# Лекция 1

## Технология проектирования баз данных

Вопросы:

1. Проектирование базы данных как элемент информационной технологии;

Теоретические основы проектирования БД;

1. Синтез БД.

### Проектирование базы данных как элемент информационной технологии

Как видно из материалов предыдущих лекций основу большинства информационных технологий составляют большие массивы накопленной информации. Основной формой организации хранения данных в информационных системах являются базы данных. В курсе “Автоматизированные системы обработки учетной информации” мы рассмотрели основные понятия, связанные с моделями данных, теоретические основы разработки простейших баз данных и жизненный цикл баз данных. Теперь, рассматривая БД как часть информационной технологии, необходимо по новому взглянуть на проблему проектирования базы.

Проблемы проектирования связаны с функциями БД в программно - технологической среде, поддерживающей информационные технологии. В общем случае место БД можно отразить следующей схемой:

Приложения поддержки

База

данных

Операционная система

Аппаратная

среда

информационных

технологий

Прочие

приложения

Поскольку база данных является связующим звеном между пользовательскими приложениями и аппаратными средствами, ее проектирование можно разделить на два направления: проектирование структуры и пользовательских приложений и распределение данных по аппаратным средствам (в случае баз данных на сетях). В данном разделе мы рассмотрим вопросы проектирования структуры базы данных. В дисциплине АСОЭИ, рассматривая основы реляционной алгебры и разработки реляционных моделей, мы коснулись вопросов проектирования реляционных баз данных. Одной из распространенных технологий разработки БД является следующая:

1. сбор данных о предметной области;
2. анализ представлений пользователей;
3. интеграция представлений пользователей;
4. разработка сетевой модели;
5. преобразование сетевой модели в первую нормальную форму реляционной модели;
6. нормализация отношений путем преобразования их к третьей нормальной форме.

В результате получается модель реляционной базы данных, которая представляет собой совокупность взаимосвязанных отношений.

Построение сетевой модели связано скорее с потребностью разработчика графически представить взаимосвязь данных, полученных в результате интеграции представлений пользователей. Преобразование сетевой модели в реляционную дает первую нормальную форму последней. Напомним, что отношение R находится в первой нормальной форме, если значения в dom(A) являются атомарными для каждого атрибута А в R . Вторая и третья нормальные формы позволяют избежать аномалий при обновлении данных и избавится от информационной избыточности в отношениях. Напомним, что отношение R нормальной форме, если оно находится в первой нормальной форме и каждый атрибут не являющийся ключом полностью зависит от любого ключа в R. И отношение R находится в третьей нормальной форме, если оно находится во 2НФ и каждый атрибут, не являющийся первичным ключом не транзитивно зависит от любого возможного ключа.

Недостатком такого подхода является то, что в моделях, имеющих десятки и сотни атрибутов очень трудно имперически построить модель, все отношения которой заданы в третьей нормальной форме и связаны между собой таким образом, что составляют единое целое.

Пример.

А В С

G V M N

D F G H

B M T X

Другим подходом является возможность формального синтеза модели на основании априорно установленных зависимостей между атрибутами. Зависимости между атрибутами устанавливаются на основании смысловой связи.

Пример.

*НОМЕР\_ЗАЧЕТКИ - ИМЯ\_СТУДЕНТА*

*НОМЕР\_РЕЙСА - ДАТА\_ВЫЛЕТА*

Безусловно такой подход к разработке модели базы данных предпочтительнее, так как позволяет автоматизировать процесс моделирования. Для реализации этого подхода необходимо расширение теоретической базы, полученной в курсе АСОЭИ.

### Теоретические основы проектирования БД.

Основные понятия.

Поскольку рассматриваемый подход к разработке реляционной модели базируется на формальной логике, то в его основе должны лежать некоторые фундаментальные формализации. В теории реляционных баз данных к ним относятся понятия атрибута, отношения, ключа и функциональной зависимости.

