Общие понятия реляционного подхода к организации БД. Основные концепции и термины

Домен - определяется путем задания некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу этого типа данных (*ограничения домена*). Элемент данных является элементом домена в том и только в том случае, если вычисление этого логического выражения дает результат *истина* (для логических значений мы будем попеременно использовать обозначения *истина* и *ложь* или *true* и *false*). С каждым доменом связывается имя, уникальное среди имен всех доменов соответствующей базы данных.

З*аголовком* (или *схемой*) *отношения* r (**Hr**) называется конечное множество упорядоченных пар вида <A, T>, где **A** называется именем атрибута, а **T** обозначает имя некоторого базового типа или ранее определенного домена. По определению требуется, чтобы все имена атрибутов в заголовке отношения были различны.

Кортежем tr, соответствующим заголовку Hr, называется множество упорядоченных триплетов вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr. Третий элемент – v – триплета <A, T, v> должен являться допустимым значением типа данных или домена T.

Телом Br отношения r называется произвольное множество кортежей tr.

Значением Vr отношения r называется пара множеств Hr и Br.

Переменной VARr называется именованный контейнер, который может содержать любое допустимое значение Vr. Естественно, что при определении любой VARr требуется указывать соответствующий заголовок отношения Hr.

Степенью, или *«арностью»*, заголовка отношения, кортежа, соответствующего этому заголовку, тела отношения, значения отношения и переменной отношения является мощность заголовка отношения.

Схема реляционной базы данных набор пар <имя_VARr, Hr>, включающий имена и заголовки всех переменных отношения, которые определены в базе данных.

Реляционная база данных – это набор пар <VARr, Hr> (конечно, каждая переменная отношения в любой момент времени содержит некоторое значение-отношение, в частности, пустое).

Фундаментальные свойства отношений

Отсутствие кортежей-дубликатов, первичный и возможные ключи отношений. Тело любого отношения никогда не содержит кортежей-дубликатов, следует из определения тела отношения как множества кортежей.

Отсутствие упорядоченности кортежей. Свойство отсутствия упорядоченности кортежей в значении отношения также является следствием определения тела отношения как множества кортежей.

Отсутствие упорядоченности атрибутов. Атрибуты отношений не упорядочены, поскольку по определению заголовок отношения есть множество пар <имя атрибута, имя домена>.

Атомарность значений атрибутов, первая нормальная форма отношения. Значения всех атрибутов являются атомарными (вернее, скалярными). Это следует из определения домена как потенциального множества значений скалярного типа данных, т. е. среди значений домена не могут содержаться значения с видимой структурой, в том числе множества значений (отношения).

Первичным ключом переменной отношения является такое подмножество S множества атрибутов ее заголовка, что в любое время значение первичного ключа (составное, если в состав первичного ключа входит более одного атрибута) в любом кортеже тела отношения отличается от значения первичного ключа в любом другом кортеже тела этого отношения, а никакое собственное подмножество S этим свойством не обладает.

Остальные минимальные наборы атрибутов, обладающие свойством уникальности, называются **возможными ключами.**

Реляционная модель данных: общее понятие и составные части.

Структурная, целостная и манипуляционная часть.

Модель данных (в контексте области баз данных) описывает некий набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели.

Требованием целостности сущности (entity integrity): у любой переменной отношения должен существовать первичный ключ, и никакое значение первичного ключа в кортежах значения-отношения переменной отношения не должно содержать неопределенных значений.

Атрибут такого рода (возможно, составной) называется ***внешним ключом (foreign key),*** поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (т. е. задают значения их первичного ключа).

Требование целостности по ссылкам, или *требование целостности внешнего ключа*, состоит в том, что для каждого значения внешнего ключа, появляющегося в кортеже значения-отношения ссылающейся переменной отношения, либо в значении-отношении переменной отношения, на которую указывает ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть полностью неопределенным (т. е. ни на что не указывать). Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что вообще запрещается производить удаление кортежа, для которого существуют ссылки (т. е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится полностью неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

Реляционная алгебра Кодда

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для реляционных баз данных.

- При выполнении операции *объединения* (UNION) двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов.
- Операция *пересечения* (INTERSECT) двух отношений с одинаковыми заголовками производит отношение, включающее все кортежи, которые входят в оба отношения-операнда.
- Отношение, являющееся *разностью* (MINUS) двух отношений с одинаковыми заголовками, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, которое является вторым операндом.
- При выполнении *декартова произведения* (TIMES) двух отношений, пересечение заголовков которых пусто, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов.
- Результатом *ограничения* (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.
- При выполнении *проекции* (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.
- При *соединении* (JOIN) двух отношений по некоторому условию образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию.
- У операции *реляционного деления* (DIVIDE BY) два операнда – бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.
- Операция *переименования* (RENAME) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.
- Операция *присваивания* (:=) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Алгебра А

Базисом предложенной Крисом Дейтом и Хью Дарвеном **Алгебры А** являются операции реляционного отрицания (дополнения), реляционной конъюнкции (или дизъюнкции) и проекции (удаления атрибута). Реляционные аналоги логических операций определяются в терминах отношений на основе обычных теоретико-множественных операций и позволяют выражать напрямую операции пересечения, декартова произведения, естественного соединения, объединения отношений и т. д. Путем комбинирования базовых операций выражаются операции переименования атрибутов, соединения общего вида, взятия разности отношений. Алгебра А позволяет лучше осознать логические основы реляционной модели, хотя, безусловно, является в меньшей степени ориентированной на практическое применение, чем алгебра Кодда

Во всех формальных спецификациях exists обозначает ***квантор существования***; exists tr означает «существует такой t*r*, что». Символ «$\in$» означает принадлежность одного множества другому; t*r*$\in$B*r* означает, что элемент t*r* принадлежит множеству B*r*. Выражение t*r*$\notin$B*r* означает, что элемент t*r* не принадлежит множеству B*r*. Операции minus и union являются традиционными теоретико-множественными операциями взятия разности и объединения множеств.

