Реляционная модель данных

Введение

Основоположником теории реляционных баз данных является британский учёный Эдгар Кодд, который в 1970 году опубликовал первую работу по реляционной модели данных. Наиболее распространенная трактовка реляционной модели данных принадлежит Кристоферу Дейту. Согласно Дейту, реляционная модель состоит из трех частей: *структурной*, *целостностной* и *манипуляционной*.

Структурная часть реляционной модели

Структурная часть реляционной модели описывает, из каких объектов состоит реляционная модель. Постулируется, что основной структурой данных, используемой в реляционной модели, являются нормализованные «n-арные» отношения. Основными понятиями структурной части реляционной модели являются *тип данных*, *домен*, *атрибут*, *схема отношения*, *схема базы данных*, *кортеж*, *отношение*, *потенциальный*, *первичный* и *альтернативные ключи*, *реляционная база данных*.

Понятие *типа данных* в реляционной модели полностью адекватно понятию типа данных в языках программирования.

Понятие *домена* можно считать уточнением типа данных. Домен можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл. Домен характеризуется следующими свойствами:

- домен имеет уникальное имя (в пределах базы данных),
- домен определен на некотором типе данных или на другом домене,
- домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена,
- домен несет определенную смысловую нагрузку.

Говорят, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл. Основное значение доменов состоит в том, что домены ограничивают сравнения. Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип.
Понятие домена помогает правильно моделировать предметную область. Не все домены обладают логическим условием, ограничивающим возможные значения домена. В таком случае множество возможных значений домена совпадает с множеством возможных значений типа данных.

Атрибут отношения – это пара вида <имя_атрибута, имя_домена >. Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Схема отношения – это именованное множество упорядоченных пар <имя_атрибута, имя_домена>. *Степенью* или «*арностью*» схемы отношения является мощность этого множества. *Схема базы данных* в реляционной модели – это множество именованных схем отношений. Понятие схемы отношения близко к понятию структурного типа в языках программирования (например, **record** в языке Pascal или **struct** в языке C).

Кортеж, соответствующий данной схеме отношения, – это множество упорядоченных пар <имя_атрибута, значение_атрибута>, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. Значение атрибута должно быть допустимым значением домена, на котором определен данный атрибут. *Степень* или «*арность*» кортежа совпадает с «арностью» соответствующей схемы отношения.

Отношение, определенное на множестве из n доменов (не обязательно различных), содержит две части: заголовок (схему отношения) и тело (множество из m кортежей). Значения n и m называются соответственно *степенью* и *кардинальностью отношения*. Отношения обладают следующими свойствами.

- В отношении нет одинаковых кортежей. Действительно, тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые элементы.
- Кортежи не упорядочены (сверху вниз). Причина следующая – тело отношения есть множество, а множество не упорядочено.
- Атрибуты не упорядочены (слева направо). Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. В таблицах в отличие от отношений столбцы упорядочены.

- Каждый кортеж содержит ровно одно значение для каждого атрибута. Отношение, удовлетворяющее этому свойству, называется нормализованным или представленным в первой нормальной форме (1NF).
- Все значения атрибутов атомарны, т. е. не обладают структурой. Трактовка этого свойства в последнее время претерпела существенные изменения. Исторически в большинстве публикаций по базам данных считалось недопустимым использовать атрибуты со структурированными значениями. (А в большинстве изданий это считается таковым и поныне.) В настоящее время принимается следующая точка зрения: тип данных атрибута может быть произвольным, а, следовательно, возможно существование отношения с атрибутами, значениями которых также являются отношения. Именно так в некоторых постреляционных базах данных реализована работа со сколь угодно сложными типами данных, создаваемых пользователями.

В реляционной модели каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения (т.е. любое отношение должно обладать уникальным ключом).

Непустое подмножество множества атрибутов схемы отношения будет *потенциальным ключом* тогда и только тогда, когда оно будет обладать свойствами *уникальности* (в отношении нет двух различных кортежей с одинаковыми значениями потенциального ключа) и *неизбыточности* (никакое из собственных подмножеств множества потенциального ключа не обладает свойством уникальности).

В реляционной модели по традиции один из потенциальных ключей должен быть выбран в качестве *первичного* ключа, а все остальные потенциальные ключи будут называться *альтернативными*.

Реляционная база данных – это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных.

Целостностная часть реляционной модели

В целостностной части реляционной модели фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это *целостность сущностей* и *ссылочная целостность* (или *целостность внешних ключей*).

Простой объект реального мира представляется в реляционной модели как кортеж некоторого отношения. Требование целостности сущностей заключается в следующем: каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения (т.е. любое отношение должно обладать потенциальным ключом). Вполне очевидно, что если данное требование не соблюдается (т.е. кортежи в рамках одного отношения не уникальны), то в базе данных может храниться противоречивая информация об одном и том же объекте. Поддержание целостности сущностей обеспечивается средствами СУБД. Это осуществляется с помощью двух ограничений:
1) при добавлении записей в таблицу проверяется уникальность их первичных ключей,
2) не позволяется изменение значений атрибутов, входящих в первичный ключ.

Сложные объекты реального мира представляются в реляционной модели данных в виде кортежей нескольких нормализованных отношений, связанных между собой. При этом
1) связи между данными отношениями описываются в терминах *функциональных зависимостей*,
2) для отражения функциональных зависимостей между кортежами разных отношений используется дублирование первичного ключа одного отношения (родительского) в другое (дочернее). Атрибуты, представляющие собой копии ключей родительских отношений, называются внешними ключами.

Внешний ключ в отношении R2 – это непустое подмножество множества атрибутов FK этого отношения, такое, что: a) существует отношение R1 (причем отношения R1 и R2 необязательно различны) с потенциальным ключом CK; b) каждое значение внешнего ключа FK в текущем значении отношения R2 обязательно совпадает со значением ключа CK некоторого кортежа в текущем значении отношения R1.

Требование ссылочной целостности состоит в следующем:
- для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

Как правило, поддержание ссылочной целостности также возлагается на СУБД. Например, она может не позволить пользователю добавить запись, содержащую внешний ключ с несуществующим (неопределенным) значением.

Манипуляционная часть реляционной модели

Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными – *реляционную алгебру* и *реляционное исчисление*. Принципиальное различие между реляционной алгеброй и

реляционным исчислением заключается в следующем: реляционная алгебра в явном виде предоставляет набор операций, а реляционное исчисление представляет систему обозначений для определения требуемого отношения в терминах данных отношений. Формулировка запроса в терминах реляционной алгебры носит предписывающий характер, а в терминах реляционного исчисления – описательный характер. Говоря неформально, реляционная алгебра носит процедурный характер (пусть на очень высоком уровне), а реляционное исчисление – непроцедурный характер.