Атрибутом будем называть поименованное свойство объекта и обозначать Аi , где . Домен атрибута Аi обозначим dom(Аi). Тогда отношением R называется конечное множество атрибутов . Ключ отношения R является подмножеством К = со следующим свойством. Для любых двух различных кортежей t1 и t2 в R существует такое , что t1(B)t2(B). Другими словами , не существует двух кортежей, имеющих одно и то же значение на всех атрибутах из К . Таким образом, достаточно знать К - значение кортежа, чтобы идентифицировать кортеж однозначно.

Пример.

*СТУДЕНТ[НОМЕР\_ЗАЧЕТКИ,ИМЯ,КУРС,ГРУППА]*

Ключи, явно указанные в модели называются выделенными. Могут быть ключи отличные от выделенных и называемые неявными ключами. Например ИМЯ в предыдущем прмере.

Под функциональной зависимостью атрибутов или F-зависимостью понимают такую связь между атрибутами, когда значения кортежа на одном множестве атрибутов единственным образом определяют эти значения на другом множестве атрибутов. Так в отношении:

*ГРАФИК[ПИЛОТ,РЕЙС,ДАТА,ВРЕМЯ]*

##### ПИЛОТ функционально зависит от {РЕЙС,ДАТА}

F-зависимости принято обозначать {РЕЙС,ДАТА}-> ПИЛОТ и говорят, что РЕЙС и ДАТА функционально определяют ПИЛОТ.

В терминах теории множеств и реляционной алгебры F-зависимость определяется так. Пусть R отношение и X, Y подмножества атрибутов в R. Отношение R удовлетворяет функциональной зависимости X -> Y, если πY(σX-x®) имеет не более чем один кортеж для каждого Х - значения х. В F-зависимости X->Y подмножество X называется левой частью, а Y - правой частью.

# Лекция 2

Такая интерпретация функциональной зависимости является основой алгоритма SATISFIES, приводимого ниже.

#### SATISFIES

Вход: Отншение R и F-зависимость X->Y.

Выход: истина, если R удовлетворяет X->Y, ложь - в противном случае.

#### SATISFIES(R,X->Y)

1. Отсортировать отношение R по Х-столбцам так, чтобы собрать кортежи с равными Х-значениями вмести.
2. Если каждая совокупность кортежей с равными Х-значениями имеет также равные Y-значения, то на выходе получаем истину, а в противном случае - ложь.

Этот алгоритм проверяет, удовлетворяет ли отношение R F-зависимости X -> Y.

Пример.

В результате выполнения алгоритма SATISFIES выясним удовлетворяет ли F-зависимость РЕЙС -> ВРЕМЯ\_ВЫЛЕТА следующему отношению

#### ГРАФИК

|  |  |  |  |
| --- | --- | --- | --- |
| ПИЛОТ | РЕЙС | ДАТА | ВРЕМЯ\_ВЫЛЕТА |
| А... | 83 | 9 авг | 10:15 |
| П... | 83 | 11 авг | 10:15 |
| А... | 116 | 10 авг | 13:25 |
| Р... | 116 | 12 авг | 13:25 |
| П... | 281 | 8 авг | 5:50 |
| С... | 281 | 9 авг | 5:50 |
| П... | 301 | 12 авг | 18:35 |
| С... | 412 | 15 авг | 13:25 |

Однако F-зависимость ВРЕМЯ\_ВЫЛЕТА -> РЕЙС согласно этому алгоритму не выполняется для этого отношения

#### ГРАФИК

|  |  |  |  |
| --- | --- | --- | --- |
| ПИЛОТ | РЕЙС | ДАТА | ВРЕМЯ\_ВЫЛЕТА |
| П... | 281 | 8 авг | 5:50 |
| С... | 281 | 9 авг | 5:50 |
| А... | 83 | 9 авг | 10:15 |
| П... | 83 | 11 авг | 10:15 |
| А... | 116 | 10 авг | 13:25 |
| Р... | 116 | 12 авг | 13:25 |
| С... | 412 | 15 авг | 13:25 |
| П... | 301 | 12 авг | 18:35 |