Операция реляционного дополнения.

Пусть *s* обозначает результат операции <NOT> *r*. Тогда:

- H*s* = H*r* (заголовок результата совпадает с заголовком операнда);
- B*s* = {t*s* : exists t*r* (t*r* $\notin$B*r* and t*s* = t*r*) } (в тело результата входят все кортежи, соответствующие заголовку и не входящие в тело операнда).

Операция <NOT> производит дополнение *s* заданного отношения *r*. Заголовком *s* является заголовок r. Тело *s* включает все кортежи, соответствующие этому заголовку и не входящие в тело *r*.

Операция удаления атрибута.

- Пусть *s* обозначает результат операции *r* <REMOVE> A. Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип (или домен) T такой, что <A, T> $\in$H*r* (т. е. в состав заголовка отношения *r* должен входить атрибут A). Тогда: H*s* = H*r* minus {<A, T>}, т. е. заголовок результата получается из заголовка операнда изъятием атрибута A;
- B*s* = {t*s* : exists t*r* exists v (t*r* $\in$B*r* and v $\in$T and <A,T,v> $\in$t*r* and t*s* = t*r* minus {<A,T,v>})}, т. е. в тело результата входят все кортежи операнда, из которых удалено значение атрибута A.

Операция <REMOVE> производит отношение *s*, формируемое путем удаления указанного атрибута A из заданного отношения *r*. Операция эквивалентна взятию проекции *r* на все атрибуты, кроме A. Заголовок *s* получается теоретико-множественным вычитанием из заголовка *r* множества из одного элемента {<A, T>}. Тело *s* состоит из таких кортежей, которые соответствуют заголовку *s*, причем каждый из них является подмножеством некоторого кортежа тела отношения *r*.

Операция переименования.

Пусть *s* обозначает результат операции *r* <RENAME> (A, B). Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип T, такой, что <A, T> $\in$Hr, и чтобы не существовал такой тип T, что <B, T> $\in$H*r*. (Другими словами, в схеме отношения *r* должен присутствовать атрибут A и не должен присутствовать атрибут B.) Тогда:

- H*s* = (H*r* minus {<A, T>}) union {<B, T>}, т. е. в схеме результата B заменяет A;
- B*s* = {t*s* : exists t*r* exists v (t*r* $\in$B*r* and v $\in$T and <A, T, v> $\in$t*r* and t*s* = (t*r* minus {<A, T, v>}) union {<B, T, v>})}, т. е. в кортежах тела результата имя значений атрибута A меняется на B.

Операция <RENAME> производит отношение *s*, которое отличается от заданного отношения *r* только именем одного его атрибута, которое изменяется с A на B. Заголовок *s* такой же, как

заголовок r, за исключением того, что пара <B, T> заменяет пару <A, T>. Тело *s* включает все кортежи тела *r*, но в каждом из этих кортежей триплет <B, T, v> заменяет триплет <A, T, v>.

Операция реляционной конъюнкции

Пусть *s* обозначает результат операции r_1 <AND> r_2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1> $\in$ Hr_1 и <A, T2> $\in$ Hr_2, то T1=T2. (Другими словами, если в двух отношениях-операндах имеются одноименные атрибуты, то они должны быть определены на одном и том же типе (домене).) Тогда:

- H*s* = Hr_1 union Hr_2, т. е. заголовок результата получается путем объединения заголовков отношений-операндов, как в операциях TIMES и JOIN из предыдущей лекции;
- B*s* = { t*s* : exists tr_1 exists tr_2 ((tr_1 $\in$ Br_1 and tr_2 $\in$ Br_2) and t*s* = tr_1 union tr_2)}; обратите внимание на то, что кортеж результата определяется как *объединение кортежей операндов*; поэтому:
    - если схемы отношений-операндов имеют непустое пересечение, то операция <AND> работает как естественное соединение;
    - если пересечение схем операндов пусто, то <AND> работает как расширенное декартово произведение;
    - если схемы отношений полностью совпадают, то результатом операции является пересечение двух отношений-операндов.

Операция <AND> является реляционной конъюнкцией, в некоторых случаях выдающей в результате отношение *r*s, ранее называвшееся естественным соединением двух заданных отношений r_1 и r_2. Заголовок *r*s является объединением заголовков r_1 и r_2. Тело *s* включает каждый кортеж, соответствующий заголовку *s* и являющийся надмножеством некоторого кортежа из тела r_1 и некоторого кортежа из тела r_2.

Операция реляционной дизъюнкции

Пусть *s* обозначает результат операции r_1 <OR> r_2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1> $\in$ Hr_1 и <A, T2> $\in$ Hr_2, то должно быть T1 = T2 (одноименные атрибуты должны быть определены на одном и том же типе). Тогда:

- H*s* = Hr_1 union Hr_2 (из схемы результата удаляются атрибуты-дубликаты);
- B*s* = { t*s* : exists tr_1 exists tr_2 ((tr_1 $\in$ Br_1 or tr_2 $\in$ Br_2) and t*s* = tr_1 union tr_2)}; очевидно, что при этом:
    - если у операндов нет общих атрибутов, то в тело результирующего отношения входят все такие кортежи t*s*, которые являются объединением кортежей tr_1 и tr_2, соответствующих заголовкам отношений-операндов, и хотя бы один из этих кортежей принадлежит телу одного из операндов;
    - если у операндов имеются общие атрибуты, то в тело результирующего отношения входят все такие кортежи t*s*, которые являются объединением кортежей tr_1 и tr_2, соответствующих заголовкам отношений-операндов, если хотя бы один из этих кортежей принадлежит телу одного из операндов, и значения общих атрибутов tr_1 и tr_2 совпадают;
    - если же схемы отношений-операндов совпадают, то тело отношения-результата является объединением тел операндов.

