Модели «клиент-сервер» в технологии распределенных баз данных

1. Модель удаленного управления данными. Модель файлового сервера.
2. Модель удаленного доступа к данным.
3. Модель сервера баз данных.

Модели «клиент—сервер» в технологии баз данных

- Термин **«клиент-сервер»** применяется к архитектуре программного обеспечения, функционирующего по принципу взаимодействия двух программных процессов, один из которых в этой модели называется **«клиентом»,** а другой - **«сервером».**
- Клиентский процесс запрашивает услуги, а серверный процесс обеспечивает их выполнение.

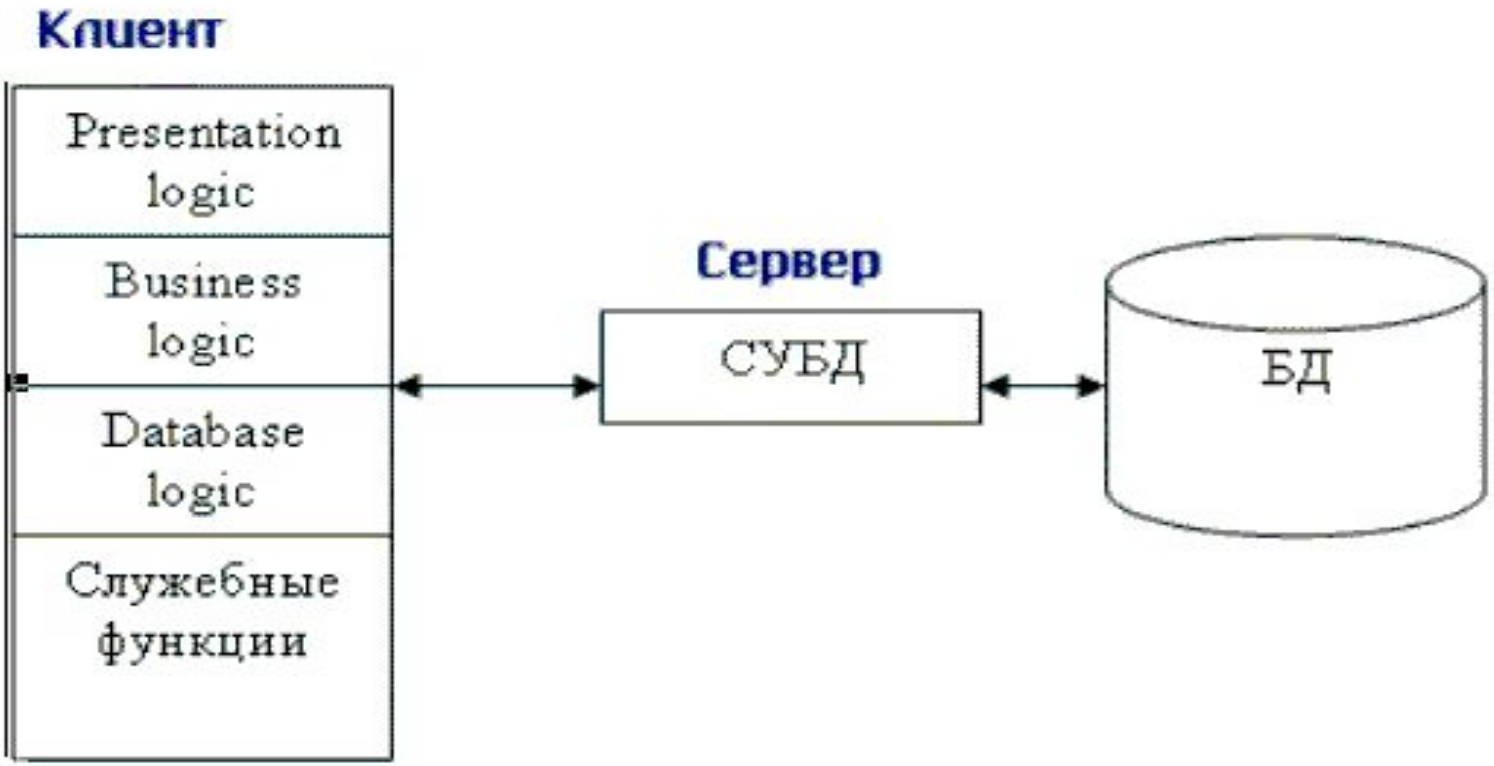
При этом предполагается, что один серверный процесс может обслужить множество клиентских процессов.