Для разработки модели базы данных необходимо знать полное множество F-зависимостей. Чтобы найти их, необходимы семантические знания об исходном отношении R. Поэтому можно считать семейство F-завсимостей заданным. Обозначим его F. Однако при таком подходе нельзя быть уверенным, что найдены все F-зависимости отношения R. Для того, чтобы найти все F-зависимости, если известны некоторые из них, можно воспользоваться аксиомами вывода. Возможность получения новых F-зависимостей с помощью аксиом вывода базируется на следующем правиле. Мнжество F-зависимостей F влечет за собой F-зависимость X -> Y (обозначение: F =X -> Y ), если каждое отношение удовлетворяющее всем зависимостям в F, удовлетворяет также зависимости X -> Y. Аксиома вывода - это правило, устанавливающее, что если отношение удовлетворяет определенным F-зависимостям, то оно должно удовлетворять и некоторым другим F-зависимостям. Существует шесть аксиом вывода:

Рефлексивность: X -> X.

Пополнение: X -> Y влечет за собой XZ -> y.

Аддитивность: X -> Y и X -> Z влечет за собой X -> YZ.

Проективность: X -> YZ влечет за собой X -> Z.

Транзитивность: X -> Y и Y -> Z влечет за собой X -> Z.

Псевдотранзитивность: X -> Y и YZ -> W влечет за собой XZ -> W.

Пример.

Пусть дано отношение R , а X , Y и Z подмножества R . Предположим, что отношению удовлетворяет XY -> Z и X -> Y . Согласно аксиоме псевдотранзитивности получим XX -> Z или X -> Z.

Если даны аксиомы рефлексивности, пополнения и псевдотранзитивности, то из них можно вывести все остальные. Иногда их называют аксиомами Армстронга.

Пусть F-множество F-зависимостей для отношения R . *Замыкание* F , обозначаемое F­­­­+ , - это наименьшее содержащее F множество, такое что при применении к нему аксиом Армстронга нельзя получить ни одной F - зависимости, не принадлежащей F.

Пример.

Пусть F = {AB -> C, C -> B } - множество F-зависимостей на R(ABC). F+ = {A -> A, AB -> A, AC -> A, ABC -> A, B -> B, AB -> B, BC -> B, ABC -> B, C -> C, AC -> C, BC -> C, ABC -> C, AB -> AB, ABC -> AB, AC -> AC, ABC -> AC, BC -> BC, ABC -> BC, ABC -> ABC, AB -> C, AB -> AC, AB -> BC, AB -> ABC, C -> B, C -> BC, AC -> B, AC -> AB}

Таким образом, если известно множество F-зависимостей удовлетворяющих отношению R, можно найти все F- зависимости, удовлетворяющие этому отношению. Говорят, что F = X -> Y ,если X -> Y F+ .

# Лекция 3

Получение замыкания F+ не обязательно для установления F = X -> Y.

Для этого достаточно воспользоваться алгоритмом MEMBER .

Алгоритм MEMBER.

Вход: Множество F-зависимостей F и F-зависимость X -> Y.

Выход: истина, если F = F = X -> Y, ложь в противном случае.

#### MEMBER(F, X -> Y)

begin

if Y CLOSURE(X,F) then return (истина)

else return(ложь)

end

Здесь CLOSURE алгоритм, позволяющий выявить список атрибутов входящих в множество F, который имеет вид.

Алгоритм CLOSURE.

Вход: Множество атрибутов Х и множество F-зависимостей F.

Выход: Замыкание Х над F.

