Московский государственный университет экономики, статистики и информатики (МЭСИ)

«Базы данных»
Лекция 8-9
Индексы. Архитектуры.

Начальник отдела НИЧ, к.э.н., доцент Д.Г. Корнеев

2009 год

Использование индексов. Методы доступа к данным

- Основная причина, побуждающая к постоянному совершенствованию всей технологии организации структур хранения и методов доступа, состоит в том, что характеристики доступа к диску намного хуже по сравнению с соответствующими характеристиками доступа к оперативной памяти. (Как правило, значения времени поиска составляют порядка 5 или 6 мс, частоты вращения — до 10000 об/мин, а скорость передачи данных обычно находится в пределах 5—10 Мбайт/с, поэтому, как правило, в любой конкретной системе обмен данными с оперативной памятью происходит по меньшей мере на четыре или пять порядков быстрее по сравнению с диском).
- **Таким образом, наиболее важное направление повышения производительности состоит в уменьшении до минимума количества операций доступа к диску (или дисковых операций ввода—вывода).**

Использование индексов. Методы доступа к данным.

Далее рассматриваются методы, способствующие достижению этой цели, иными словами, методы такого размещения данных на диске, чтобы любой необходимый фрагмент данных, скажем, некоторую конкретную запись можно было найти за минимально возможное количество операций ввода—вывода.

- Определение: конкретное размещение данных на диске называется **структурой хранения.**
- В решении задачи поиска конкретного фрагмента данных в базе данных и передачи его пользователю участвует несколько различных уровней программного обеспечения. Безусловно, подробности устройства этих уровней в значительной степени зависят от конкретной системы (к тому же в разных системах часто применяется различная терминология), но используемые при этом принципы являются довольно стандартными, и эти принципы кратко описаны ниже.

Использование индексов. Методы доступа к данным.

- Шаг 1. Вначале СУБД определяет, какая ей требуется запись, и передает **диспетчеру файлов запрос на выборку этой записи.** (В целях этого простого описания предполагается, что СУБД обладает способностью заблаговременно и точно определять, какая именно запись ей потребуется. На практике чаще всего возникает необходимость сделать выборку набора из нескольких записей и выполнить поиск среди этих записи в оперативной памяти, чтобы найти ту конкретную запись, которая действительно требуется. Но, в принципе, это означает лишь то, что последовательность шагов 1—3 иногда приходится повторять для каждой записи из этого набора.)
- Шаг 2. Диспетчер файлов в свою очередь определяет, какая страница содержит требуемую запись, и передает **диспетчеру диска запрос на выборку этой страницы.**
- Шаг 3. Наконец, диспетчер диска определяет физическое местонахождение желаемой страницы на диске и выдает необходимый запрос на выполнение операции ввода-вывода на диске.

Использование индексов. Методы доступа к данным.

- Безусловно, что иногда требуемая страница может уже находиться в буфере оперативной памяти в результате ранее выполненной операции выборки, и в этом случае, безусловно, необходимость повторно осуществлять ее выборку не возникает.
- Поэтому, выражаясь неформально, СУБД обладает представлением о базе данных как о коллекции записей, и это представление поддерживается диспетчером файлов; диспетчер файлов, в свою очередь, обладает представлением **о базе данных как о коллекции страниц, и это представление поддерживается** диспетчером диска, а диспетчер диска обладает таким представлением о диске, "каким диск является в действительности". Далее эти неформальные определения рассматриваются более подробно.

Использование индексов. Методы доступа к данным.

Использование индексов. Методы доступа к данным. Кластеризация.

- Это обзорное описание не будет полным без краткого упоминания такой темы, **как кластеризация данных**. Основная идея, лежащая в основе кластеризации, заключается в том, что необходимо обеспечить ***хранение логически связанных записей (которые поэтому часто используются совместно) в непосредственной близости друг от друга на физическом диске.*** Физическая кластеризация данных играет исключительно важную роль с точки зрения производительности, в чем можно легко убедиться

  на следующем примере. Предположим, что последней по времени записью, к которой был выполнен доступ, является Rl, а следующая требуемая запись— это R2. Допустим также, что Rl хранится на странице P1,

  а R2 — на странице P2.

  В таком случае справедливы приведенные ниже утверждения.

- 1. Если P1 и P2 представляют собой одну и ту же страницу, то для доступа к R2 вообще не потребуется каких-либо физических операций ввода—вывода, поскольку необходимая страница p2 уже будет находиться в буфере оперативной памяти.