Реляционная алгебра

Реляционная алгебра является основным компонентом реляционной модели, опубликованной Коддом, и состоит из восьми операторов, составляющих две группы по четыре оператора:

1) Традиционные операции над множествами: *объединение* (UNION), *пересечение* (INTERSECT), *разность* (MINUS) и *декартово произведение* (TIMES). Все операции модифицированы, с учетом того, что их операндами являются отношения, а не произвольные множества.
2) Специальные реляционные операции: *ограничение* (WHERE) , *проекция* (PROJECT), *соединение* (JOIN) и *деление* (DIVIDE BY).

Результат выполнения любой операции реляционной алгебры над отношениями также является отношением. Эта особенность называется свойством *реляционной замкнутости*. Утверждается, что поскольку реляционная алгебра является замкнутой, то в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры. Если рассматривать свойство реляционной замкнутости строго, то каждая реляционная операция должна быть определена таким образом, чтобы выдавать результат с надлежащим типом отношения (в частности, с соответствующим набором атрибутов или заголовком). Для достижения этой цели вводится новый оператор *переименование* (RENAME), предназначенный для переименования атрибутов в определенном отношении.

В качестве основы для последующих обсуждений рассмотрим упрощенный синтаксис выражений реляционной алгебры в форме БНФ.

реляционное_выражение ::=
    унарное_выражение | *бинарное_выражение*

унарное_выражение ::=
    переименование | *ограничение* | *проекция*

переименование ::=
    терм **RENAME** *имя_атрибута* **AS** *имя_атрибута*

терм ::=
    имя_отношения | (*реляционное_выражение*)

ограничение ::=
    терм **WHERE** *логическое_выражение*

проекция ::=
    терм | *терм* [*список_имен_атрибутов*]

бинарное_выражение ::=
    проекция бинарная_операция реляционное_выражение

бинарная_операция :: =
    UNION | **INTERSECT** | **MINUS** | **TIMES** | **JOIN** | **DIVIDEBY**

По приведенной грамматике можно сделать следующие замечания.

1) Реляционные операторы UNION, INTERSECT и MINUS требуют, чтобы отношения были совместимыми по типу, т. е. имели идентичные заголовки.
2) Реляционные операторы UNION, INTERSECT, TIMES и JOIN ассоциативны и коммутативны.
3) Если отношения A и B не имеют общих атрибутов, то операция соединения A JOIN B эквивалентна операции A TIMES B, т. е. в таком случае соединение вырождается в декартово произведение. Такое соединение называют естественным.
4) Другой допустимый синтаксис для синтаксической категории переименования таков: (терм RENAME список-переименований). Здесь каждый из элементов списка переименований представляет собой выражение имя_атрибута AS имя_атрибута.

5) Несмотря на большие возможности, предоставляемые операторами реляционной алгебры, существует несколько типов запросов, которые нельзя выразить этими средствами. Для таких случаев необходимо использовать процедурные расширения реляционных языков.

В алгебре Кодда не все операторы являются независимыми, т. е. некоторые из реляционных операторов могут быть выражены через другие реляционные операторы.

Оператор естественного соединения по атрибуту **Y** определяется через оператор декартового произведения и оператор ограничения:

A JOIN B = ((A TIMES (B RENAME Y AS Y1)) WHERE Y=Y1)[X, Y, Z]

Оператор пересечения выражается через вычитание следующим образом:

A INTERSECT B = A MINUS (A MINUS B)

Оператор деления выражается через операторы вычитания, декартового произведения и проекции следующим образом:

A DIVIDEBY B = A[X] MINUS ((A[X] TIMES B) MINUS A)[X]

Оставшиеся реляционные операторы (объединение, вычитание, декартово произведение, ограничение, проекция) являются примитивными операторами – их нельзя выразить друг через друга.

В качестве примера рассмотрим запросы на языке реляционной алгебры для схемы базы данных «Поставщики и детали», представленной следующими схемами отношений:

S(Sno: integer, Sname: string, Status: integer, City: string)
P(Pno: integer, Pname: string, Color: string, Weight: real, City: string)
SP(Sno: integer, Pno: integer, Qty: integer)

В данном примере имена доменов представлены именами типов, имена типов отделяются от имен атрибутов двоеточием, первичные ключи выделены подчеркиванием, а имена внешних ключей схемы отношения SP (ПОСТАВКА) совпадают с именами первичных ключей схем отношений S (ПОСТАВЩИК) и P (ДЕТАЛЬ).

1) Получить имена поставщиков, которые поставляют деталь под номером 2.

((SP **JOIN** S) **WHEPE** Pno = 2) [Sname]

2) Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь.

(((P **WHERE** Color = 'Красный') **JOIN** SP) [Sno] **JOIN** S) [Sname]

Другая формулировка того же запроса:

(((P **WHERE** Color = 'Красный') [Pno] **JOIN** SP) **JOIN** S) [Sname]

Этот пример подчеркивает одно важное обстоятельство: возможность сформулировать один и тот же запрос несколькими способами.

3) Получить имена поставщиков, которые поставляют все детали.

((SP [Sno, Pno] **DIVIDE BY** P [Pno] **JOIN** S) [Sname]

4) Получить номера поставщиков, поставляющих по крайней мере все те детали, которые поставляет поставщик под номером 2.

SP [Sno, Pno] **DIVIDE BY** (SP **WHEPE** Sno = 2) [Pno]

5) Получить все пары номеров поставщиков, размещенных в одном городе

(((S **RENAME** Sno **AS** FirstSno) [FirstSno, City] **JOIN**
(S **RENAME** Sno **AS** SecondSno) [SecondSno , City])

WHEPE FirstSno < SecondSno) [FirstSno, SecondSno]

6) Получить имена поставщиков, которые не поставляют деталь под номером 2.

((S[Sno] **MINUS** (SP **WHEPE** Pno = 2) [Sno]) **JOIN** S) [Sname]

Вычислительные возможности реляционной алгебры можно увеличить путем введения дополнительных операторов.

Дополнительные операторы реляционной алгебры

Многие авторы предлагали новые алгебраические операторы после определения Коддом первоначальных восьми. Рассмотрим несколько таких операторов:

SEMIJOIN (полусоединение),
SEMIMINUS (полувычитание),
EXTEND (расширение),
SUMMARIZE (обобщение) и
TCLOSE (транзитивное замыкание).
TRANSFORM
ROLLUP
CUBE

Синтаксис этих операторов выглядит следующим образом.