#### CLOSURE(X,F)

begin

##### OLDDEP = 0; NEWDEP = X

while NEWDEP OLDDEP do begin

###### OLDDEP = NEWDEP

for каждая F- зависимость W -> Z в F do

if NEWDEP W then

NEWDEP = NEWDEP Z

end

return(NEWDEP)

end

##### Пример работы алгоритма MEMBER

Пусть F = {НОМЕР\_РЕЙСА ДАТА\_ВЫЛЕТА -> КОЛИЧЕСТВО\_МЕСТ,

НОМЕР\_РЕЙСА -> ПУНКТ\_ОТПРАВЛЕНИЯ, НОМЕР\_РЕЙСА ДАТА\_ВЫЛЕТА -> ПИЛОТ} и необходимо установить F |= НОМЕР\_РЕЙСА -> ПИЛОТ

Используем для этого алгоритм MEMBER

### Покрытия функциональных зависимостей

Для формирования БД, как системы взаимосвязанных отношений на основании известных (из семантических соображений) F-зависимостей необходимо иметь способ минимизации первоначального множества F-зависимостей. Это необходимо для минимизации дублирования данных, обеспечения их согласованности и целостности. Теоретической основой для построения такого способа является теория покрытий функциональных зависимостей.

Определение.

Два множества F-зависимостей F и G над отношением R эквивалентны, , если F+ = G+ . Если , то F есть *покрытие* для G. Предполагается, что имеет смысл рассматривать в качестве покрытий такие множества F, которые не превосходят множество G по числу F-зависимостей.

Из этого определения следует, что для установления факта, что множество функциональных зависимостей F является покрытием G , необходимо определить эквивалентность F и G. Практически это достигается путем использования следующего алгоритма:

Алгоритм EQUIV

Вход: два множества F- зависимостей F и G.

Выход: истина, если ; ложь в противном случае.

#### EQUIV(F,G)

begin

v=DERIVES(F,G) and DERIVES(G,F);

return(v)

end

Здесь DERIVES алгоритм проверяет условие F |= G и имеет вид:

Алгоритм DERIVES

Вход: два множества F- зависимостей F и G.

Выход: истина, если F |= G; ложь в противном случае.

#### DERIVES(F,G)

begin

v= истина

for каждая F-зависимость X -> Y из G do

v = v and MEMBER(F, X -> Y)

end

return(v)

end

Множество F-зависимостей F не избыточно, если у него нет такого собственного подмножества F’ , что F’F . Если такое множество F’ существует, то F избыточно. F является не избыточным покрытием G, если F есть покрытие G и F не избыточно.

Пример. Пусть G = { AB -> C, A -> B, B -> C, A -> C}. Множество F= {AB -> C, A -> B, B -> C} эквивалентно G, но избыточно, потому что F’ = {A -> B, B -> C} также является покрытием G. Множество F’ представляет собой не избыточное покрытие G.

Действительно, согласно алгоритму EQUIV , так как DERIVES(F,G) дает истину и DERIVES(G,F) так же истина. То же самое можно сказать относительно F’ и G.

##### (Подробнее)

Множество F не избыточно, если в нем не существует F-зависимости X -> Y, такой, что F - { X -> Y} |= X -> Y . Назовем F-зависимость из F избыточной в F , если F - { X -> Y} |= X -> Y.

Для любого множества F-зависимостей G существует некоторое подмножество F, такое, что F является не избыточным покрытием G. Если G не избыточно, то F=G. Для определения не избыточного покрытия множества F- зависимостей G существует алгоритм NONREDUN, который имеет вид:

Вход: множество F-зависимостей G.

Выход: не избыточное покрытие G.

#### NONREDUN(G)

begin

F=G

for каждая зависимость X -> Y из G do

if MEMBER(F-{X->Y],X->Y) then F=F-{X->Y}

end

return(F)

end

Пример: Пусть G= {A -> B, B -> A, B -> C, A -> C}.

Результатом работы алгоритма является множество:

{A -> B, B -> A, A -> C}.

Если бы G было представлено в порядке {A -> B, A -> C, B -> A , B -> C} , то результатом работы алгоритма было бы

{A -> B, B -> A, B -> C}.

Из примера видно, что множество F-зависимостей G может иметь более одного неизбыточного покрытия. Могут так же существовать неизбыточные покрытия G, не содержащиеся в G. В приведенном примере таким неизбыточным покрытием будет множество

F = {A -> B, B -> A, AB -> C}.