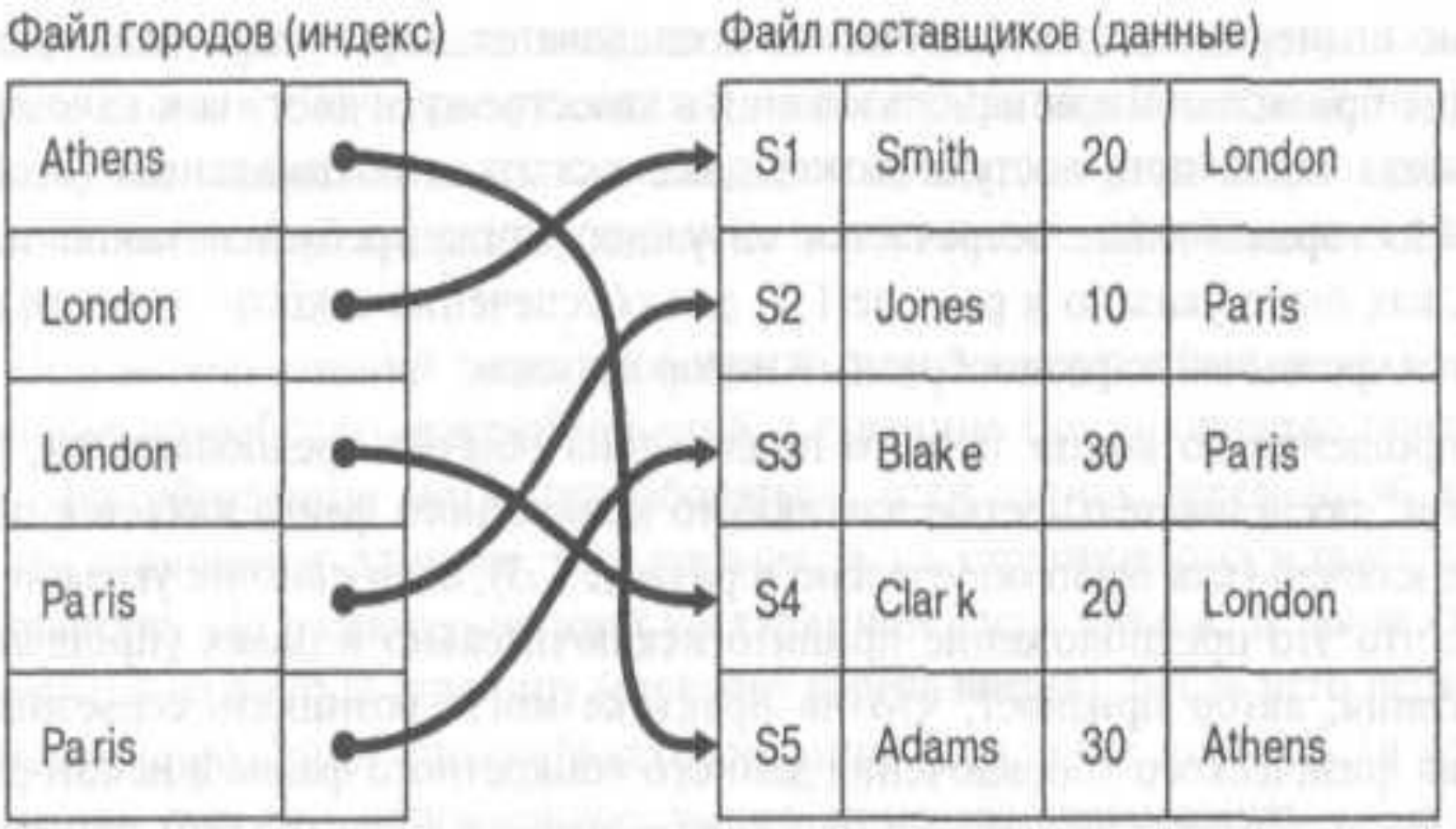
Использование индексов. Методы доступа к данным. Кластеризация.

- **2.** Если страницы p1 и p2 являются разными, но расположенными на физическом устройстве достаточно близко друг к другу (в частности, если они являются физически смежными), то для доступа к r2 потребуется физическая операция ввода—вывода (безусловно, за исключением того случая, что p2 также будет находиться в буфере оперативной памяти), но время поиска, требуемое в этой операции ввода—вывода, будет предельно сокращено, поскольку головки чтения-записи уже должны находиться недалеко от нужной позиции на диске. В частности, время перехода с дорожки на дорожку будет равно нулю, если страницы p1 и p2 находятся на одном и том же цилиндре.

Использование индексов

Рассмотрим данные о поставщиках. Предположим, что одним из самых важных (т.е. часто выполняемых и поэтому требующих высокой производительности) является запрос: ***"Определить всех поставщиков из города c" (где c — формальный параметр).*** С учетом такого требования администратор базы данных может выбрать хранимое представление, показанное на следующем слайде. В этом представлении применяются два файла — файл поставщиков и файл городов файл городов, который предполагается хранить в последовательности городов (поскольку CITY — первичный ключ), включает указатели (идентификаторы записей) на файл поставщиков.