<полусоединение> ::= <реляционное выражение> **SEMIJOIN** <реляционное выражение>
< полувычитание > ::= <реляционное выражение> **SEMIMINUS** <реляционное выражение>
< расширение> ::= **EXTEND** <реляционное выражение> **ADD**
<список добавляемых расширений>
<добавляемое расширение> ::= <выражение> **AS** <имя атрибута>
<обобщение> ::= **SUMMARIZE** <реляционное выражение> **PER** <реляционное выражение>
ADD <список добавляемых обобщений>
<добавляемое обобщение> ::= <тип обобщения> [(<скалярное выражение>)] AS <имя атрибута>
<тип обобщения> ::= **COUNT** | **SUM** | **AVG** | **MAX** | **MIN** | **ALL** | **ANY** | **COUNTD | SUMD** | **AVGD**
<транзитивное замыкание> ::= **TCLOSE** <реляционное выражение>

Операция расширения

С помощью **операции расширения** из определенного отношения (по крайней мере, концептуально) создается новое отношение, которое содержит дополнительный атрибут, значения которого получены посредством некоторых скалярных вычислений. Примеры.

EXTEND S **ADD** 'Supplier' **AS** TAG
EXTEND P **ADD** (Weight * 454) **AS** GMWT
(**EXTEND** P **ADD** (Weight * 454) **AS** GMWT) **WHERE** GMWT > Weight (10000.0)) { **ALL BUT** GMWT }
EXTEND (P **JOIN** SP) **ADD** (Weight * Qty) **AS** SHIPWT
(**EXTEND** S **ADD** City **AS** SCity) { ALL BUT City }
EXTEND P **ADD** Weight * 454 AS GMW, Weight * 16 AS OZWT)
EXTEND S **ADD** COUNT ((SP **RENAME** SNo AS X) **WHERE** X = SNo) AS NP

Рассмотрим вкратце обобщающие функции. Общее назначение этих функций состоит в том, чтобы на основе значений некоторого атрибута определенного отношения получить скалярное значение. В языке Tutorial D параметр <вызов обобщающей функции> является особым случаем параметра <скалярное выражение> и в общем случае имеет следующий вид.
<имя функции> (<реляционное выражение> [, <имя атрибута>])
Если параметр <имя функции> имеет значение COUNT, то параметр <имя атрибута> недопустим и должен быть опущен. В остальных случаях параметр <имя атрибута> может быть опущен тогда и только тогда, когда параметр <реляционное выражение> задает отношение со степенью, равной единице.

Операция обобщения

В реляционной алгебре операция расширения позволяет выполнять "горизонтальные" вычисления в отношении отдельных строк. **Оператор обобщения** выполняет аналогичную функцию для "вертикальных" вычислений в отношении отдельного столбца. Примеры.

SUMMARIZE SP **PER** SP { PNo } **ADD** SUM (Qty) **AS** TOTQTY

В результате его вычисления создается отношение с заголовком {P#, TOTQT Y}, содержащее один кортеж для каждого значения атрибута P# в проекции SP{P#}. Каждый из этих кортежей содержит значение атрибута P# и соответствующее общее количество деталей. Другими словами, концептуально исходное отношение P «перегруппировано» в множество групп кортежей (по одной группе для каждого уникального значения атрибута P#), после чего для каждой полученной группы сгенерирован один кортеж, помещаемый в окончательный результат.

В общем случае значение выражения

SUMMARIZE A **PER** B **ADD** <*обобщение*> AS

Определяется следующим образом

1. Отношение B должно иметь такой же тип, как и некоторая проекция отношения A, T.e. каждый атрибут отношения B должен одновременно присутствовать в отношении A. Примем, что атрибутами этой проекции (или, что эквивалентно, атрибутами отношения B) являются атрибуты A1, A2, … , An.
2. Результатом вычисления данного выражения будет отношение с заголовком {Al, A2, ... , An, Z}, где Z является новым добавленным атрибутом.
3. Тело результата содержит все кортежи t, где t является кортежем отношения B, расширенным значением нового атрибута Z. Это значение нового атрибута Z подсчитывается посредством вычисления *обобщающего выражения* по всем кортежам отношения A, которое имеет те же значения для атрибутов A1, A2, … , An, что и кортеж t. (Разумеется, если в отношении A нет кортежей, принимающих те же значения для атрибутов A1, A2, .. . , An, что и кортеж t, то *обобщающее выражение* будет вычислено для пустого множества.) Отношение B не должно содержать атрибут с именем Z, а *обобщающее выражение* не должно ссылаться на атрибут Z. Заметьте, что кардинальность результата равна кардинальности отношения B, а степень результата равна степени отношения B плюс единица. Типом переменной Z в этом случае будет тип *обобщающего выражения.*

Вот еще один пример.

SUMMARIZE (P **JOIN** SP) **PER** P { City } **ADD** COUNT **AS** NSP

Легко заметить, что оператор SUMMARIZE не примитивен – его можно моделировать с помощью оператора EXTEND. Рассмотрим следующее выражение

SUMMARIZE SP **PER** S { SNo } **ADD** COUNT AS NP

По сути, это сокращенная запись представленного ниже более сложного выражения

{ **EXTEND** S { Sno } **ADD** ((SP **RENAME** Sno AS X) **WHERE** X=Sno) **AS** Y, COUNT (Y) **AS** NP) { Sno, NP }

Группирование и разгруппирование

Поскольку значениями атрибутов отношений могут быть другие отношения, было бы желательным наличие дополнительных реляционных операторов, называемых операторами группирования и разгруппирования. Рассмотрим пример

SP **GROUP** (PNo, Qty) **AS** PQ

который можно прочесть как «сгруппировать отношение SP по атрибуту SNo», поскольку атрибут SNo является единственным атрибутом отношения SP, не упомянутым в предложении GROUP. В результате получится отношение, заголовок которого выглядит так

{ SNo SNo, PQ **RELATION** { PNo PNo, Qty Qty } }

Другими словами, он состоит из атрибута PQ, принимающего в качестве значений отношения (PQ, в свою очередь, имеет атрибуты PNo и Qty), а также из всех остальных атрибутов отношения SP (в нашем случае "все остальные атрибуты" - это атрибут SNo). Тело этого отношения содержит ровно по одному кортежу для всех различных значений атрибута SNo исходного отношения SP.

Перейдем теперь к операций разгруппирования. Пусть SPQ - это отношение, полученное в результате группирования. Тогда выражение

SPQ **UNGROUP** PQ

Возвращает нас к отношению SP (как и следовало ожидать). Точнее, оно выдает в качестве результата отношение, заголовок которого выглядит так

(SNo SNo, PNo PNo, Qty Qty)

Реляционные сравнения

Реляционная алгебра в том виде, в котором она была изначально определена, не поддерживает прямого сравнения двух отношений (например, проверки их равенства или того, является ли одно из них подмножеством другого). Это упущение легко исправляется следующим образом. Сначала определяется новый вид условия - реляционное сравнение - со следующим синтаксисом.

