A**, что символически выражается записью **A** ->> **B**, тогда и только тогда, когда множество значений **B**, соответствующее заданной паре (значение **A**, значение **C**) переменной-отношения **R**, зависит от **A**, но не зависит от **C**.

Другое определение МЗ. Многозначная зависимостей **X ->> Y** в схеме отношения **R**, где **X** и **Y** подмножества **R**, указывает следующее ограничение на любое отношении **r** над схемой **R**: если два кортежа t1 и t2 существуют в **r** такие, что t1[X] = t2[X], то должны существовать также два кортежа t3 и t4 в **r** над **R** со следующими свойствами, где **Z = (R - (X -> Y))**:

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

Всякий раз, когда имеет место **X ->> Y**, то говорят, что **X** многозначно определяет **Y**. В силу симметрии в определении, когда имеет место X ->> Y в **R**, то имеет место и **X ->> Z**. Следовательно, **X ->> Y** влечет **X ->> Z**, и поэтому иногда пишут **X ->> Y | Z**.

Лемма Фейгина. Для данной переменной-отношения **R** { **A**, **B**, **C** } **МЗ A** ->> **B** выполняется тогда и только тогда, когда также выполняется **МЗ A** ->> **C**.

Таким образом, **МЗ** всегда образуют связанные пары, поэтому обычно их представляют вместе в символическом виде **A** ->> **B** | **C**.

Возвращаясь к исходной задаче с переменной-отношением **CTX**, можно отметить следующее: проблема с переменной-отношением **CTX** возникает из-за того, что она содержит **МЗ**, которые не являются **ФЗ**. Проекции **CT** и **CX** не содержат **МЗ**, а потому они действительно представляют собой некоторое усовершенствование исходной структуры. Замена исходной переменной-отношения **CTX** двумя проекциями **CT** и **CX** является правомочной в соответствии с теоремой Фейгина.

Теорема Фейгина. Пусть **A**, **B** и **C** являются множествами атрибутов переменной-отношения **R** { **A**, **B**, **C** }. Переменная-отношение **R** будет равна соединению ее проекций { **A**, **B** } и { **A**, **C** } тогда и только тогда, когда для переменной-отношения **R** выполняется **МЗ A** ->> **B** | **C**.

Эта теорема является более строгой версией теоремы Хита. Теперь, следуя работе Фейгина, можно дать определение *четвертой нормальной формы*.

Определение тривиальной МЗ. **МЗ A** ->> **B** называется тривиальной, если выполняется хотя бы одно из условий:
1. Множество **A** является надмножеством **B**;
2. Объединение **A** и **B** образует весь заголовок отношения.

Определение 4НФ. Переменная-отношение R находится в **четвертой нормальной форме** (**4НФ**) тогда и только тогда, когда в случае существования таких подмножеств **A** и **B** атрибутов этой переменной-отношения **R**, для которых

выполняется нетривиальная **МЗ A ->> B**, все атрибуты переменной-отношения **R** также функционально зависят от атрибута **A**.

Это определение можно также сформулировать в следующей эквивалентной форме: переменная-отношение **R** находится в **4НФ**, если она находится **НФБК** и все **МЗ** в переменной-отношении **R** фактически представляют собой **ФЗ** от ее ключей.

Переменная-отношение **СТХ** не находится в **4НФ**, поскольку содержит **МЗ**, которые не являются **ФЗ**, не говоря уже о том, что последняя должна быть еще и **ФЗ** от ключа. Однако, обе ее проекции, **СТ** и **СХ**, находятся в **4НФ**. Следовательно, **4НФ** обеспечивает лучшую структуру данных по сравнению с **НФБК**. Фейгин показал, что **4НФ** всегда является достижимой, но в реальной практике такая декомпозиция (или даже декомпозиция до **НФБК**) не всегда оказывается полезной и нужной.

В заключение можно констатировать следующее:

- полностью ключевое отношение всегда находится в НФБК поскольку оно не имеет ФЗ.
- полностью ключевое отношение, которое не имеет ФЗ но имеет МЗ, не находится в 4НФ.
- отношение, которое не находится в 4НФ в связи с нетривиальным МЗ должны быть декомпозировано, чтобы преобразовать его в набор отношений в 4НФ.
- декомпозиция устраняет избыточность, вызванную МЗ.

[Дополнение к 4НФ

Определение. Многозначная зависимостей X ->> Y в схеме отношения R, где X и Y подмножества R, указывает следующее ограничение на любое отношении r над схемой R: если два кортежа t1 и t2 существуют в r такие, что t1[X] = t2[X], то должны существовать также два кортежа T3 и T4 в r над R со следующими свойствами, где Z = (R - (X -> Y)):

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

Всякий раз, когда имеет место X ->> Y, то говорят, что X multidetermines Y. В силу симметрии в определении, когда имеет место X ->> Y в R, то имеет место и X ->> Z. Следовательно, X ->> Y влечет X ->> Z, и поэтому иногда пишут X ->> Y | Z.