Использование индексов



Использование индексов

- 1. Выполнить поиск во всем файле поставщиков, отбирая те записи, в которых значение города равно London.
- 2. Найти в файле городов элементы со значением London и после обнаружения каждого такого элемента перейти по указателю к соответствующей записи в файле поставщиков.
- Если относительное количество поставщиков из Лондона по сравнению с другими невелико, то вторая из этих стратегий, по-видимому, будет более эффективной, чем первая, поскольку, во-первых, в СУБД имеется информация о физическом упорядочении файла городов (и поиск в этом файле может быть прекращен сразу после обнаружения города, который следует за Лондоном в алфавитном порядке), и, во-вторых, даже если и придется выполнить поиск во всем файле городов, этот поиск все равно, скорее всего, потребует в целом меньшего количества операций ввода-вывода, поскольку файл городов имеет меньшие физические размеры, чем файл поставщиков (в связи с тем, что записи в нем короче).

Использование индексов

- В данном примере файл городов может рассматриваться как индекс ("индекс CITY") к файлу поставщиков; равным образом эту мысль можно выразить так, что файл поставщиков проиндексирован с помощью файла городов. Таким образом, индекс представляет собой **файл особого рода.** А именно, индекс — это файл, каждый элемент (т.е. каждая запись) которого состоит точно из двух значений:
- значения данных и указателя (идентификатора записи); ***значением данных является значение некоторого поля индексированного файла, а указатель определяет запись в этом файле, имеющую такое же значение этого же поля.***

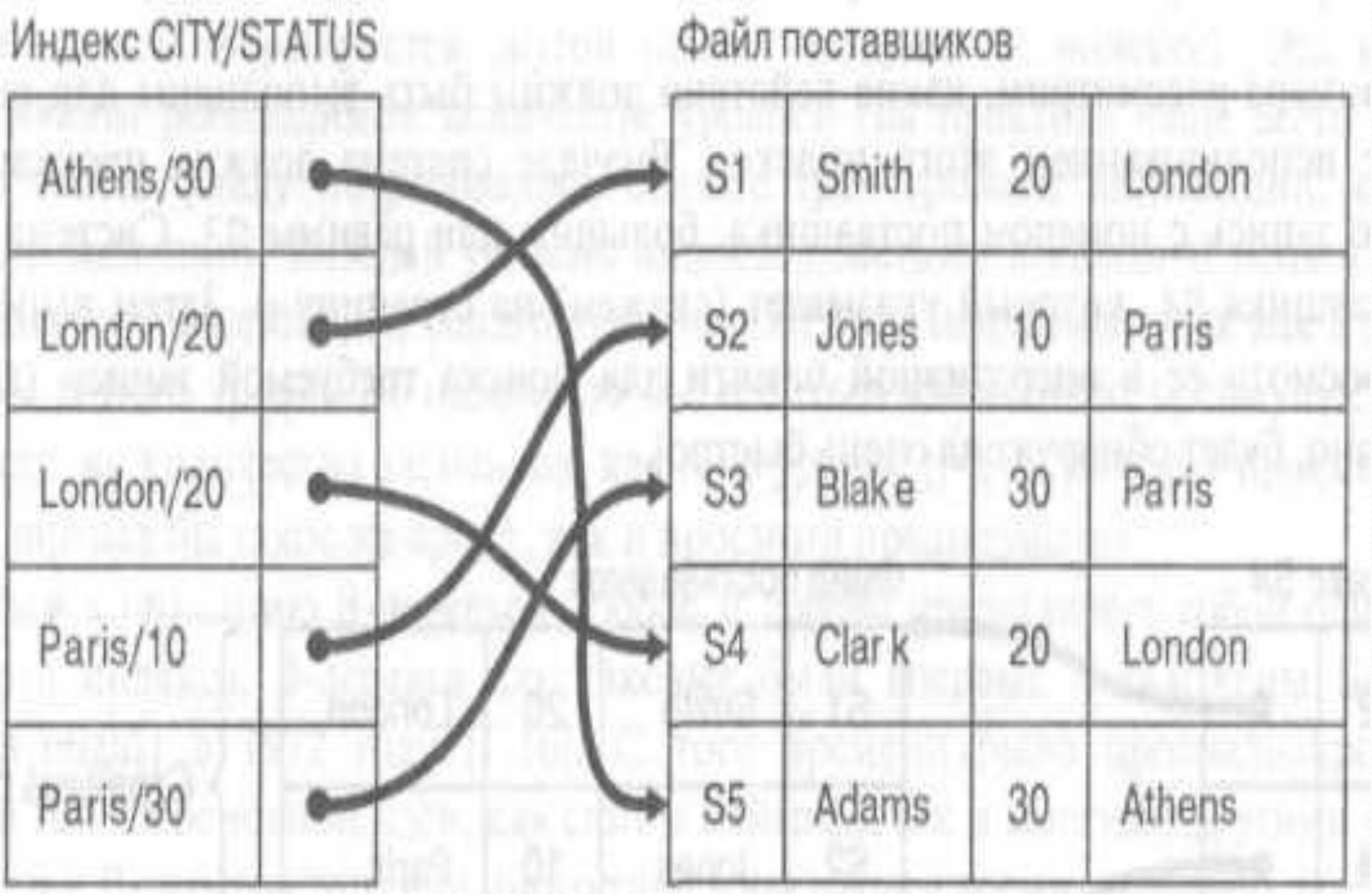
Использование индексов

- Фундаментальным преимуществом любого индекса по сравнению с другими путями доступа является то, что он ускоряет поиск. ***Но применение индексов связано также с определенным недостатком — они замедляют операции обновления.***
- Например, после вставки каждой новой записи в индексированный файл необходимо также вводить новый элемент в индекс. В качестве более конкретного примера достаточно представить себе, какие действия СУБД должны выполнить над индексом CITY, если поставщик S2 переезжает, скажем, из Парижа в Лондон.
- Поэтому, вообще говоря, рассматривая возможность использования некоторого поля в качестве кандидата для индексации, необходимо прежде всего найти ответ на такой вопрос: "Что важнее, эффективная выборка на основе значений рассматриваемого поля или издержки обновления, связанные с обеспечением такой эффективной выборки?".