Если F-неизбыточное множество F-зависимостей, то в нем нет “лишних” зависимостей в том смысле, что нельзя уменьшить F , удалив некоторые из них. Удаление любой F-зависимости из F приведет к множеству, не эквивалентному F. Однако размер можно уменьшить, удалив некоторые атрибуты F-зависимостей F.

Определение. Пусть F-множество F-зависимостей над R и X -> Y есть F-зависимость из F. Атрибут А из R называется посторонним в X -> Y относительно F, если

1. и (F - {X -> Y}) {Z -> Y}F или
2. Y = AW, YW и (F - {X -> Y}) {X -> W}F.

Иными словами, А - посторонний атрибут, если он может быть удален из правой или левой части X -> Y без изменения замыкания F.

Пример. Пусть G = {A -> BC,B -> C,AB -> D}. Атрибут С является посторонним в правой части A -> BC, а атрибут B - в левой части AB -> D.

Определение. Пусть F - множество F-зависимостей над R и X -> Y принадлежит F. F-зависимость X -> Y называется редуцированной слева, если Х не содержит постороннего атрибута А и редуцированной справа, если Y не содержит атрибута А , постороннего для X -> y. Зависимость X -> Y называется редуцированной, если она редуцирована слева и справа и Y . Редуцированная слева F-зависимость называется также полной F-зависимостью.

Определение. Множество F-зависимостей F называется редуцированным (слева, справа), если каждая F-зависимость из F редуцирована (соответственно слева, справа).

Пример. Множество G = {A -> BC, B -> C, AB -> D} не является редуцированным ни справа, ни слева. Множество G1 = {A -> BC, B -> C, A -> D} редуцировано слева, но не справа, а G2 = {A -> B, B -> C, AB -> D} редуцировано справа, но не слева. Множество G3 = {A -> B, B -> C, A -> D} редуцировано слева и справа, следовательно, поскольку правые части не пусты, редуцировано.

Для нахождения редуцированных покрытий используется алгоритм:

#### REDUCE

Вход: множество F-зависимостей G.

Выход: редуцированное покрытие G.

#### REDUCE(G)

begin

F = RIGHTRED(LEFTRED(G))

удалить из F все F-зависимости вида X ->

return(F)

end

здесь RIGHTRED и LEFTRED алгоритмы редуцирования соответственно справа и слева, которые имеют вид:

#### RIGHTRED

Вход: множество F-зависимостей G.

Выход: редуцированное справа покрытие G.

#### RIGHTRED(G)

begin

F = G

for каждая зависимость X -> Y из G do

for каждый атрибут А из Y do

if MEMBER(F - {X -> Y} {X ->(Y - A)}, X -> A) then

удалить А из Y в X -> Y из F

end

end

return(F)

end

Алгоритм LEFTRED

Вход: множество F-зависимостей G.

Выход: редуцированное слева покрытие G.

Begin

F = G

for каждая зависимость X -> Y из G do

for каждый атрибут А из Х do

if MEMBER(F,{X - A) -> Y) then

удалить А из X в X -> Y из F

end

end

return(F)

end

Пример. Пусть G’= {A -> C, AB -> DE, AB ->CDI, AC -> J}.Из LEFTRED(G’) получаем G” = {A -> C, AB -> DE, AB -> CDI, A -> J} и из RIGHTRED(G”) получаем F= {A -> C, AB -> E, AB -> DI, A -> J}, уже редуцированное.

Далее потребуется находить разбиение множества F- зависимостей, заданных на отношении R на подмножества, которые представляют собой классы F- зависимостей с эквивалентной левой частью.

Определение: два множества атрибутов X и Y называются эквивалентными на множестве F- зависимостей F, если F |= X->Y и F |= Y ->X.