Операция <OR> является реляционной дизъюнкцией и обобщением того, что ранее называлось объединением. Заголовок *s* есть объединение заголовков r_1 и r_2. Тело *s* состоит из всех кортежей, соответствующих заголовку *s* и являющихся надмножеством *либо* некоторого кортежа из тела r_1, *либо* некоторого кортежа из тела r_2.

Полнота алгебры А.

Покажем, что Алгебра А является полной, т. е. на основе введенных операций выражаются все операции алгебры Кодда, рассмотренной в предыдущей лекции. К настоящему моменту в состав базовых операций Алгебры А входят операция <REMOVE> в качестве аналога операции PROJECT, а также операция переименования атрибутов <RENAME>. UNION является частным случаем операции <OR>, TIMES, INTERSECT и NATURAL JOIN – частные случаи операции <AND>. Нам осталось показать, что **через операции Алгебры А выражаются операции взятия разности MINUS, ограничения (WHERE), соединения общего вида (JOIN) и реляционного деления (DIVIDE BY).**

Выводимость операции взятия разности-если отношения r_1 и r_2 совместимы по объединению, то r_1 MINUS r_2 = r_1 <AND> <NOT> r_2.

Интерпретация операции ограничения

мы определяли операцию ограничения *r* WHERE comp, где *r* – отношение, а *comp* – простое условие ограничения вида (*a comp-op b*), где *a* и *b* – имена атрибутов ограничиваемого отношения, для которых осмыслена операция сравнения *comp-op*, либо вида (*a comp-op* const), где *a* – имя атрибута ограничиваемого отношения, а const – литерально заданная константа. Операцией сравнения *comp-op* может быть «=», «≠», «>», «<», «≥», «≤». Покажем на нескольких примерах, как можно выразить операцию ограничения с помощью базовых операций Алгебры А для всех простых допустимых условий.

СЛУЖАЩИЕ_1 **WHERE** СЛУ_НОМЕР = РУК_НОМ. Получаем:

СЛУЖАЩИЕ_1 <**AND**> ((((СЛУЖАЩИЕ_1 <**REMOVE**> СЛУ_НОМЕР) <**REMOVE**> СЛУ_ИМЯ) <**REMOVE**> СЛУ_ЗАРП) <**RENAME**> (РУК_НОМ, СЛУ_НОМЕР))

Соединения общего вида

При наличии того факта, что операция взятия расширенного декартова произведения TIMES является частным случаем операции <AND>, после того как мы научились с помощью Алгебры А выполнять ограничения, становится очевидно, что через операции Алгебры А выражаются и соединения общего вида. В общем случае, чтобы получить результат соединения общего вида произвольных отношений А и В, нужно:

- выполнить над одним из отношений одну или несколько операций <RENAME>, чтобы избавиться от общих имен атрибутов;
- выполнить над полученными отношениями операцию <AND>, производящую расширенное декартово произведение;
- и для полученного отношения выполнить одну или несколько операций <AND> с отношениями-константами, чтобы должным образом ограничить его.

Реляционное деление

Пусть имеются отношения r_1{A, B} и r_2{B}. Утверждается, что результат r_1 DIVIDE BY r_2 совпадает с результатом выражения (r_1 PROJECT A) MINUS ((r_2 TIMES (r_1 PROJECT A)) MINUS r_1) PROJECT A) в терминах операций реляционной алгебры Кодда или (r_1 <REMOVE> B) <AND> <NOT> ((r_2 <AND> (r_1 <REMOVE> B)) <AND> <NOT> r_1) <REMOVE> B) в терминах операций Алгебры А.

Действительно, результатом выполнения операции r_1 PROJECT A является унарное отношение со схемой {A}, кортежи тела которого содержат все значения атрибута А из тела отношения r_1. Результат выражения r_2 TIMES (r_1 PROJECT A) – это бинарное отношение со схемой {A, B}, в тело которого входят все возможные комбинации значений атрибута В в теле отношения r_2 и атрибута А в теле отношения r_1. В теле результата вычисления выражения (r_2 TIMES (r_1 PROJECT A)) MINUS r_1 останутся только те кортежи, которые не входят во второй операнд, т. е. кортежи с таким

значением атрибута A, что значение атрибута B, принадлежащее телу r_2, не является значением атрибута B ни в одном кортеже тела отношения r_1. Следовательно, если мы возьмем проекцию результата выражения (r_2 TIMES (r_1 PROJECT A)) MINUS r_1 на атрибут A, то в результирующем унарном отношении останутся только те значения A, которые не должны попасть в результат операции r_1 DIVIDE BY r_2. После выполнения завершающей операции MINUS мы получим желаемый результат.

Избыточность алгебры A

В формальной математической логике стандартным базисом для выражения всех возможных булевских функций является набор {NOT, AND, OR} (отрицание, дизъюнкция и конъюнкция). Известно, что этот набор традиционен, но избыточен, поскольку верны тождества A AND B ≡NOT (NOT A OR NOT B) и A OR B ≡NOT (NOT A AND NOT B). Оказывается, что аналогичные тождества справедливы для операций <NOT>, <AND> и <OR> Алгебры A. Тем самым, в наборе базовых операций Алгебры A можно оставить операции <AND> и <NOT> (или <OR> и <NOT>).

Более того, в алгебре логики существуют две операции, через каждую из которых выражаются все три «базовые» операции: «**штрих Шеффера**» – sh (A, B) ≡NOT A OR NOT B – и «**стрелка Пирса**» – pi (A, B) ≡NOT A AND NOT B.
Легко видеть, что

- sh (A, A) ≡NOT A;
- sh (NOT A, NOT B) ≡A OR B и
- NOT sh (A, B) ≡A AND B.