Использование индексов

- Существует также возможность сформировать индекс ***на основе значений двух или нескольких полей, составляющих единую комбинацию.***
- Например, на следующем рис. показан индекс на файл поставщиков, в котором используется комбинация полей CITY и STATUS в указанном порядке. С применением такого индекса в СУБД можно получить ответ на запрос: *"Определить поставщиков из Парижа со статусом 30"* за один просмотр единственного индекса. А если бы этот комбинированный индекс был заменен двумя отдельными индексами, то для выполнения такого запроса потребовалось бы два отдельных просмотра индексов.

Использование индексов



Операторы SQL

Создание индекса

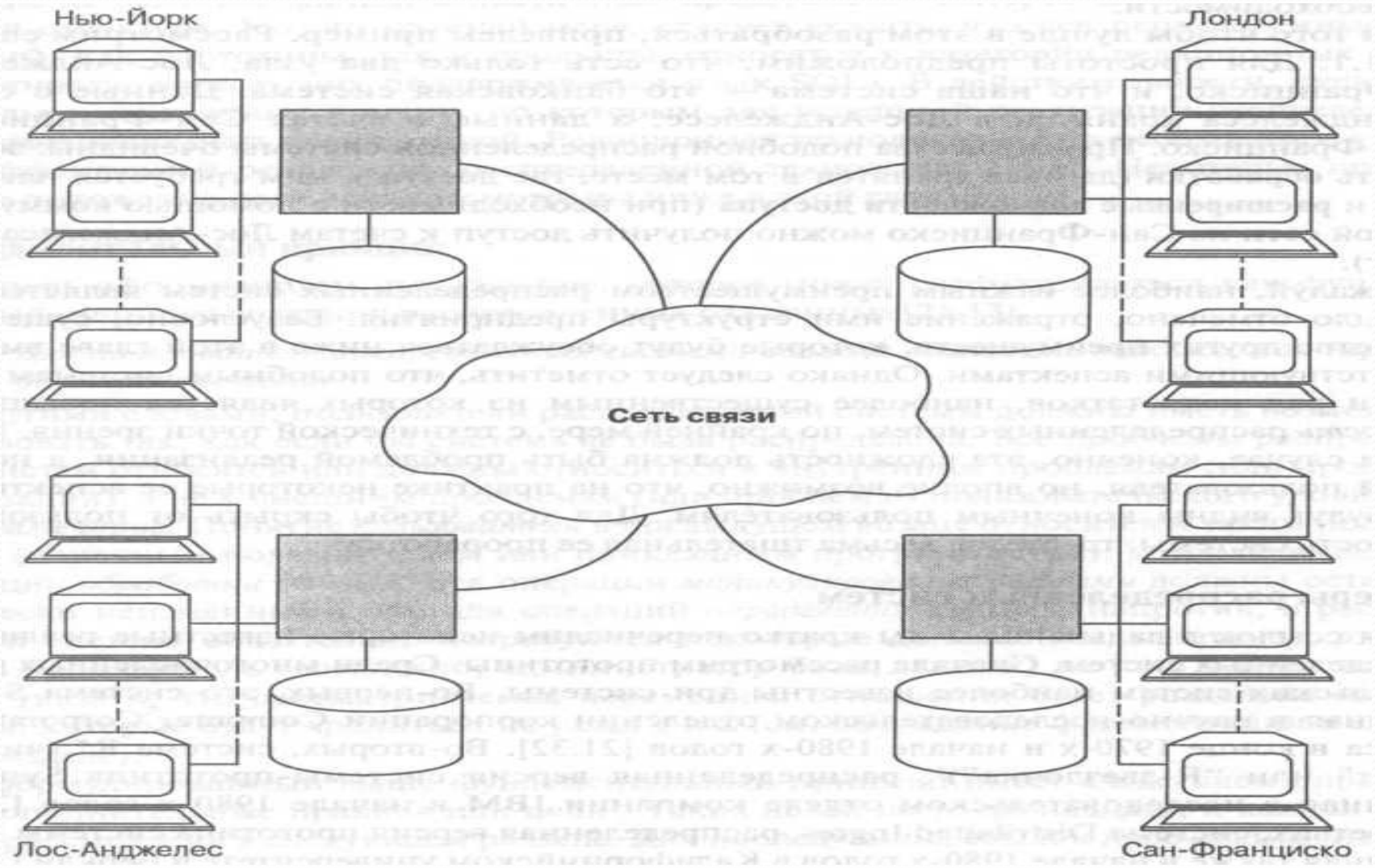
CREATE INDEX <имя индекса>
ON <имя таблицы> (<имя столбца 1> ,
<имя сттолбца 2> ,
……..);