<реляционное выражение> <отношение> <реляционное выражение>
<отношение> ::=
> -- Собственное супермножество
| >= -- Супермножество
| < -- Собственное подмножество
| <= -- Подмножество
| = -- Равно
| <> -- Не равно

Здесь параметр <реляционное выражение> в обоих случаях выражения реляционной алгебры, представляющие совместимые по типу отношения. Примеры.

S (City) = P (City)

Смысл выражения: совпадает ли проекция отношения поставщиков S по атрибуту City с проекцией отношения деталей P по атрибуту City?

S (SNo) > SPJ (SNo)

Смысл выражения: есть ли поставщики, вообще не поставляющие деталей?
На практике часто требуется определить, является ли данное отношение пустым. Соответствующий оператор, возвращающий логическое значение имеет вид

IS_EMPTY (<реляционное выражение>)
Не менее часто требуется проверить, присутствует ли данный кортеж t в данном отношении R. Для этой цели подойдет следующее реляционное сравнение.

RELATION { t } <= R

Однако, с точки зрения пользователя, удобнее применять следующее сокращение

t **IN** R

Реляционное исчисление

Реляционное исчисление основано на разделе математической логики, который называется исчислением предикатов. Реляционное исчисление существует в двух формах: *исчисление кортежей* и *исчисление доменов*. Основное различие

между ними состоит в том, что переменные исчисления кортежей являются переменными кортежей (они изменяются на отношении, а их значения являются кортежами), в то время как переменные исчисления доменов являются переменными доменов (они изменяются на доменах, а их значения являются скалярами).

Исчисление кортежей

Реляционное исчисление является альтернативой реляционной алгебре. Внешне два подхода очень отличаются – исчисление описательное, а алгебра предписывающая, но на более низком уровне они представляют собой одно и то же, поскольку любые выражения исчисления могут быть преобразованы в семантически эквивалентные выражения в алгебре и наоборот.

Исчисление существует в двух формах: исчисление кортежей и исчисление доменов. Основное различие между ними состоит в том, что переменные исчисления кортежей являются переменными кортежей (они изменяются на отношении, а их значения являются кортежами), в то время как переменные исчисления доменов являются переменными доменов (они изменяются на доменах, а их значения являются скалярами; в этом смысле, действительно, "переменная домена" - не очень точный термин).

Выражение исчисления кортежей содержит заключенный в скобки список целевых элементов и выражение WHERE, содержащее формулу WFF ("правильно построенную формулу"). Такая формула WFF составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, литералов, операторов сравнения, логических (булевых) операторов и скобок. Каждая свободная переменная, которая встречается в формуле WFF, должна быть также перечислена в списке целевых элементов.

Синтаксис исчисления кортежей

Упрощенный синтаксис выражений исчисления кортежей в форме БНФ имеет вид.

объявление-кортежной-переменной ::=
RANGE OF *переменная* **IS** *список-областей*

область ::=
отношение |
реляционное-выражение

реляционное-выражение ::=
(*список-целевых-элементов*)[**WHERE** *wff*]

целевой-элемент ::=
переменная |
переменная.атрибут [**AS** *атрибут*]

wff ::=
условие |
NOT *wff* |
условие **AND** *wff* |
условие **OR** *wff* |
IF *условие* **THEN** *wff* |
EXISTS *переменная* (*wff*) |
FORALL *переменная* (*wff*) |
(*wff*)

условие ::=
(*wff*) |
компаранд операция-отношения компаранд

По приведенной грамматике можно сделать следующие замечания.
1) Квадратные скобки здесь указывают на компоненты, которые по умолчанию могут быть опущены.
2) Категории *отношение, атрибут* и *переменная* – это идентификаторы (т. е. имена).
3) Реляционное выражение содержит заключенный в скобки список целевых элементов и выражение **WHERE,** содержащее формулу *wff* («правильно построенную формулу»). Такая формула *wff* составляется из кванторов (**EXISTS** и **FORALL**), свободных и связанных переменных, констант, операторов сравнения, логических (булевых)

операторов и скобок. Каждая свободная переменная, которая встречается в формуле *wff*, должна быть также перечислена в списке целевых элементов.

4) Категория *условие* представляет или формулу *wff*, заключенную в скобки, или простое скалярное сравнение, где каждый *компаранд* оператора сравнения – это либо скалярная константа, либо значение атрибута в форме *переменная.атрибут*.

Пусть кортежная переменная T определяются следующим образом:

RANGE OF T **IS** R1, R2, ..., Rn

Тогда отношения R1, R2, ..., Rn должны быть совместимы по типу т. е. они должны иметь идентичные заголовки, и кортежная переменная T изменяется на объединении этих отношений, т. е. её значение в любое заданное время будет некоторым текущим кортежем, по крайней мере одного из этих отношений.

Примеры объявлений кортежных переменных.

RANGE OF SX **IS** S
RANGE OF SPX **IS** SP
RANGE OF SY **IS**
(SX) **WHERE** SX.City = 'Смоленск',
(SX) **WHERE EXISTS** SPX (SPX.Sno = SX.Sno **AND** SPX.Pno = 1)

Здесь переменная кортежа SY может принимать значения из множества кортежей S для поставщиков, которые или размещены в Смоленске, или поставляют деталь под номером 1, или и то и другое.

Для сравнения с реляционной алгеброй рассмотрим некоторые запросы на языке исчисления кортежей, которые соответствуют рассмотренным ранее.

Для сравнения с реляционной алгеброй некоторые примеры соответствуют рассмотренным ранее. Любые примеры можно расширить, включив в них завершающий шаг *присвоения*, присвоив значение выражения некоторому именованному отношению; этот шаг для краткости опускается.

1) Получить номера поставщиков из Смоленска со статусом больше 20

(SX.Sno) **WHERE** SX.City = 'Смоленск' **AND** SX.Status > 20

2) Получить все такие пары номеров поставщиков, что два поставщика размещаются в одном городе

(SX.Sno **AS** FirstSno, SY.Sno **AS** SecondSno) **WHERE** SX.City = SY.City **AND** SX.Sno < SY.Sno

Замечание. Спецификации «**AS FirstSno**» и «**AS SecondSno**» дают имена атрибутам *результата;* следовательно, такие имена недоступны для использования во фразе **WHERE** и потому второе сравнение во фразе **WHERE** «**SX.Sno < SY.Sno**», а не «**FirstSno < SecondSno**».