Аналогично,

- pi (A, A) ≡NOT A;
- pi (NOT A, NOT B) ≡A AND B и
- NOT pi (A, B) ≡A OR B.

Аналогичные тождества справедливы для реляционных вариантов штриха Шеффера (<sh> (r1, r2) ≡<NOT> r1 <OR> <NOT> r2) и стрелки Пирса (<pi> (r1, r2) ≡<NOT> r1 <AND> <NOT> r2). Поэтому можно свести набор операций Алгебры A к трем операциям: <sh> (или <pi>), <RENAME> и <REMOVE>.

Избыточность операции переименования.
Наконец, покажем, что избыточна и операция <RENAME>. Ну там надо тупо добавить к таблице еще один столбец, копию столбца, который собираемся переименовывать, но уже с новым именем. А потом REMOVE столбец со старым названием.

Тем самым, можно сократить набор операций Алгебры A до двух операций: <sh> (или <pi>) и <REMOVE> и операции присваивания переменной отношения.

Реляционное исчисление кортежей

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. В основе исчисления лежит понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

В **исчислении кортежей** областями определения переменных являются тела отношений базы данных, т. е. допустимым значением каждой переменной является кортеж тела некоторого отношения.

Для определения кортежной переменной используется оператор RANGE. Например, для того чтобы определить переменную СЛУЖАЩИЙ, областью определения которой является отношение СЛУЖАЩИЕ, нужно употребить конструкцию

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ

Как уже говорилось, из этого определения следует, что в любой момент времени переменная СЛУЖАЩИЙ представляет некоторый кортеж отношения СЛУЖАЩИЕ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной (это аналогично тому, как, например, при программировании на языке С можно сослаться на значение поля структурной переменной). Например, для того, чтобы сослаться на значение атрибута СЛУ_ИМЯ переменной СЛУЖАЩИЙ, нужно употребить конструкцию СЛУЖАЩИЙ.СЛУ_ИМЯ.

Правильно построенная формула (Well-Formed Formula, **WFF**) служит для выражения условий, накладываемых на кортежные переменные.

Способ реализации, который приводит к получению области истинности рассмотренной формулы, в действительности является наиболее общим (и зачастую неоптимальным) способом выполнения операций соединения (он называется ***методом вложенных циклов*** – *nested loops join*). (два вложенных цикла, по первому переменной и по второй, применяется в случае формулы типа перем1.что-то=перем2.кто-то)

При построении WFF допускается использование **кванторов существования (EXISTS)** и **всеобщности (FORALL)**. Если form – это WFF, в которой участвует переменная var, то конструкции EXISTS var (form) и FORALL var (form) представляют собой WFF. По определению, формула EXISTS var (form) принимает значение true в том и только в том случае, если в области определения переменной var найдется хотя бы одно значение (кортеж), для которого WFF form принимает значение true. Формула FORALL var (form) принимает значение true, если для всех значений переменной var из ее области определения WFF form принимает значение true.

Переменные, входящие в WFF, могут быть свободными или связанными. По определению, все переменные, входящие в WFF, при построении которой не использовались кванторы, являются ***свободными***. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех WFF, построенных с ее участием, var является ***связанной переменной***. Это означает, что такая переменная не видна за пределами минимальной WFF, связавшей эту переменную. При вычислении значения такой WFF используется не одно значение связанной переменной, а вся область ее определения.

Итак, WFF обеспечивают средства формулировки условия выборки из отношений БД. Чтобы можно было использовать исчисление для реальной работы с БД, требуется еще один компонент, который определяет набор и имена атрибутов результирующего отношения. Этот компонент называется *целевым списком (target list)*.

Целевой список строится из целевых элементов, каждый из которых может иметь следующий вид:

- var.attr, где var – имя свободной переменной соответствующей WFF, а attr – имя атрибута отношения, на котором определена переменная var;
- var, что эквивалентно наличию подсписка $var.attr_1$, $var.attr_2$, ..., $var.attr_n$, где {$attr_1$, $attr_2$, ..., $attr_n$} включает имена всех атрибутов определяющего отношения;
- new_name = var.attr; new_name – новое имя соответствующего атрибута результирующего отношения.

Последний вариант требуется в тех случаях, когда в WFF используется несколько свободных переменных с одинаковой областью определения. Фактически применение целевого списка к области истинности WFF эквивалентно действию алгебраической операции проекции, а последний из приведенных вариантов представляет собой некоторую разновидность алгебраической операции переименования атрибута.

Выражением реляционного исчисления кортежей называется конструкция вида target_list WHERE WFF. Значением выражения является отношение, тело которого определяется WFF, а множество атрибутов и их имена – целевым списком.

Реляционное исчисление доменов.

В исчислении доменов областью определения переменных являются не отношения, а домены. Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного множества предикатов, позволяющих выражать так называемые *условия членства*. Если R – это *n*-арное отношение с атрибутами a_1, a_2, ..., a_n, то условие членства имеет вид R (a_{i1} : v_{i1}, a_{i2} : v_{i2}, ..., a_{im} : v_{im}) ($m \leq n$), где v_{ij} – это либо литерально задаваемая константа, либо имя доменной переменной. Условие членства принимает значение true в том и только в том случае, если в отношении R существует кортеж, содержащий указанные значения указанных атрибутов. Если v_{ij} – константа, то на атрибут a_{ij} накладывается жесткое условие, не зависящее от текущих значений доменных переменных; если же v_{ij} – имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Функциональные зависимости, замыкание множества функциональных зависимостей, аксиомы Армстронга, замыкание множества атрибутов. Минимальное покрытие множества функциональных зависимостей

Пусть задана переменная отношения *R*, и X и Y являются произвольными подмножествами заголовка *R* («составными» атрибутами). В значении переменной отношения *R* ***атрибут Y функционально зависит от атрибута X*** в том и только в том случае, если каждому значению X соответствует в точности одно значение Y. В этом случае говорят также, что атрибут **X *функционально определяет*** **атрибут Y** (X является **детерминантом** (*определителем*) для Y, а Y является зависимым от X). Будем обозначать это как *R*.X⟶*R*.Y.