Удаление индекса
DROP INDEX <имя индекса>;

Распределенные базы данных

- Система распределенных баз данных состоит из набора **узлов (site), связанных** коммуникационной сетью, в которой:
- а) каждый узел — это полноценная СУБД сама по себе, но
- б) узлы взаимодействуют между собой таким образом, что пользователь любого из них может получить доступ к любым данным в сети так, как будто они находятся на его собственном узле.
- Из этого определения следует, что так называемая *распределенная база данных в дейст*вительности представляет собой *виртуальную базу данных, компоненты которой физичес*ки хранятся в нескольких различных *реальных базах данных на нескольких различных* узлах (в сущности, являясь логическим объединением этих реальных баз данных).

Распределенные базы данных



Распределенные базы данных

- Зачем нужны распределенные базы данных? Основная причина заключается в том, что сами предприятия обычно уже распределены, по крайней мере, логически, т.е. разбиты на подразделения, отделы, рабочие группы и т.д. Очень часто они распределены и физически, т.е. разделены на отдельно расположенные заводы, фабрики, лаборатории и т.д.
- Из этого следует, что данные также обычно распределены, поскольку каждая организационная единица на предприятии создает и обрабатывает собственные данные, относящиеся к ее деятельности. Таким образом, информация предприятия разбивается на отдельные автономные части, которые иногда называют *островами информации. А рас*пределенная система обеспечивает *мосты для их соединения в единое целое.*
- Иначе говоря, распределенная система позволяет структуре базы данных ***отображать структуру предприятия*** — локальные данные могут храниться локально, в соответствии с логической принадлежностью, тогда как к удаленным данным доступ может осуществляться по мере необходимости.

Распределенные базы данных

- Для того чтобы лучше в этом разобраться, приведем пример. Предположим, что есть только два узла, Лос-Анджелес и Сан-Франциско, и что наша система — это банковская система. Данные о счетах Лос-Анджелеса хранятся в Лос-Анджелесе, а данные о счетах Сан-Франциско — в Сан-Франциско.
- Преимущества подобной распределенной системы очевидны: **эффективность обработки** (данные хранятся в том месте, где доступ к ним требуется наиболее часто) и **расширенные возможности доступа** (при необходимости с помощью коммуникационной сети из Сан-Франциско можно получить доступ к счетам Лос-Анджелеса и наоборот).
- Пожалуй, наиболее важным преимуществом распределенных систем является, как уже было отмечено, **отражение ими структуры предприятия.**

Распределенные базы данных

- **Принцип локальной независимости**

Узлы в распределенной системе должны быть **независимы, или автономны.** Локальная независимость означает, что все операции на узле контролируются этим узлом. Никакой узел X не должен зависеть от некоторого узла Y, чтобы успешно функционировать (иначе, если узел Y будет отключен, узел X не сможет функционировать, даже если на самом узле X будет все в порядке; возникновение таких ситуаций, безусловно, нежелательно).

Распределенные базы данных

- **Принцип отсутствия зависимости от центрального узла**

Локальная независимость предполагает, что **все узлы в распределенной системе должны рассматриваться как равные.** Поэтому, в частности, не должно быть никаких обращений к *центральному, или главному, узлу для получения некоторой централизованной*