3) Получить имена поставщиков, которые поставляют деталь с номером 2

SX.Sname **WHERE EXISTS** SPX (SPX.Sno = SX.Sno **AND** SPX.Pno = 2)

4) Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь

SX.Sname **WHERE EXISTS** SPX (SX.Sno = SPX.Sno **AND EXISTS** PX (PX.Pno = SPX.Pno **AND** PX.Color = 'Красный'))

Или эквивалентная формула (но в предваренной нормальной форме, в которой все кванторы записываются в начале формулы WFF):

SX.Sname **WHERE EXISTS** SPX (**EXISTS** PX (SX.Sno = SPX.Sno **AND** SPX. Pno = PX.Pno AND PX.Color = 'Красный'))

Предваренная нормальная форма не является более или менее правильной по сравнению с другими формами, но немного попрактиковавшись можно убедиться, что в большинстве случаев это наиболее естественная формулировка запросов. Кроме того, эта форма позволяет уменьшить количество скобок, как показано ниже.
Формулу WFF

квантор_1 переменная_1 (квантор_*2 переменная_2* (*wff*))

(где каждый из кванторов *квантор_1* и *квантор_2* представляет или квантор **EXISTS**, или квантор **FORALL**, *переменная_1* и *переменная_2* - имена переменных) по умолчанию можно однозначно сократить к виду

квантор_1 переменная_1 квантор_*2 переменная_2* (*wff*)

Таким образом, приводимое выше выражение исчисления можно переписать (при желании) следующим образом:

SX.Sname **WHERE EXISTS** SPX **EXISTS** PX (SX.Sno = SPX.Sno **AND** SPX.Pno = PX.Pno **AND** PX.Color = 'Красный')

Однако для ясности мы будем продолжать показывать все скобки во всех остальных примерах.

5) Получить имена поставщиков, которые поставляют по крайней мере одну деталь, поставляемую поставщиком под номером 2

SX.Sname **WHERE EXISTS** SPX (**EXISTS** SPY (SX.Sno = SPX.Sno **AND** SPX.Pno = SPX.Pno **AND** SPY.sno = 2))

6) Получить имена поставщиков, которые поставляют все детали

SX.SNAME **WHERE FORALL** PX (**EXISTS** SPX (SPX.Sno = SX.Sno **AND** SPX.Pno = PX.Pno))

Или равносильное выражение без использования квантора **FORALL**:

SX. SNAME **WHERE NOT EXISTS** PX (**NOT EXISTS** SPX (SPX.Sno = SX.Sno **AND** SPX.Pno = PX.Pno))

Отметим, что содержание этого запроса – «имена поставщиков, которые поставляют все детали» - является точным на 100% (в отличие от случая с алгебраическим аналогом при использовании оператора **DIVIDEBY**).

7) Получить имена поставщиков, которые не поставляют деталь под номером 2

SX.Sname **WHERE NOT EXISTS** SPX (SPX.Sno = SX.Sno **AND** SPX.Pno = 2)

Обратите внимание, как просто это решение можно получить из примера 3.

8) Получить номера поставщиков, которые поставляют по крайней мере все детали, поставляемые поставщиком с номером 2

SX.Sno **WHERE FORALL** SPY (SPY.Sno <> 2 **OR EXISTS** SPZ (SPZ.Sno = SX.Sno **AND** SPZ.Pno = SPY.Pno))

Переформулируем запрос в соответствии с приведенным в выражении: «Получить номера поставщиков, скажем **SX**, таких, что для всех поставок **SPY** поставка осуществляется либо не от поставщика с номером **2**, либо если от поставщика с номером **2**, то существует поставка **SPZ** детали **SPY** от поставщика **SX**».
Введем другое синтаксическое соглашение, чтобы облегчить формулирование таких сложных запросов, как этот, а именно: явную синтаксическую форму для оператора логической импликации.

Если *f* и *g* - формулы WFF, то выражение логической импликации
IF *f* **THEN** *g*
также будет формулой WFF с семантикой, идентичной семантике формулы
(NOT *f* **) OR** *g*

Таким образом, выражение, приведенное выше, может быть переписано (при желании) так:

SX.Sno **WHERE FORALL** SPY (**IF** SPY.Sno = 2 **THEN EXISTS** SPZ (SPZ.Sno = SX.Sno **AND** SPZ.Pno = SPY.Pno))

Дадим словесную формулировку: "Получить номера поставщиков, скажем **SX**, таких, что для всех поставок **SPY** в случае поставки от поставщика с номером **2** существует поставка **SPZ** детали **SPY** от поставщика **SX**».

9) Получить номера деталей, которые или весят более 16 ед., или поставляются поставщиком с номером 2, или и то и другое

RANGE OF PU IS PX.Pno **WHERE** PX.Weight > 16, SPX.Pno **WHERE** SPX.Sno = 2;
PU.Pno

В реляционном алгебраическом эквиваленте здесь могло бы использоваться явное объединение. Альтернативная формулировку этого запроса имеет вид.

PX.Pno **WHERE** PX.Weight > 16 **OR EXIST**S SPX (SPX.Pno = PX.Pno **AND** SPX.Sno = 2)

Однако тот факт, что каждый номер детали в отношении **SP** присутствует и в отношении **P**, делает эту вторую формулировку не в «стиле объединения».

Вычислительные возможности исчисления кортежей

Добавить вычислительные возможности в исчисление довольно просто: необходимо расширить определение компарандов и целевых элементов так, чтобы они включали новую категорию - скалярные выражения, в которых операнды, в свою очередь, могут включать литералы, ссылки на атрибуты и (или) ссылки на итоговые функции. Для целевых элементов также требуется использовать спецификацию вида "**AS** *attribute*" для того, чтобы дать подходящее имя результирующему атрибуту, если нет очевидного наследуемого имени.

Т. к. что смысл скалярного выражения легко воспринимается, подробности опускаются. Однако синтаксис для ссылок на итоговые функции будет показан:

aggregate_function **(** *expression* [**,** *attribute*] **)**

где

aggregate_function - это **COUNT, SUM, AVG, MAX** или **MIN** (возможны, конечно, и некоторые другие функции),
expression - это выражение исчисления кортежей (вычисляющее отношение),
attribute - это такой атрибут результирующего отношения, по которому подсчитывается итог.

Для функции **COUNT** аргумент *attribute* не нужен и опускается; для других итоговых функций его можно опустить по умолчанию, если и только если вычисление аргумента *expression* дает отношение степени один и в таком случае единственный атрибут результата вычисления выражения *expression* подразумевается по умолчанию. Обратите внимание, что ссылка на итоговую функцию возвращает скалярное значение и поэтому допустима в качестве операнда скалярного выражения.