Определение. Схема отношения R в 4NF по отношению к множеству зависимостей F (которая включает в себя функциональные зависимости и многозначных зависимостей), если для каждой нетривиальной многозначной зависимости X ->> Y в F+ X является суперключом для R.

Можем констатировать следующее:

- полностью ключевое отношение всегда находится в BCNF поскольку оно не имеет FD.
- полностью ключевое отношение, которое не имеет FDs но имеет MVD, не находится в 4NF.
- отношение, которое не находится в 4NF в связи с нетривиальным MVD должны быть декомпозировано, чтобы преобразовать его в набор отношений в 4NF.
- декомпозиция устраняет избыточность, вызванную MVD.

Конец дополнения к 4НФ]

Зависимости соединения и пятая нормальная форма

До сих пор предполагалось, что единственной необходимой или допустимой операцией в процессе нормализации является замена переменной-отношения по правилам декомпозиции без потерь *только двумя* ее проекциями. Однако существуют переменные-отношения, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые *можно* подвергнуть декомпозиции без потерь на три или более проекций. Подобные переменные-отношения обозначаются термином "n-декомпозируемая переменная-отношение".

В качестве примера можно рассмотреть переменную-отношение **SPJ** из базы данных «Поставщики, детали и проекты» (в целях упрощения изложения атрибут Qty исключен). Эта переменная-отношение состоит, только из ключевых атрибутов, не содержит нетривиальных **ФЗ** и **МЗ** и потому находится в **4НФ**. На рисунках показаны следующие компоненты:

- Исходное отношение **SPJ**.
- Три бинарные проекции, **SP**, **PJ** и **JS**, переменной-отношения **SPJ**.
- Результат соединения проекций **SP** и **PJ** по атрибуту **Pno**.
- Соединение этого результата с проекцией **JS** по комбинации атрибутов (**Jno**, **Sno**).

SPJ (Исходное отношение)

Sno	Pno	Jno
1	1	2
1	2	1
2	1	1
1	1	1

SP

Sno	Pno
1	1
1	2
2	1

PJ

Pno	Jno
1	2
2	1
1	1

JS

Jno	Sno
2	1
1	1
1	2

SPJ^ (Соединение **SP** и **PJ** по атрибуту **Pno**)

Sno	Pno	Jno
1	1	2
1	2	1
2	1	1
2	1	2
1	1	1

SPJ (Соединение **SPJ^** и **JS** по комбинации атрибутов (**Jno**, **Sno**))

Sno	Pno	Jno
1	1	2
1	2	1
2	1	1
1	1	1

В результате первого соединения получается копия исходной переменной-отношения **SPJ^** с одним дополнительным (излишним) кортежем, а в результате второго соединения этот лишний кортеж исключается. Иначе говоря, исходная переменная-отношение **SPJ** является 3-декомпозируемой. Переменная-отношение **SPJ** может быть получена только в результате соединения всех трех ее бинарных проекций, но не любых двух из них.

Представленный пример выполнен в терминах *отношений*, а не *переменных-отношений*. 3-декомпозируемость переменной-отношения SPJ может быть более фундаментальным и не зависящим от времени свойством (т.е. свойством, которое удовлетворяется для всех допустимых значений данной переменной-отношения), если данная переменная-отношение удовлетворяет определенному не зависящему от времени ограничению целостности:

ЕСЛИ кортежи (**s1, p1, j2**), (**s2, p1, j1**), (**s1, p2, j1**) присутствуют в **SPJ**,
ТО кортеж (**s1, p1, j1**) также присутствует в **SPJ**.

Следует обратить внимание на циклическую структуру этого ограничения. *Переменная-отношение будет n-декомпозируемой для n>2 тогда и только тогда, когда она удовлетворяет некоторому циклическому ограничению.* Ограничение 3-декомпозируемости для краткости принято называть ***3Д-ограничением***. Поскольку **3Д**-ограничение удовлетворяется тогда и только тогда, когда переменная-отношение равносильна соединению некоторых ее проекций, такое ограничение называется **зависимостью соединения** (**ЗС**).

Определение зависимости соединения. Пусть **R** является переменной-отношением, а **A**, **B**, ..., **Z** - произвольными подмножествами множества ее атрибутов. Переменная-отношение **R** удовлетворяет **зависимости соединения** *{ **A**, **B**, ..., **Z**

} (читается «звездочка **A**, **B**, ..., **Z**») тогда и только тогда, когда любое допустимое значение переменной-отношения **R** эквивалентно соединению ее проекций по подмножествам атрибутов **A**, **B**, ..., **Z**.

Например, если использовать сокращенную символьную запись **SP** для подмножества { **Sno**, **Pno** } множества атрибутов переменной-отношения **SPJ** и аналогично использовать сокращения **PJ** и **JS** для двух других подмножеств, то переменная-отношение **SPJ** будет удовлетворять зависимости соединения * { **SP**, **PJ**, **JS** }. Отсюда ясно, что переменная-отношение **SPJ** с зависимостью соединения *{ **SP**, **PJ**, **JS** } может быть 3-декомпозируемой.