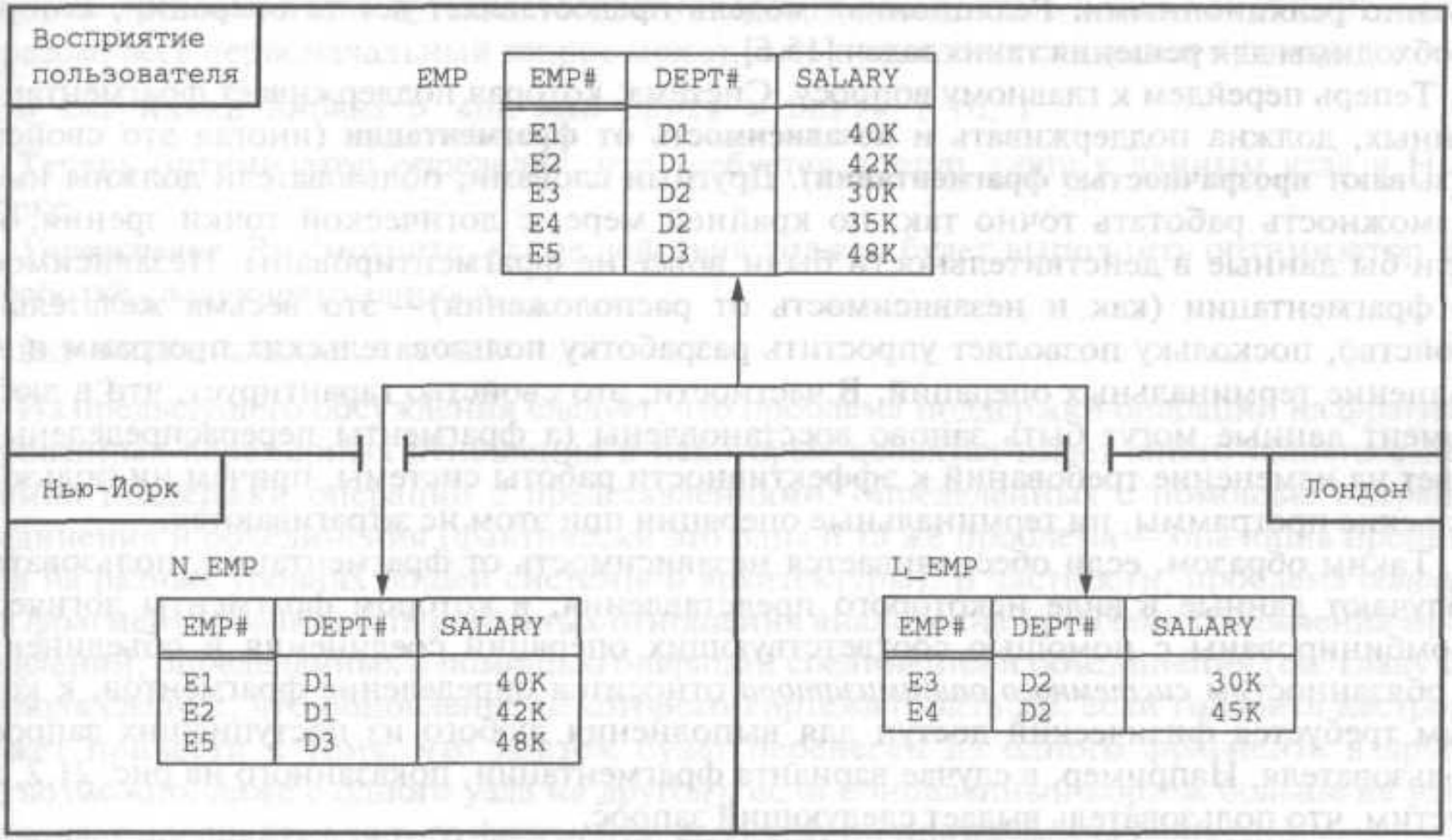
услуги. Не должно быть, например, централизованной обработки запросов, централизованного управления транзакциями или централизованной службы присваивания имен, **поскольку в таких случаях система в целом будет зависимой от центрального узла.**

Распределенные базы данных

- **Принцип независимости от фрагментации**

Система поддерживает **независимость данных от фрагментации, если некоторая пере**менная отношения может быть разделена на части, или *фрагменты, при организации ее* физического хранения, а различные фрагменты могут храниться на разных узлах. Фрагментация желательна для повышения производительности системы. В этом случае данные могут храниться в том месте, где они чаще всего используются, что позволяет достичь локализации большинства операций и уменьшения сетевого трафика.

Распределенные базы данных



Распределенные базы данных

К обязанностям *системного оптимизатора относится определение фрагментов,* к которым требуется физический доступ для выполнения любого из поступивших запросов пользователя. Например, в случае варианта фрагментации, показанного на предыдущем рис., допустим, что пользователь выдает следующий запрос:

SELECT *

FROM EMP

WHERE SALARY > 40K AND DEPT# = ' Dl ‘;