Из сказанного можно сделать следующие выводы:

1. Итоговая функция действует в некоторых отношениях как новый тип квантора. В частности, если аргумент *expression* в данной ссылке на итоговую функцию представляется в виде "**(** *tic* **) WHERE** *f*", где *tic*- целевой элемент списка (*target_item_commalist*), а *f* - это формула WFF, и если экземпляр переменной кортежа *T* свободен в формуле *f*, то такой экземпляр переменной *T* связан в ссылке на итоговую функцию

aggregate_function **((** *tic* **) WHERE** *f* [**,** *attribute*] **)**

2. Пользователи, владеющие языком SQL, могут заметить, что с помощью двух следующих аргументов в ссылке на итоговую функцию, *expression* и *attribute*, можно избежать необходимых для SQL специальных приемов

использования оператора **DISTINCT** для исключения дублирующих кортежей, если это требуется, перед выполнением итоговой операции. Вычисление аргумента *expression* дает отношение, из которого повторяющиеся кортежи всегда исключаются по определению. Аргумент *attribute* обозначает атрибут такого отношения, по которому выполняются итоговые вычисления, а дублирующие значения перед подсчетом итога из такого атрибута не удаляются. Конечно, атрибут может не содержать никаких дублирующих значений в любом случае, в частности, если такой атрибут является первичным ключом.

Примеры

Чтобы облегчить сравнение с реляционной алгеброй, некоторые примеры соответствуют рассмотренным ранее.

1) Получить номера деталей и их вес всех типов деталей, вес которых превышает 10 ед.

(PX.Pno, PX.Weight **AS** GMWT) **WHERE** PX.Weight > 10

Обратите внимание, что спецификация **AS GMWT** дает имя соответствующему атрибуту результата. Поэтому такое имя недоступно для использования во фразе **WHERE** и потому выражение **PX.Weight** записывается в двух местах.

2) Получить всех поставщиков, добавив для каждого литеральное значение "Поставщик"

(SX, 'Поставщик' **AS** TAG)

3) Получить каждую поставку с полными данными о входящих в нее деталях и общим весом поставки

(SPX.Sno, SPX.Qty, PX, PX.Weight * SPX.Qty **AS** SHIPWT) **WHERE** PX.Pno = SPX.Pno

4) Для каждой детали получить ее номер и общее поставляемое количество

(PX.Pno, SUM (SPX **WHERE** SPX.Pno = PX.Pno, Qty) **AS** TOTQTY)

Заметьте, что вместо фразы вида "**BY (Pno)**" ссылка на итоговую функцию включает в качестве' первого аргумента выражение исчисления; вычисление этого выражения (дляданной детали) дает точное множество кортежей, по которому подсчитывается итог. Также стоит отметить, что результат будет включать один кортеж для каждой детали, даже для деталей, которые в данное время не поставляются никакими поставщиками (количество для каждой такой детали будет равно нулю).

5) Получить общее количество поставляемых деталей

(**SUM** (SPX, Qty) **AS** GRANDTOTAL))

в отличие от своего эквивалента - оператора **SUMMARIZE**, это выражение возвращает корректный результат (а именно нуль), если отношение **SP** пустое.

6) Для каждого поставщика получить его номер и общее количество поставляемых им деталей

(SX.Sno, **COUNT** (SPX **WHERE** SPX.Sno = SX.Sno) **AS** NUMBER_OF_PARTS)

В отличие от своего эквивалента - оператора **SUMMARIZE**, это выражение дает результат, который включает кортеж для поставщика с номером 5 (с нулевым количеством).

7) Получить города, в которых хранится а) более пяти красных деталей; б) не более пяти красных деталей

PX.City **WHERE COUNT** (PY **WHERE** PY.City = PX.City **AND** PY.Color = 'Красный') > 5

Это выражение представляет собой решение задачи для части (а). В отличие от своего эквивалента - оператора **SUMMARIZE**, это решение может быть преобразовано в решение для части (б) простой заменой знака ">" на "<=".

Исчисление доменов

Реляционное исчисление, ориентированное на домены (или исчисление доменов), отличается от исчисления кортежей тем, что в нем используются переменные доменов вместо переменных кортежей, т. е. переменные, принимающие свои значения в пределах домена, а не отношения. (Замечание. "Переменную домена" было бы лучше назвать "скалярной переменной", так как ее значения - это *элементы* домена, т.е. скаляры, а не *сами* домены.)
С практической точки зрения большинство очевидных различий между версиями исчисления для доменов и кортежей основано на том, что версия для доменов поддерживает дополнительную форму условия, которое мы будем называть **условием принадлежности**. В общем виде условие принадлежности можно записать так:

R (pair, pair, ...),

где **R** - это имя отношения, а каждая пара **pair** имеет вид **A : v** (где **A** - атрибут отношения **R**, а **v** - или переменная домена, или литерал). Проверка условия дает значение истина, если и только если существует кортеж в отношении **R,** имеющий определенные значения для определенных атрибутов. Например, вычисление выражения

SP (Sno : 1, Pno : 1)

дает значение истина, если и только если в отношении **SP** существует кортеж со значением **Sno**, равным **1**, и значением **Pno**, равным **1**. Аналогично, условие принадлежности

SP (Sno : SX, Pno : PX)

принимает значение истина, если и только если в отношении **SP** существует кортеж со значением **Sno,** эквивалентным текущему значению переменной домена **SX** (какому бы то ни было), и значением **Pno**, эквивалентным текущему значению переменной домена **PX** (опять же какому бы то ни было).

Далее будем подразумевать существование переменных доменов с именами: образуемыми добавлением букв **X, Y, Z, ...** к соответствующим именам доменов. Напомним, что в базе данных поставщиков и деталей каждый атрибут имеет такое же имя, как и соответствующий ему домен, за исключением атрибутов **Sname** и **Pname**, для которых соответствующий домен называется просто **Name.**

Примеры выражений исчисления доменов:

1. (SX)
2. (SX) **WHERE** S (Sno : SX)
3. (SX) **WHERE** S (Sno : SX, City : 'Смоленск')
4. (SX, CityX) **WHERE** S (Sno : SX, City : CityX) **AND** SP (Sno : SX, Pno : 2)
5. (SX, PX) **WHERE** S (Sno : SX, City : CityX) **AND** P (Pno : PX, City : CityY) **AND** CityX <> CityY

Семантика выражений запросов:
1. Множество всех номеров поставщиков.
2. Множество всех номеров поставщиков отношения **S**.
3. Подмножество номеров поставщиков из города **Смоленск**.
4. Получить номера и города поставщиков, поставляющих деталь под номером 2. (Обратите внимание, что для этого запроса, выраженного в терминах исчисления кортежей, требовался квантор существования; заметьте также, что это первый из приведенных здесь примеров, где действительно необходимы скобки для списка целевых элементов).
5. Получить такие пары ‘номер поставщика’ – ‘номер детали’, что поставщики и детали размещены в одном городе".