Итак, мы будем иметь дело с FD, которые выполняются для всех возможных состояний тела соответствующего отношения и могут рассматриваться как ограничения целостности. Таких зависимостей может быть очень много. Поскольку они трактуются как ограничения целостности, за их соблюдением должна следить СУБД. Поэтому важно уметь сократить набор FD до минимума, поддержка которого гарантирует выполнение всех зависимостей.

FD A⟶B называется ***тривиальной***, если A⊇B (т. е. множество атрибутов A включает множество B или совпадает с множеством B). Очевидно, что любая тривиальная FD всегда выполняется. Частным случаем тривиальной FD является A⟶A. Поскольку тривиальные FD выполняются всегда, их нельзя трактовать как ограничения целостности, и поэтому они не представляют интереса с практической точки зрения. Однако в теоретических рассуждениях их наличие необходимо учитывать.

Замыканием множества *FD* ***S*** является множество FD S^+, включающее все FD, логически выводимые из FD множества S.

FD A⟶C называется ***транзитивной***, если существует такой атрибут B, что имеются функциональные зависимости A⟶B и B⟶C и отсутствует функциональная зависимость C⟶A.

Аксиомами Армстронга:

Подход к решению проблемы поиска замыкания S^+ множества FD S впервые предложил Вильям Армстронг. Им был предложен набор правил вывода новых FD из существующих (эти правила обычно называют **аксиомами Армстронга**, хотя справедливость правил доказывается на основе определения FD). Обычно принято формулировать эти правила вывода в следующей форме. Пусть A, B и C являются (в общем случае, составными) атрибутами отношения *R*. Множества A, B и C могут иметь непустое пересечение. Для краткости будем обозначать через AB A UNION B. Тогда:

1. если B ⊆A, то A⟶B (рефлексивность);
2. если A⟶B, то AC⟶BC (пополнение);
3. если A⟶B и B⟶C, то A⟶C (транзитивность).

Истинность первой аксиомы Армстронга следует из того, что при B⊆A FD A⟶B является тривиальной.

Справедливость второй аксиомы докажем от противного. Предположим, что FD AC⟶BC не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t_1 и t_2, такие, что t_1 {AC} = t_2 {AC} (a), но t_1 {BC} ≠ t_2 {BC} (b) (здесь t {A} обозначает проекцию кортежа t на множество атрибутов A). По аксиоме рефлексивности из равенства (a) следует, что t_1 {A} = t_2 {A}. Поскольку имеется FD A⟶B, должно соблюдаться равенство t_1 {B} = t_2 {B}. Тогда из неравенства (b) следует, что t_1 {C} ≠ t_2 {C}, что противоречит наличию тривиальной FD AC⟶C. Следовательно, предположение об отсутствии FD AC⟶BC не является верным, и **справедливость второй аксиомы доказана.**

Аналогично **докажем истинность третьей аксиомы** Армстронга. Предположим, что FD A⟶C не соблюдается. Это означает, что в некотором допустимом теле отношения найдутся два кортежа t_1 и t_2, такие, что t_1 {A} = t_2 {A}, но t_1 {C} ≠ t_2 {C}. Но из наличия FD A⟶B следует, что t_1 {B} = t_2 {B}, а потому из наличия FD B⟶C следует, что t_1 {C} = t_2 {C}. Следовательно, предположение об отсутствии FD A⟶C не является верным, и **справедливость третьей аксиомы доказана**.

Можно доказать, что система правил вывода Армстронга ***полна* и *совершенна*** *(sound and complete)* в том смысле, что для данного множества FD S любая FD, потенциально выводимая из S, может быть выведена на основе аксиом Армстронга, и применение этих аксиом не может привести к выводу лишней FD. Тем не менее Дейт по практическим соображениям предложил расширить базовый набор правил вывода еще пятью правилами:

4. A⟶A (самодетерминированность) – прямо следует из правила (1);
5. если A⟶BC, то A⟶B и A⟶C (декомпозиция) – из правила (1) следует, что BC⟶B; по правилу (3) A⟶B; аналогично, из BC⟶C и правила (3) следует A⟶C;
6. если A⟶B и A⟶C, то A⟶BC (объединение) – из правила (2) следует, что A⟶AB и AB⟶BC; из правила (3) следует, что A⟶BC;
7. если A⟶B и C⟶D, то AC⟶BD (композиция) – из правила (2) следует, что AC⟶BC и BC⟶BD; из правила (3) следует, что AC⟶BD;
8. если A⟶BC и B⟶D, то A⟶BCD (накопление) – из правила (2) следует, что BC⟶BCD; из правила (3) следует, что A⟶BCD.

Пусть заданы отношение *R*, множество Z атрибутов этого отношения (подмножество заголовка *R*, или составной атрибут *R*) и некоторое множество FD S, выполняемых для *R*. Тогда ***замыканием Z над S*** называется наибольшее множество Z^+ таких атрибутов Y отношения *R*, что FD Z⟶Y входит в S^+.

Алгоритм вычисления Z^+

```
K := 0; Z[0] := Z;
DO
   K := K+1;
   Z[K] := Z[K-1];
   FOR EACH FD A→B IN S DO
      IF A⊆Z[K] THEN Z[K] := (Z[K] UNION B) END DO;
UNTIL Z[K] = Z[K-1];
Z+ := Z[K];
```

Докажем корректность алгоритма по индукции. На нулевом шаге Z[0] = Z, FD Z⟶Z[I], очевидно, принадлежит S^+ (тривиальная FD «выводится» из любого множества FD). Пусть для некоторого K выполняется FD Z⟶Z[K], и пусть мы нашли в S такую FD A⟶B, что A⊆Z[K]. Тогда можно представить Z[K] в виде AC, и, следовательно, выполняется FD Z⟶AC. Но по правилу (8) мы имеем FD Z⟶ACB, т.е. FD Z⟶(Z[K] UNION B) входит во множество S^+, что переводит нас на следующий шаг индукции.