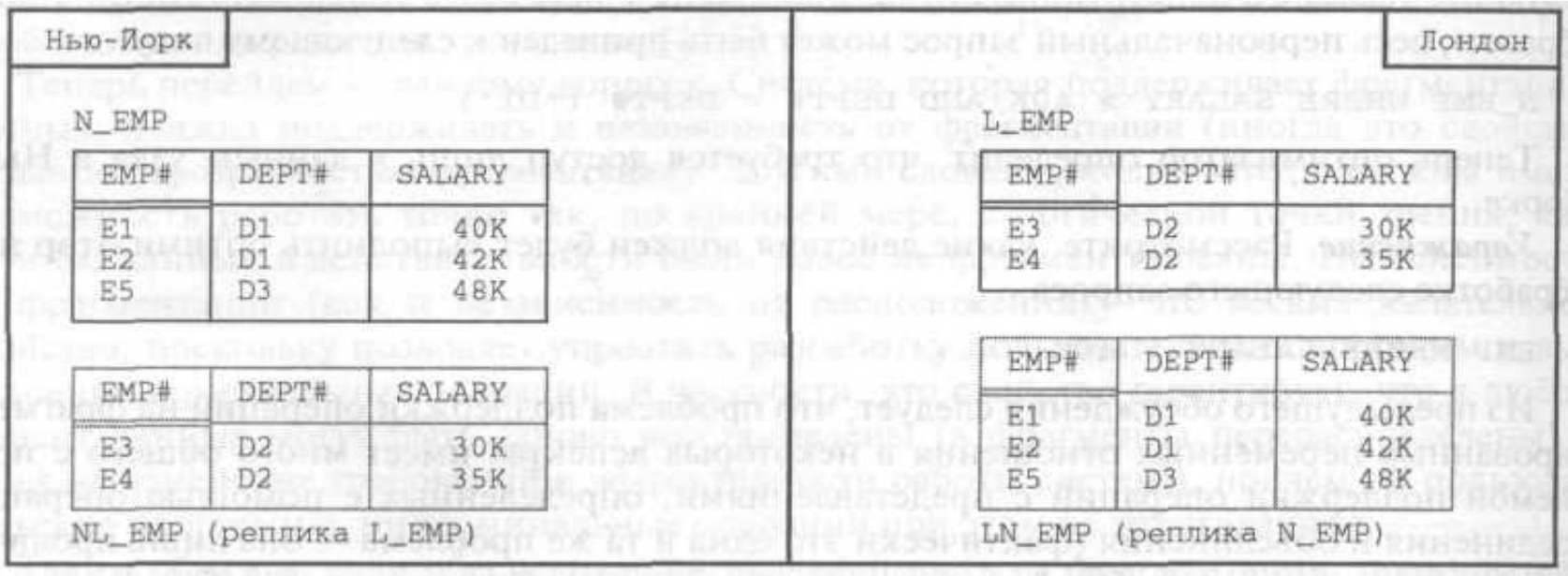
Из определений фрагментов (которые хранятся, конечно же, в каталоге; оптимизатору должно быть известно, что весь требуемый результат может быть получен только по данным узла в Нью-Йорке, а значит, нет никакой необходимости обращаться к узлу в

Лондоне.

Распределенные базы данных

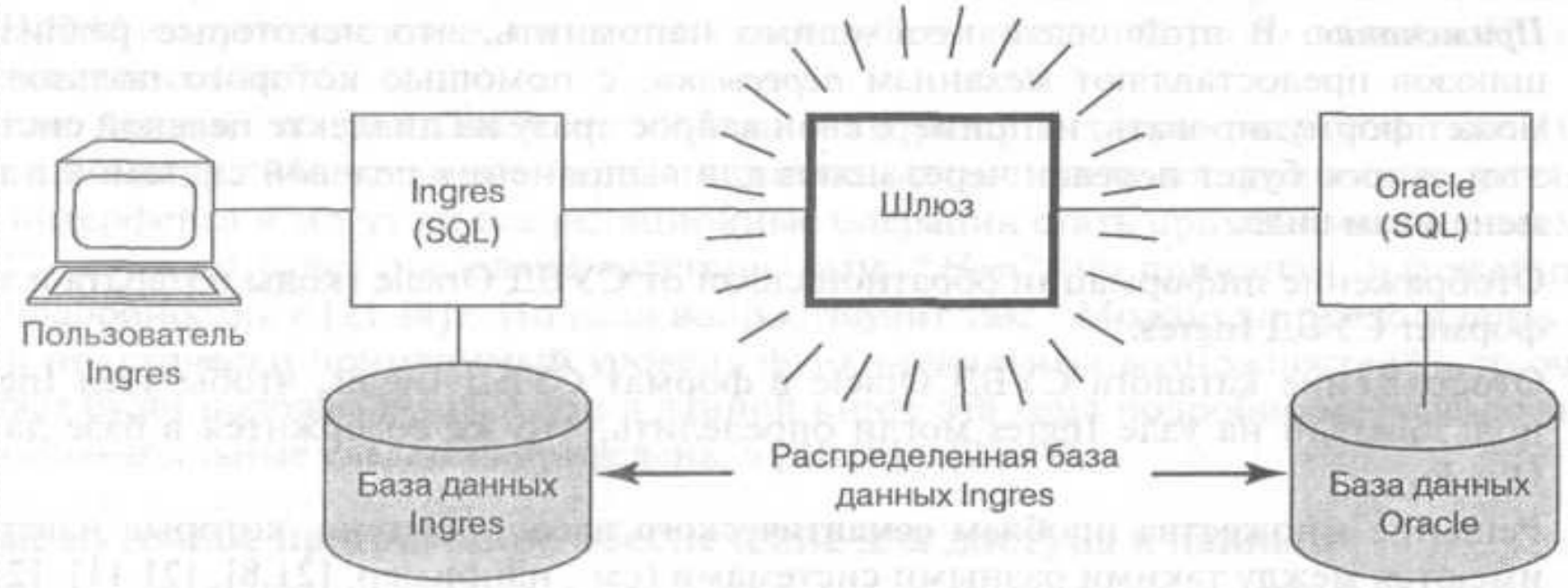
Принцип независимости от репликации

Система поддерживает **репликацию данных,** если отношение (или в общем случае данный *фрагмент отношения*) может быть представлен несколькими отдельными копиями, или *репликами, которые* хранятся на нескольких отдельных узлах.