Например пусть дано F={A -> BC, B -> A, AD -> E}, найдем все F- зависимости эквивалентные левой части первой, т.е. А. Левая часть второй F- зависимости (В) эквивалентна А ? Проверим F |= A -> B и F |= B -> A . Это действительно так. Следовательно А эквивалентно В и первые две F- зависимости можно объединить в один класс эквивалентности, который обозначается в общем случае ЕА(Х). Множество классов эквивалентности для заданного множества F- зависимостей обозначается F . Сокращенным способом записи F- зависимостей с эквивалентными левыми частями является составная функциональная зависимость (CF-зависимость), которая имеет вид:

##### (X1, X2, ..., Xk) -> Y

где X1, X2, ..., Xk , множество эквивалентных левых частей F- зависимостей, а Y объединение правых частей F- зависимостей.

## Синтез реляционных баз данных

База данных состоит из множества атрибутов и ключей. С точки зрения теоретико-множественного описания реляционной базой данных d называется такая совокупность отношений {R1, R2, ...,Rp}, в которой каждое отношение имеет вид Ri= (Si,Ki), где Si- множество атрибутов, а Ki - множество атрибутов образующих ключ.

Предположим на входе задано множество F- зависимостей F над R. С их помощью требуется создать базу данных R=( R1, R2, ...,Rp). Эта БД должна удовлетворять следующим требованиям:

1. множество F полностью характеризуется с помощью R , т.е.



где К – выделенный ключ Ri}

1. Каждое отношение Ri находится в третьей нормальной форме.
2. Не существует базы данных с меньшим числом отношений, удовлетворяющим пунктам 1 и 2.
3. Соединение всех полученных отношений Ri дает исходное отношение R.

Алгоритм порождающий базу данных из заданных F-зависимостей называется алгоритмом синтеза.

Определение. Если R – база данных и на ней задано множество F-зависимостей G, то в ней существует по крайней мере |EG| отношений. Это означает, что в R столько же отношений, сколько и классов эквивалентности. Из этого следует следующее.

Пусть F - множество F – зависимостей. Любая база данных должна иметь |EF’| отношений, где F’ неизбыточное покрытие для F.

Исходя из этого строится способ построения структуры базы данных.

Сначала находится неизбыточное покрытиеF’ для F и в EF’ вычисляем классы эквивалентности. Для каждого EF’(X) строим отношение, состоящее из всех атрибутов, появляющихся в EF’(X). При этом атрибуты левой части каждого класса эквивалентности образуют выделенный ключ.

Реализация этого способа позволяет получить алгоритм SYNTHESIZE

Вход: множество F – зависимостей F над R.

Выход: полная схема баз данных для F.

1. Наити для F редуцированное минимальное покрытие G.
2. Для каждой CF – зависимости (X1,X2,…,Xk) Y из G построить отношение Rj= X1X2…XkY с выделенными ключами K={X1,X2,…Xk).
3. Вернуться к п. 2.

Пример.

A B1B2C1C2DEI1I2I3J

B1B2C1 AC2DEI1I2I3J

B1B2C2 AC1DEI1I2I3J

E I1I2I3

C1D J C2D J

I1I2 I3  I2I2 I1 I1I3 I2

И пусть R= AB1B2C1C2DEI1I2I3J

Множество минимально, но не редуцировано. Редуцируя F , получим

F’= {A B1B2C1C2DE E I1I2

B1B2C1 A B1B2C2 A

C1D J C2D J

I1I2 I3  I2I2 I1 I1I3  I2}

Образуя классы эквивалентности имеем

G={ (AB1B2C1 ,B1B2C2) DE

(E) I1I2

(C1D) J (C2D) J

(I1I2, I2I2, I1I3)}

Преобразуя каждую CF – в отношения с выделенными ключами, получим

R1=AB1B2C1C2DE K1= {AB1B2C1 ,B1B2C2}

R2= EI1I2 K2={E}

R3= C1DJ K3={C1D}

R4= C2DJ K4={C2D}

R5= I1I2I3  K5={ I1I2, I2I2, I1I3}

Окончательная схема БД будет R=( R1, R2, R3, R4 ,R5)

## Распределенная обработка данных

Под распределенной обработкой данных понимается такой способ хранения и обработки данных, когда отдельное приложение может обрабатывать данные,, распределенные на множестве различных баз данных, управление которыми осуществляют различными СУБД, работающие на различных машинах с различными операционными системами, соединенных коммуникационными системами. Распределенная база данных (РБД) является виртуальным объектом, части которого расположены на удаленных базах данных, связанных каналами связи.