Пусть для примера имеется отношение с заголовком {A, B, C, D, E, F} и заданным множеством FD S = {A⟶D, AB⟶E, BF⟶E, CD⟶F, E⟶C}. Пусть требуется найти $\{AE\}^+$ над S. На первом проходе тела цикла DO Z[1] равно AE. В теле цикла FOR EACH будут найдены FD A⟶D и E⟶C, и в конце цикла Z[1] станет равным ACDE. На втором проходе тела цикла DO при Z[2], равном ACDE, в теле цикла FOR EACH будет найдена FD CD⟶F, и в конце цикла Z[2] станет равным ACDEF. Следующий проход тела цикла DO не изменит Z[3], и Z^+ ($\{AE\}^+$) будет равно ACDEF.

Алгоритм построения замыкания множества атрибутов Z над заданным множеством FD S помогает легко установить, входит ли заданная FD Z⟶B в замыкание S^+. Очевидно, что необходимым и достаточным условием для этого является B⊆Z^+, т. е. вхождение составного атрибута B в замыкание Z.

Суперключом отношения *R* называется любое подмножество K заголовка *R*, включающее, по меньшей мере, хотя бы один возможный ключ *R*.

Одно из следствий этого определения состоит в том, что подмножество K заголовка отношения *R* является суперключом тогда и только тогда, когда для любого атрибута A (возможно, составного) заголовка отношения *R* выполняется FD K⟶A. В терминах замыкания множества атрибутов K является суперключом тогда и только тогда, когда K^+ совпадает с заголовком *R*.

Множество FD S2 называется ***покрытием множества FD*** S1, если любая FD, выводимая из S1, выводится также из S2.

Легко заметить, что S2 является покрытием S1 тогда и только тогда, когда $S1^+$⊆$S2^+$. Два множества FD S1 и S2 называются ***эквивалентными***, если каждое из них является покрытием другого, т. е. $S1^+ = S2^+$.

Множество FD S называется минимальным в том и только в том случае, когда удовлетворяет следующим свойствам:

1. правая часть любой FD из S является множеством из одного атрибута (простым атрибутом);
2. детерминант каждой FD из S обладает свойством *минимальности*; это означает, что удаление любого атрибута из детерминанта приводит к изменению замыкания S^+, т. е. порождению множества FD, не эквивалентного S[30];
3. удаление любой FD из S приводит к изменению S^+, т. е. порождению множества FD, не эквивалентного S.

Для любого множества FD S существует (и даже может быть построено) эквивалентное ему **минимальное множество S^-.**

Приведем общую схему построения S^- по заданному множеству FD S. Во-первых, используя правило (5) (декомпозиции), мы можем привести множество S к эквивалентному множеству FD S1, правые части FD которого содержат только одноэлементные множества (простые атрибуты). Далее, для каждой FD из S1, детерминант D $\{D_1, D_2, \ldots, D_n\}$ которой содержит более одного атрибута, будем пытаться удалять атрибуты D_i, получая множество FD S2. Если после удаления атрибута D_i S2 эквивалентно S1, то этот атрибут удаляется, и пробуется следующий атрибут. Назовем S3 множество FD, полученное путем допустимого удаления атрибутов из всех

детерминантов FD множества S1. Наконец, для каждой FD f из множества S3 будем проверять эквивалентность множеств S3 и S3 MINUS {f}. Если эти множества эквивалентны, удалим f из множества S3, и в заключение получим множество S4, которое минимально и эквивалентно исходному множеству FD S.

Пример (оставил ради понимания что происходит) **:**

Пусть, например, имеется отношение *R* {A, B, C, D} и задано множество FD S = {A⟶B, A⟶BC, AB⟶C, AC⟶D, B⟶C}. По правилу декомпозиции S эквивалентно множеству S1 {A⟶B, A⟶C, AB⟶C, AC⟶D, B⟶C}. В детерминанте FD AC⟶D можно удалить атрибут C, поскольку по правилу дополнения из FD A⟶C следует A⟶AC; по правилу транзитивности выводится FD A⟶D, поэтому атрибут C в детерминанте FD AC⟶D является избыточным. FD AB⟶C может быть удалена, поскольку может быть выведена из FD A⟶C (по правилу пополнения из этой FD выводится AB⟶BC, а по правилу декомпозиции далее выводится AB⟶C). Наконец, FD A⟶C тоже выводится по правилу транзитивности из FD A⟶B и B⟶C. Таким образом, мы получаем множество зависимостей {A⟶B, A⟶D, B⟶C}, которое является минимальным и эквивалентно S по построению.

Минимальным покрытием множества *FD S* называется любое минимальное множество FD S1, эквивалентное S.

Поскольку для каждого множества FD существует эквивалентное минимальное множество FD, у каждого множества FD имеется хотя бы одно минимальное покрытие, причем для его нахождения не обязательно находить замыкание исходного множества.

Декомпозиция без потерь и функциональные зависимости, теорема Хита

Мы будем обсуждать подход к проектированию реляционных баз данных на основе нормализации, т. е. **декомпозиции** (разбиения путем проецирования) отношения, находящегося в предыдущей нормальной форме, на два или более отношений, 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. Такие декомпозиции называются ***декомпозициями без потерь*.**

Теорема Хита (Корректность декомпозиции)***.***
Пусть задано отношение *r* {A, B, C} (A, B и C, в общем случае, являются составными атрибутами) и выполняется FD A⟶B. Тогда *r* = (*r* PROJECT {A, B}) NATURAL JOIN (*r* PROJECT {A, C}).