Теорема Фейгина (которая рассматривалась ранее) утверждает, что переменная-отношение **R** { **A**, **B**, **C** } может быть декомпозирована без потерь на проекции с атрибутами { **A**, **B** } и { **A**, **C** } тогда и только тогда, когда для переменной-отношения **R** выполняются **МЗ A** ->> **B** и **A** ->> **C**.

Теперь теорема Фейгина может быть сформулирована иначе. Переменная-отношение **R** { **A**, **B**, **C** } удовлетворяет зависимости соединения * { **AB**, **AC** } тогда и только тогда, когда она удовлетворяет многозначной зависти **A** ->> **B** | **C**.

Поскольку эту теорему можно использовать в качестве определения многозначной зависимости, то либо многозначная зависимость является частным случаем зависимости соединения, либо (что эквивалентно) зависимость соединения является обобщение понятия многозначной зависимости. Формально получается следующее **A** ->> **B** | **C** = *{ **AB**, **AC** }.

Рассмотренный выше пример обнаруживает следующую проблему: переменная-отношение **SPJ** содержит зависимость соединения, которая не является ни многозначной, ни функциональной. Такую переменную-отношение можно декомпозировать на меньшие компоненты, а именно – на проекции, определяемые зависимостью соединения. Данный процесс декомпозиции может повторяться до тех пор, пока все результирующие переменные-отношения не будут находиться в *пятой нормальной форме*.

Определение 5НФ. Переменная-отношение **R** находится в **пятой нормальной форме** (**5НФ**), которую иногда иначе называют **проекционно-соединительной нормальной формой** (**ПСНФ**), тогда и только тогда, когда каждая нетривиальная **ЗС** в переменной-отношении **R** подразумевается ее потенциальными ключами.

Определение тривиальной ЗС. ЗС *{ **A**, **B**, ..., **Z** } называется **тривиальной** тогда и только тогда, когда одна из проекций **A**, **B**, ..., **Z** является проекцией, идентичной R (т.е. проекцией по всем атрибутам переменной-отношения **R**).

Определение. ЗС * { **A**, **B**, ..., **Z** } подразумевается потенциальными ключами тогда и только тогда, когда каждое подмножество атрибутов **A**, **B**, ..., **Z** фактически является суперключом для данной переменной-отношения.

Переменная-отношение **SPJ** не находится в **5НФ**. Она удовлетворяет некоторой зависимости соединения, а именно - **ЗД**-ограничению, которое, конечно же, не подразумевается ее единственным потенциальным ключом (этот ключ является комбинацией всех ее атрибутов). Иначе говоря, переменная-отношение **SPJ** не находится в **5НФ**, поскольку она может быть 3-декомпозирована и возможность такой декомпозиции не подразумевается тем фактом, что комбинация атрибутов { **Sno**, **Pno**, **Jno** } является ее потенциальным ключом. Наоборот, после 3-декомпозиции проекции **SP**, **PJ**, **JS** находятся в **5НФ**, поскольку в них вовсе нет нетривиальных зависимостей соединения.

Возникает вопрос, следует ли выполнять такую декомпозицию? Ответ: если переменная-отношение находится в **5НФ**, то гарантируется, что она не содержит аномалий, которые могут быть исключены посредством ее разбиения на проекции. Конечно, это не значит, что данная переменная-отношение свободна от всех возможных аномалий. Это всего лишь означает, что она свободна от аномалий, которые могут быть исключены с помощью разбиения на проекции. 5НФ является окончательной нормальной формой по отношению к операциям проекции и соединения.

Общая схема процедуры нормализации

Каждый этап процесса нормализации заключается в разбиении на проекции переменных-отношений, полученных на предыдущем этапе. При этом на каждом этапе нормализации существующие ограничения используются для выбора тех проекций, которые будут получены в этот раз. Весь процесс можно неформально определить с помощью перечисленных ниже правил.

1. Переменную-отношение в **1НФ** следует разбить на такие проекции, которые позволят исключить все функциональные зависимости, не являющиеся неприводимыми. В результате будет получен набор переменных-отношений в **2НФ**.
2. Полученные переменные-отношения в **2НФ** следует разбить на такие проекции, которые позволят исключить все существующие транзитивные функциональные зависимости. В результате будет получен набор переменных-отношений в **3НФ**.
3. Полученные переменные-отношения в **3НФ** следует разбить на проекции, позволяющие исключить любые оставшиеся функциональные зависимости, в которых детерминанты не являются потенциальными ключами. В результате такого приведения будет получен набор переменных-отношений в **НФБК**. Замечание. Правила 1-3 могут

быть объединены в одно: "Исходную переменную-отношение следует разбить на проекции, позволяющие исключить все функциональные зависимости, в которых детерминанты не являются потенциальными ключами".

4. Полученные переменные-отношения в **НФБК** следует разбить на проекции, позволяющие исключить любые многозначные зависимости, которые не являются функциональными. В результате будет получен набор переменных-отношений в **4НФ**.
5. Полученные переменные-отношения в **4НФ** следует разбить на проекции, позволяющие исключить любые зависимости соединения, которые не подразумеваются потенциальными ключами. В результате будет получен набор переменных-отношений в **5НФ**.