Физически РБД состоит из набора **узлов**, связанных коммуникационной сетью, в которой:

* Каждый узел обладает своими собственными системами баз данных;
* Узлы работают согласованно, поэтому пользователь может получить доступ к данным на любом узле сети, как будто все данные находятся на собственном узле.

Каждый узел обладает своими собственными базами данных, собственными локальными пользователями, собственной СУБД и программным обеспечением для управления транзакциями, а так же собственным диспетчером передачи данных. Распределенная СУБД может рассматриваться как некий способ совместной работы отдельных локальных СУБД, расположенных на разных локальных узлах. Причем новый компонент программного обеспечения на каждом узле поддерживает все необходимые функции совместной работы. Комбинация этого компонента и существующей СУБД называется Распределенной Системой Управления Базами Данных (РСУБД).

В основе распределённых баз данных лежат следующие требования:

1. Локальная автономия;
2. Независимость от центрального узла;
3. Непрерывное функционирование;
4. Независимость от расположения;
5. Независимость от фрагментации;
6. Независимость от репликации;
7. Обработка распределённых запросов;
8. Управление распределёнными транзакциями;
9. Независимость от аппаратного обеспечения;
10. Независимость от операционной системы;
11. Независимость от сети;
12. Независимость от СУБД.

### Локальная автономия

В распределенной системе узлы следует делать автономными. Локальная автономия означает, что функционирование любого узла Х не зависит от успешного выполнения операций на некотором узле У . В противном случае выход из строя узла У может привести к невозможности выполнения операций на узле Х . Из принципа локальной автономии следует, что владение и управление данными осуществляется локально вместе с локальным ведением учета. В действительности цель локальной автономии достигается не полностью, поскольку часто узел Х должен представлять некоторую часть управления узлу У , поэтому говорят не о полной, а о максимально возможной автономии.

### Независимость от центрального узла.

Под локальной автономией понимается, что все узлы должны рассматриваться как равные. Следовательно, не должно существовать никакой зависимости и от центрального «основного» узла с некоторым централизованным обслуживанием, например централизованной обработкой запросов, централизованным управлением транзакциями или централизованным присвоением имен. Зависимость от центрального узла нежелательна по двум причинам. Во-первых, центральный узел может быть «узким» местом всей системы, а во-вторых, более важно то, что система в целом становится уязвимой, т.е. при повреждении центрального узла может выйти из строя вся система.

### Непрерывное функционирование

Одним из преимуществ распределенных систем является то, что они обеспечивают более высокую надежность и доступность.

* **Надежность** (вероятность того, что система выполняет свойственные ей функции в заданный момент времени) повышается благодаря работе распределенных систем не по принципу «все или ничего», а в постоянном режиме; т.е. работа системы продолжается , хотя и на более низком уровне, даже в случае неисправности некоторого отдельного компонента, например узла.
* Доступность (вероятность того, что система исправна и работает в течение некоторого промежутка времени) повышается частично по той же причине, а частично благодаря возможности репликации данных.

### Независимость от расположения

Эта цель предполагает обеспечение такого режима работы с данными, при котором пользователю не нужно знать, на каком узле находятся требуемые данные. При этом значительно упрощаются пользовательские программы, а также не требуется их изменения при перемещении данных в системе.

### Независимость от фрагментации

В системе поддерживается фрагментация данных, если некоторое отношение из соображений физического хранения необходимо разделить на части или фрагменты. Фрагментация желательна для повышения производительности системы, поскольку данные лучше хранить в том месте, где они наиболее часто используются. При такой организации многие операции становятся локальными, а объем передаваемых в сети данных снизится.