Доказательство. Прежде всего, докажем, что в теле результата естественного соединения (обозначим этот результат через r_1) содержатся все кортежи тела отношения *r*. Действительно, пусть кортеж {a, b, c} ∈*r*. Тогда по определению операции взятия проекции {a, b} ∈(*r* PROJECT {A, B}) и {a, c} ∈(r PROJECT {A, C}). Следовательно, {a, b, c} ∈r_1. Теперь докажем, что в теле результата естественного соединения нет лишних кортежей, т. е. что если кортеж {a, b, c} ∈*r*1, то {a, b, c} ∈*r*. Если {a, b, c} ∈r_1, то существуют {a, b} ∈(*r* PROJECT {A, B}) и {a, c} ∈(*r* PROJECT {A, C}). Последнее условие может выполняться в том и только в том случае, когда существует кортеж {a, b*, c} ∈*r*. Но поскольку выполняется FD A⟶B, то b = b* и, следовательно, {a, b, c} = {a, b*, c}. ***Конец доказательства*.**

Атрибут B минимально зависит от атрибута A, если выполняется минимальная слева FD A⟶B.

Управление буферами основной памяти

Если бы при выполнении логических действий в базу данных сразу бы помещался результат этой операции, то скорость работы такой СУБД была бы сравнима со скоростью работы магнитных носителей. А скорости работы магнитных носителей всегда были и будут несравнимо ниже скорости работы оперативной памяти. Точно также, если при каждом обращении к страницам данных, СУБД будет лесть на внешнюю память, то произойдет тоже самое. На самом деле работа СУБД организована с помощью буферов в оперативной памяти. А именно с помощью буферного пула, в которой помещаются страницы данных или же индексные страницы, а также буфера журналльных записей, в который, соответственноЭ, помещаются все действия над журналом.

Когда СУБД требуется какая-то страницы данных, она копирует ее в буферный пул, и выполняет все изменения над ней в буферном пуле.

Проблемы начинаются позже. Что нужно делать, когда буферный пул страниц переполнен. Тогда, соответственно, нужно вытолкнуть какуюто страницу данных, предварительно вытолкнув запись из буфера журнальной информации о ней(если конечно эта страница изменялась), и переписать эту страницу во внешнюю память(если она изменялась).

Но вопрос, какую из страниц?

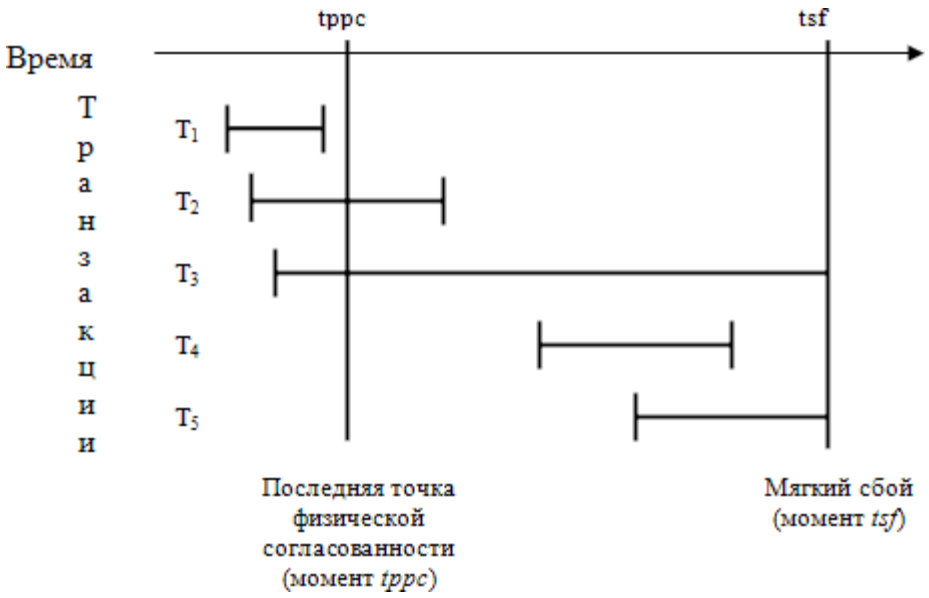
В операционных системах в таком случае использовался алгоритм LRU. Тоесть, выталкивалась та страница, которая дольше всего не использовалась в прошлом(своеобразный прогноз на будущее). Но СУБД умнее. Она может более точно прогнозировать, какая страница понадобится в будущем, а какая нет. Например, при последовательном сканировании (NEXT без использования индекса) некоторой таблицы, можно заранее предугадать, какая страница дальше потребуется, и скопировать ее во внутреннюю память. Точно также, можно с уверенностью сказать, что индексные страницы будут успользоваться очень часто(имеются в виду страницы хранящие корни индексов). Поэтому, кроме такого прогноза, еще этот алгоритм LRU в СУБД снабжен системой приоритетов. Так, например страница корневой индексной информации будет иметь самый высокий приоритет и почти никогда не будет вытолкнута.

Физически согласованное состояние базы данных

При мягком сбое(например, выключении питания) может получиться так, что некоторые страницы данных находятся в обновленом состоянии, а некоторые(эти страницы еще не вытолкнути из буферного пула) остались не обновленными.

Опр: Состояние базы данных называется физически согласованным, если все страницы данных, соответствующие каждому общекту базы данных либо соответствую состоянию объекта до изменения, либо состоянию объекта после изменения.