Примеры

Для сравнения с реляционной алгеброй некоторые примеры соответствуют рассмотренным ранее.

1) Получить номера поставщиков из Смоленска со статусом, большим 20

SX **WHERE EXISTS** StatusX (StatusX > 20 **AND** S (Sno : SX, Status : StatusX, City : 'Смоленск'))

Обратите внимание, что кванторы все еще требуются. Этот пример несколько неуклюжий по сравнению с его аналогом, выраженным в терминах исчисления кортежей. С другой стороны, конечно, есть случаи, когда верно обратное; смотрите, в частности, более сложные примеры, приведенные ниже.

2) Получить все такие пары номеров поставщиков, что два поставщика размещаются в одном городе

(SX **AS** FirstSno, SY **AS** SecondSno) **WHERE EXSIST** CityZ (S (Sno : SX, City : CityZ) **AND** S (Sno : SY, City : CityZ) **AND** SX < SY)

3) Получить имена поставщиков, которые поставляют по крайней мере одну красную деталь

NAMEX **WHERE EXISTS** SX **EXISTS** PX (S (Sno : SX, Sname : NameX) **AND** SP (Sno : SX, Pno : PX) **AND** P (Pno : PX, Color : 'Красный'))

4) Получить имена поставщиков, которые поставляют по крайней мере одну деталь, поставляемую поставщиком с номером 2

NameX **WHERE EXISTS** SX **EXISTS** PX (S (Sno : SX, Sname : NameX) **AND** SP (Sno : SX, Pno : PX) **AND** SP (Sno : 2, Pno : PX))

5) Получить имена поставщиков, которые поставляют все типы деталей

NameX **WHERE EXISTS** SX (S (Sno : SX, Sname : NameX) **AND FORALL** PX (**IF** P (Pno : PX) **THEN** SP (Sno : SX, Pno : PX)))

6) Получить имена поставщиков, которые не поставляют деталь с номером 2

NameX **WHERE EXISTS** SX (S (Sno : SX, Sname : NameX) **AND NOT** SP (Sno : SX, Pno : 2))

7) Получить номера поставщиков, которые поставляют по крайней мере все типы деталей, поставляемых поставщиком с номером 2

SX **WHERE FORALL** PX (**IF** SP (Sno : 2, Pno : PX) **THEN** SP (Sno : SX, Pno : PX))

8) Получить номера деталей, которые или весят более 16 фунтов, или поставляются поставщиком с номером 2, или и то, и другое

PX **WHERE EXISTS** WeightX (P (Pno : PX, Weight : WeightX) **AND** WeightX > 16) **OR** SP (Sno : 2, Pno : PX)

Исчисление доменов, как и исчисление кортежей, формально эквивалентно реляционной алгебре (т.е. оно реляционно полно). Для доказательства можно сослаться, например, на работы Ульмана (Ullman).

Реляционное исчисление и реляционная алгебра

Ранее утверждалось, что реляционная алгебра и реляционное исчисление в своей основе эквивалентны. Обсудим это утверждение более подробно. Вначале Кодд показал в своей статье, что алгебра, по крайней мере, мощнее исчисления. (Термин "исчисление" будет использоваться для обозначения исчисления кортежей.) Он сделал это, придумав алгоритм, называемый "алгоритмом редукции Кодда", с помощью которого любое выражение исчисления можно преобразовать в семантически эквивалентное выражение алгебры. Мы не станем приводить здесь полностью этот алгоритм, а ограничимся примером, иллюстрирующим в общих чертах, как этот алгоритм функционирует.

В качестве основы для нашего примера используется база данных поставщиков, деталей и проектов. Для удобства приводится набор примерных значений для этой базы данных.

Таблица Поставщики (S)

Sno	Sname	Status	City
1	Алмаз	20	Смоленск
2	Циклон	10	Владимир
3	Дельта	30	Владимир
4	Орион	20	Смоленск
5	Аргон	30	Ярославль

Таблица Детали (P)

Pno	Pname	Color	Weight	City
1	Гайка	Красный	12	Смоленск
2	Болт	Зеленый	17	Владимир
3	Винт	Синий	17	Рязань
4	Винт	Красный	14	Смоленск
5	Шайба	Синий	12	Владимир
6	Шпунт	Красный	19	Смоленск

Таблица Проекты (J)

Jno	Jname	City
1	Ангара	Владимир
2	Алтай	Рязань
3	Енисей	Ярославль
4	Амур	Ярославль
5	Памир	Смоленск
6	Чегет	Тверь
7	Эльбрус	Смоленск

Таблица Поставки (SPJ)

Sno	Pno	Jno	Qty
1	1	1	200
1	1	4	700
2	3	1	400

2	3	2	200
2	3	3	200
2	3	4	500
2	3	5	600
2	3	6	400
2	3	7	800
2	5	2	100
3	3	1	200
3	4	2	500
4	6	3	300
4	6	7	300
5	2	2	200
5	2	4	100
5	5	5	500
5	5	7	100
5	6	2	200
5	1	4	100
5	3	4	200
5	4	4	800
5	5	4	400
5	6	4	500

База данных поставщиков, деталей и проектов (значения для примера)

Рассмотрим запрос: "Получить имена и города поставщиков, обеспечивающих по крайней мере один проект в городе Ярославль с поставкой по крайней мере 50 штук каждой детали". Выражение исчисления для этого запроса следующее:

(SX.Name, SX.City) **WHERE EXISTS** JX **FORALL** PX **EXISTS** SPJX (JX.City = 'Ярославль' **AND**
JX.Jno = SPJX.Jno **AND** PX.Pno = SPJX.Pno **AND** SX.Sno = SPJX.Sno **AND** SPJX.Qty >= 50)

Здесь **SX, PX, JX** и **SPJX** - переменные кортежей, берущие свои значения из отношений **S, P, J** и **SPJ** соответственно. Теперь покажем, как вычислить это выражение, чтобы добиться необходимого результата.

Шаг 1. Для каждой переменной кортежа выбираем ее область значений (т.е. набор всех значений для этой переменной) если это возможно. Выражение «выбираем, если возможно» подразумевает, что существует условие выборки, встроенное в фразу WHEPE, которую можно использовать, чтобы сразу исключить из рассмотрения некоторые кортежи. В нашем случае выбираются следующие наборы кортежей:

SX : Все кортежи отношения **S** 5 кортежей
PX : Все кортежи отношения **P** 6 кортежей
JX : Кортежи отношения **J**, в которых **City = 'Ярославль'** 2 кортежа
SPJX : Кортежи отношения **SPJ**, в которых **Qty >= 50** 24 кортежа

Шаг 2. Строим декартово произведение диапазонов, выбранных на первом шаге. Получим …
Для экономии места таблица не приводится. Полное произведение содержит 5*6*2*24 = 1440 кортежей.