Существует два типа фрагментации – горизонтальная и вертикальная, которые связаны с операциями селекции и проекции соответственно, т.е. горизонтальный фрагмент может быть получен с помощью операции селекции, а вертикальный – проекцией. Реконструкцию исходного отношения на основе его фрагментов можно осуществить с помощью операций соединения (для вертикальных фрагментов) и объединения (для горизонтальных фрагментов).

В фрагментированной системе необходимо обеспечить поддержку независимости от фрагментации, т.е. пользователь не должен «ощущать» фрагментацию данных.

### Независимость от репликации

В системе поддерживается независимость от репликации, если заданное отношение или фрагмент могут быть представлены различными копиями (репликами) хранимыми на разных узлах. Репликация полезна по двум причинам. Во-первых, благодаря ей достигается большая производительность, т.к. приложения могут работать с локальными копиями , не обмениваясь данными с удаленными узлами. Во-вторых, репликация позволяет обеспечить большую доступность, т.к. реплицированный объект остается доступным для обработки до тех пор, пока остается хотя бы одна его реплика. Главный недостаток репликации заключается в том, что при обновлении реплицируемого объекта, все его копии должны синхронизироваться.

В системе, которая поддерживает репликацию данных, должна также поддерживаться независимость от репликации, т.е. пользователь не должен касаться проблем связанных с созданием и синхронизацией копий.

### Обработка распределенных запросов

При обработке в распределенной системе запроса необходимо выработать эффективную стратегию его реализации. Например, запрос на объединение отношений Rx , расположенного на узле X , и отношения Ry , хранимого на узле Y , может быть выполнен с помощью перемещения отношения Rx на узел Y , перемещения отношения Ry на узел X или перемещения этих двух отношений на третий узел Z и т.д. Это означает, что при выполнении запроса на распределенной БД необходим его предварительный анализ с последующим выбором оптимальной стратегии его реализации.

### Управление распределенными транзакциями

В распределенной системе выполнение транзакции связано с исполнением программных кодов на нескольких узлах. Транзакция это логическая единица работы, которая включает всю совокупность действий, необходимых для реализации запроса. Транзакция считается неделимым процессом, т.е. если какое либо из составляющих действий окажется не выполненным, то вся транзакция считается не выполненной. Каждый программный код, исполняемый на каком либо узле при выполнении транзакции, называется агентом. Таким образом, транзакция состоит из нескольких агентов, т.е. процессов реализующих транзакцию.

В процессе управления транзакцией выделяют управление восстановлением и управление параллельной обработкой. Первое из них базируется на протоколе двухфазной фиксации. В грубом приближении в соответствии с этим протоколом в начале транзакции устанавливается точка фиксации данных, т.е. как бы создается копия данных, которые предполагается изменить в результате транзакции. Если транзакция завершена нормально, то точка фиксации сохраняется до выполнения следующей транзакции. Если же произошел сбой, то система возвращает состояние данных в точку фиксации, позволяя не допустить необратимого неправильного изменения БД. Управление параллельной обработкой предполагает установку блокировок на отношения, группы записей с целью не допустить изменение данных другим пользователем во время выполнения транзакции.

### Независимость от аппаратного обеспечения

Используемые в настоящее время компьютеры характеризуются большим разнообразием. В связи с этим существует необходимость интеграции данных на всех системах и создания для пользователя представления единой системы. Должна иметься возможность запуска одной и той же СУБД на разном аппаратном обеспечении.

### Независимость от операционной системы

Эта цель является следствием предыдущей. Необходимо, чтобы одна и та же СУБД могла работать под управлением разных ОС.

### Независимость от сети

Если система в состоянии поддерживать несколько узлов с разным аппаратным обеспечением и разными операционными системами, то желательно, чтобы в ней поддерживались разные типы сетей.

### Независимость от СУБД

Эта цель означает, что желательно, чтобы распределенная БД допускала использование различных СУБД разными пользователями. Это возможно только если эти СУБД поддерживают некоторый общий стандарт представления данных, например, официальный стандарт языка SQL.