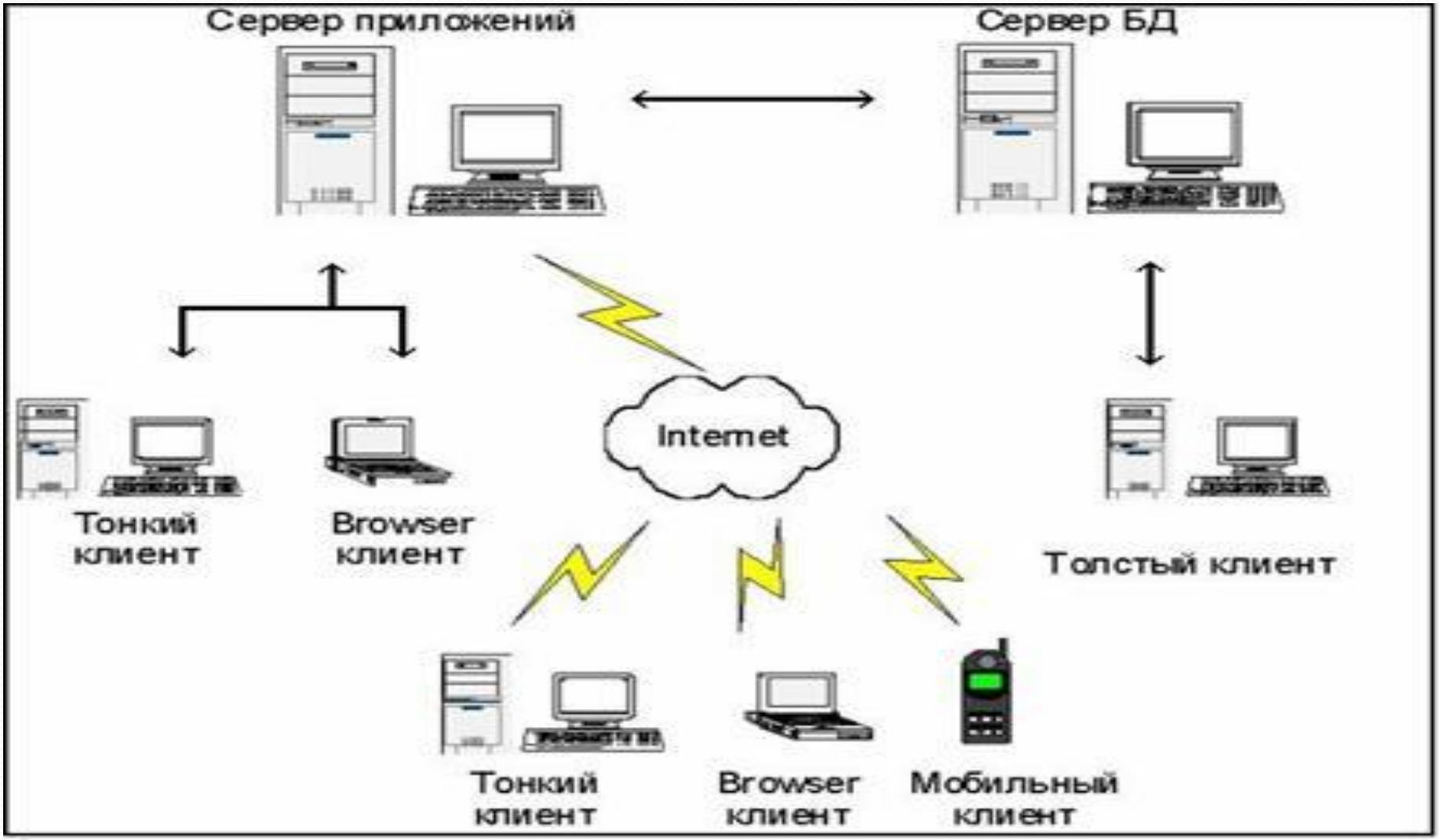
Связи между БД

Возможный шлюз между разными БД.
В ORACLE для организации доступа от одной к другой БД ORACLE используется DBLINK.



Архитектура клиент-сервер

Трехуровневая архитектура



Трехуровневая архитектура

- ***Терминал*** - это интерфейсный (обычно графический) компонент, который представляет первый уровень, собственно приложение для конечного пользователя. Первый уровень ***не должен*** иметь прямых связей с базой данных (по требованиям безопасности), быть нагруженным основной бизнес-логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надежности). На первый уровень может быть вынесена и обычно выносится простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.

Трехуровневая архитектура

- *Сервер приложений* располагается на втором уровне. На втором уровне сосредоточена большая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы (см.выше), а также погруженные в третий уровень хранимые процедуры и триггеры.

- *Сервер базы данных* обеспечивает хранение данных и выносится на третий уровень. Обычно это стандартная реляционная или объектно-ориентированная СУБД. Если третий уровень представляет собой базу данных вместе с хранимыми процедурами, триггерами и схемой, описывающей приложение в терминах реляционной модели, то второй уровень строится как программный интерфейс, связывающий клиентские компоненты с прикладной логикой базы данных.