Пример



Теневой механизм

Изначально был придуман для работы с файлами. При открытии некоторого файла, таблица отображений логических блоков файла на реальные физические загружается в оперативную память. Далее, при выполнении изменения над некоторым блоком, во внешней памяти выделяется новый блок. Все изменения происходят над таблицей отображения, а в тонефой таблице все остается без изменений, и, при возникновении сбоя, всегда есть возможность восстановить файл с помощью теневой таблицы отображений. В Базах Данных все происходит аналогично, но хранятся 2 теневыет таблицы отображений, а также есть специальный индикатор,который говорит, какая таблица текущая. После того, когда в журнал добавляетсчя новая точка физически согласованного состояния, все страницы выталкиваются из буферного пула, все логические операции завершаются. Делается новая теневая таблица отображения. И флаг меняется на B(раньше был A). Если во время этого случился сбой, то используется старая теневая таблица отображения.

Чтобы восстановить последнее согласованное состояние текущая таблица отображения заменяется теневой таблицей.

Другой подход- Журнализация постраницных изменений. Он основан на том, что нужно журнулировать не только лоигические изменения над базой данных(изменения кортежей), но и физические постраницные изменения.Сначала откатываются все незавершенные логические операции над БД. Тоочно также журнализация постраничных изменений от одной лигической операции заканчивается завершающей операцией.

Проблема: необходимо понять, какие страницы нуждаются в восстановлении. Для этого при выталкивании текоторой страницы, в нее помещяется номер последней записи о постраничном изменении этой страницы. Этот же номер помещается и в саму запись. Соответственно, если в журнале номер последней записи больше, значит восстанавливать эту страницу не требуется.

Иногда эти два журнала(логический и физический) хранят вместе, но обычно их разделяют на два

При создании новой точки физической согласованности:

1) Прекращается инициализация новых логических операций
2) После завершения всех лог операций, все страницы выталкиывются из буферного пула
3) Формируется и вуталкивается в буферный пул специальная запись о точке физического согласованного состояния
4) Если предыдущая запись выполнилась нормально, то внови разрешают логические операции.

Ситуации, требующие восстановление базы данных:

1) Откат транзакций. Это может произойти путем выполнения операции ROLLBACK или же путем возникновения исключительной ситуации(например, деление на 0). Также транзакция может быть выбрана жертвой при возникновении тупиковой ситуации
2) Мягкий сбой.Теряется содержимое оперативной памяти(например, при аварийном отключении питания.)
3) Самый редкий случай-жесткий сбой. Возникает при потери данных с внешней памяти(примером может служить поломка магнитного диска)

Во всех трех случаях для восстановления базы данных используется журнал. В нем хранится вся последовательность изменений, происходящих с базой данных.

Существуют два основных подходя ведения журнала. Первый из которых- также хранение индивидуальных журналов для каждой транзакций. Обесечивает быстрый откат транзакций.

Второй подход основывается на ведении общего журнала.

Но каждый из подходов означает хранение избыточной информации для обеспечения возможности восстановления базы данных.

Индивидуальный откат транзакций

При индивидуальном откате транзакиций все операции, относящиеся к данной транзакции связывают в обратный список,в котором последней операцие всегда является операция начала транзакции. В общем случае для завершенной транзакции, результаты всех операций уже будут вытолкнуты из буферного пула, а значит первой операцией будет операция, завершающая транзакцию. Такие транзакции уже невозможно откатить. В случае, когда транзакция не завершена, какие-то даже уже выполненные операции могут быть из буфера не вытолкнуты и, соответственно, в журнале(по протоколу write ahead Log) находиться не могут. Тогда первой для такой транзакции будет некоторая операция транзакции.

1) Сначала по порядку из списка выбираются операции
2) Для выполнения отката индивидуальной транзакции, все операции из данного списка, заменяясь на противоположные(тоесть вставка заменяется на удаление удаление на вставку а обновление заменяется на обновление, производящще обратные действия)
3) Каждая такая операция также журнализируется. В случае происхождения сбоя во время отката транзакции, все операции по откату откатываются и потом откат начинается сначала.

4) Далее, в случае успешного завершения, журнале отражается операция завершения транзакции, и такая транзакция считается завершенной и зафиксированной.

Протокол Write Ahead Log.

При выполнении некоторых транзакций, действия изменения базы данных не сразу отражаются в базе данных, а сначала буферизуются, и когда в буфере нет места для вставки новой страницы, и некоторая страница выталкивается из буфера,и, если она отличается от страницы в во внешней памяти, то страница во внешней памяти заменчется на нее.Журнальная информация выталкивается из буфера в том случае, если этот буфер целиком заполнился. Вопрос состоит в том, в каком порядке нужно помещать информацию из буферов так, чтобы при возможности можно было восстановить БД при сбое во время такого выталкивания. Ответ дает протокол Write Ahead Log

Он заключается в том, что сначала из буфера журнальной информации выталкивается информация об изменении, а лишь потом выталкивается соответствующее изменение базы данных в основную память. Если наоборот, то, если после выталкивания страницы произойдет сбой, то никакой возможности восстановления уже нет, так как такой информации о изменении в журнале нет.

Восстановление базы данных после жесткого сбоя

Самый простой способ восстановления:

Нужел логический журнал и архивная копия

1) Копирование Базы данных из архивной копии на носитель
2) Обратное выполнение всех операций из логического журнала

Если журнал утерян, то единственный способ-возврат к архивной копии.

Способы архивирования:

1) Администратор сам решает, когда нужно архивировать
2) Тогда, когда переполняется журнал

Но можно выполнять архивацию и реже, чем переполняется журнал.Можно выполнять архивацию самого журнала. В таком случае для восстановления БД нужна последняя версия логического журнала, последовательность архивов журналов и архивная копия БД.

Также архивные копии БД можно сжимать

1) Можно сжимать отдельную копию, оставляя только последнее действие над каким-то кортежем. Например, если последнее действие – удаление, то остальные действия можно не хранить
2) Можно аналогичным способом сжимать и последоватьельные архивные копии логического журнала.

Тоесть, для восстановления БД достаточно зранить архивную копию БД, сжатый архив журнала и последньь версию логического журнала.