Шаг 3. Осуществляем выборку из произведения, построенного на втором шаге в соответствии с частью «условие соединения» фразы **WHERE**. В нашем примере эта часть следующая:

JX.Jno = SPJX.Jno **AND** PX.Pno = SPJX.Pno **AND** SX.Sno = SPJX.Sno **AND**

Поэтому из произведения исключаются кортежи, для которых значение **Sno** поставщика не равно значению **Sno** поставки, значение **Pno** детали не равно значению **Pno** поставки, значение **Jno** проекта не равно значению **Jno** поставки, после чего получаем подмножество декартова произведения, состоящее только из 10 кортежей.

Шаг 4. Применяем кванторы справа налево следующим образом:

- Для квантора **«EXISTS RX»** (где **RX** - переменная кортежа, принимающая значение на не котором отношении **R**) проецируем текущий промежуточный результат, чтобы исключить все атрибуты отношения **R.**
- Для квантора **«FORALL RX»** делим текущий промежуточный результат на отношение «выбранной области значений», соответствующее **RX**, которое было получено выше. При выполнении этой операции также будут исключены все атрибуты отношения **R.**

В нашем примере имеем следующие кванторы:

EXISTS JX **FORALL** PX **EXISTS** SPJX

Выполняем соответствующие операции.

1. **EXISTS SPJX**. Проецируем, исключая атрибуты отношения **SPJ (SPJ.Sno, SPJ.Pno, SPJ.Jno и SPJ.Qty**). В результате получаем:

Sno	Sname	Status	City	Pno	Pname	Color	Weight	City	Jno	Jname	City
1	Алмаз	20	Смоленск	1	Гайка	Красный	12	Смоленск	4	Амур	Ярославль
2	Циклон	10	Владимир	3	Винт	Синий	17	Рязань	3	Енисей	Ярославль
2	Циклон	10	Владимир	3	Винт	Синий	17	Рязань	4	Амур	Ярославль
4	Орион	20	Смоленск	6	Шпунт	Красный	19	Смоленск	3	Енисей	Ярославль
5	Аргон	30	Ярославль	2	Болт	Зеленый	17	Владимир			Ярославль
5	Аргон	30	Ярославль	1	Гайка	Красный	12	Смоленск	4	Амур	Ярославль
5	Аргон	30	Ярославль	3	Винт	Синий	17	Рязань	4	Амур	Ярославль
5	Аргон	30	Ярославль	4	Винт	Красный	14	Смоленск	4	Амур	Ярославль
5	Аргон	30	Ярославль	5	Шайба	Синий	12	Владимир	4	Амур	Ярославль
5	Аргон	30	Ярославль	6	Шпунт	Красный	19	Смоленск	4	Амур	Ярославль

2. **FORALL PX.** Делим на отношение **P**. В результате получаем:

Sno	Sname	Status	City	Jno	Jname	City
5	Аргон	30	Ярославль	4	Амур	Ярославль

(Теперь у нас есть место, чтобы показать отношение полностью, без сокращений.)

3. **EXISTS JX**. Проецируем, исключая атрибуты отношения **J (J.Jno, J.name и J.City)**. В результате получаем:

Sno	Sname	Status	City
5	Аргон	30	Ярославль

Шаг 5. Проецируем результат шага 4 в соответствии со спецификациями в целевом списке элементов. В нашем примере целевым элементом списка будет: **SX.Sname, SX.City**. Следовательно, конечный результат таков:

Sname	City
Аргон	Ярославль

Из сказанного выше следует, что начальное выражение исчисления семантически эквивалентно определенному вложенному алгебраическому выражению, а если быть более точным, то проекции от проекции деления проекции выборки из произведения четырех выборок (!). Этим завершаем пример. Конечно, можно намного улучшить алгоритм, хотя многие подробности скрыты в наших пояснениях; но вместе с тем, необходим адекватный пример, предлагающий общую идею.

Теперь можно объяснить одну из причин (и не только одну) определения Коддом ровно восьми алгебраических операторов. Эти восемь операторов обеспечивают соглашение целевого языка, как средства возможной реализации исчисления. Другими словами, для данного языка, такого как QUEL, который основывается на исчислении, одно из возможных применений заключается в том, чтобы можно было брать запрос в том виде, в каком он предоставляется пользователем (являющийся, по существу, просто выражением исчисления), и применять к нему алгоритм получения эквивалентного алгебраического выражения. Это алгебраическое выражение, конечно, содержит множество алгебраических операций, которые согласно определению по своей природе выполнимы. (Следующий шаг состоит в продолжении оптимизации этого алгебраического выражения.)

Также необходимо отметить, что восемь алгебраических операторов Кодда являются мерой оценки выразительной силы любого языка баз данных (существующего или предлагаемого). Обсудим этот вопрос подробнее.

Во-первых, язык называется реляционно полным, если он по своим возможностям, по крайней мере, не уступает реляционному исчислению, т.е. любое отношение, которое можно определить с помощью реляционного исчисления, также можно определить и с помощью некоторого выражения рассматриваемого языка. «Реляционно полный» - значит, не уступающий по возможностям алгебре, а не исчислению, но это то же самое. По сути, из существования алгоритма преобразования Кодда немедленно следует, что реляционная алгебра обладает реляционной полнотой.)

Реляционную полноту можно рассматривать как основную меру возможностей выборки и выразительной силы языков баз данных вообще. В частности, так как исчисление и алгебра - реляционно полные, они могут служить базисом для проектирования языков, не уступающих им по выразительности, без необходимости пересортировки для использования циклов - особенно важное замечание, если язык предназначается конечным пользователям, хотя оно также применимо, если язык предназначается прикладным программистам.

Далее, поскольку алгебра реляционно полная, то, чтобы доказать, что некоторый язык L обладает реляционной полнотой, достаточно показать, что в языке L есть аналоги всех восьми алгебраических операций (на самом деле достаточно по казать, что в нем есть аналоги пяти примитивных операций) и что операндами любой операции языка L могут быть любые выражения этого языка. Язык SQL - это пример языка, реляционную замкнутость которого можно показать таким способом. Язык QUEL – еще один такой пример. В действительности на практике часто проще показать, что в данном языке есть эквиваленты алгебраических операций, чем найти в нем эквиваленты выражений исчисления. Именно поэтому реляционная полнота обычно определяется в терминах алгебраических выражений, а не выражений исчисления.