- Ранее приложение (пользовательская программа) не разделялась на части, а выполнялось монолитным блоком.
  Но возникла идея более рационального использования ресурсов сети, т.к. при монолитном исполнении приложения используются ресурсы только одного компьютера, а остальные компьютеры в сети рассматриваются как терминалы.
- В отличие от эпохи main-фреймов, все компьютеры в сети обладают собственными ресурсами, и разумно так распределить нагрузку на них, чтобы максимальным образом использовать их ресурсы.

Основной принцип технологии «клиент—сервер» применительно к технологии баз данных заключается в разделении функций стандартного интерактивного приложения **на 5 групп**, имеющих различную природу:

- функции ввода и отображения данных (Presentation Logic);
- прикладные функции, определяющие основные алгоритмы решения задач приложения (Business Logic);
- функции обработки данных внутри приложения (Database Logic);
- функции управления информационными ресурсами (Database Manager System);
- служебные функции, играющие роль связок между функциями первых четырех групп.

Структура типового интерактивного приложения, работающего с базой данных



Презентационная логика

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, когда работает приложение:

✔ интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения;

✔результаты решения некоторых промежуточных задач;

✔справочная информация.

Основными задачами презентационной логики:

- формирование экранных изображений;
- чтение и запись в экранные формы информации;
- управление экраном;
- обработка движений мыши и нажатий клавиш клавиатуры.

- Организация презентационной логики приложений обеспечивается знако-ориентированным пользовательским интерфейсом, задаваемым моделями:
  - ✔ CICS (Customer Control Information System);
  - ✔ IMS/DC фирмы IBM;
  - ✔ TSO (Time Sharing Option) для централизованной main-фреймовой архитектуры;
  - ✔ GUI - графического пользовательского интерфейса, модель поддерживается в операционных средах Microsoft's Windows, Windows NT, в OS/2 Presentation Manager.

Бизнес-логика

Бизнес-логика, или логика приложений (Business processing Logic), - часть кода приложения, определяющая алгоритмы решения конкретных задач приложения.

❖Код пишется на языках программирования:

✔ C;

✔ C++;

✔ Cobol;

✔SmallTalk;

✔Visual-Basic.

Логика обработки данных

- **Логика обработки данных** (Data manipulation Logic) - часть кода приложения, связанная с обработкой данных внутри приложения.
- **Данными управляет СУБД (DBMS).**
- Для обеспечения доступа к данным используются язык запросов и средства манипулирования данными стандартного языка SQL.

Database Manager System Processing

- Процессор управления данными (Database Manager System Processing) - СУБД, обеспечивающая хранение и управление базами данных.
- ❖Функции СУБД скрыты от бизнес-логики приложения, но для рассмотрения архитектуры приложения они выделяются в отдельную часть приложения.

Централизованная архитектура

- В централизованной архитектуре (Host-based processing) эти части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

Модели распределений

В децентрализованной архитектуре эти задачи по-разному распределены между серверным и клиентским процессами.

❖В зависимости от характера распределения выделяют модели распределений:

- ✔ распределенная презентация (Distribution presentation, DP);
- ✔ удаленная презентация (Remote Presentation, RP);
- ✔ распределенная бизнес-логика (Remote business logic, RBL);
- ✔ распределенное управление данными (Distributed data management, DDM);
- ✔ удаленное управление данными (Remote data management, RDA).

Распределение функций приложения в моделях «клиент -сервер»

Двухуровневые модели

- Двухуровневая модель - результат распределения пяти функций между двумя процессами, выполняемыми на двух платформах:
  - ✔на клиенте;
  - ✔на сервере.

Модель удаленного управления данными. Модель файлового сервера

Модель удаленного управления данными, или модель файлового сервера (File Server, FS).

❖В этой модели на клиенте располагается:

✔ презентационная логика;

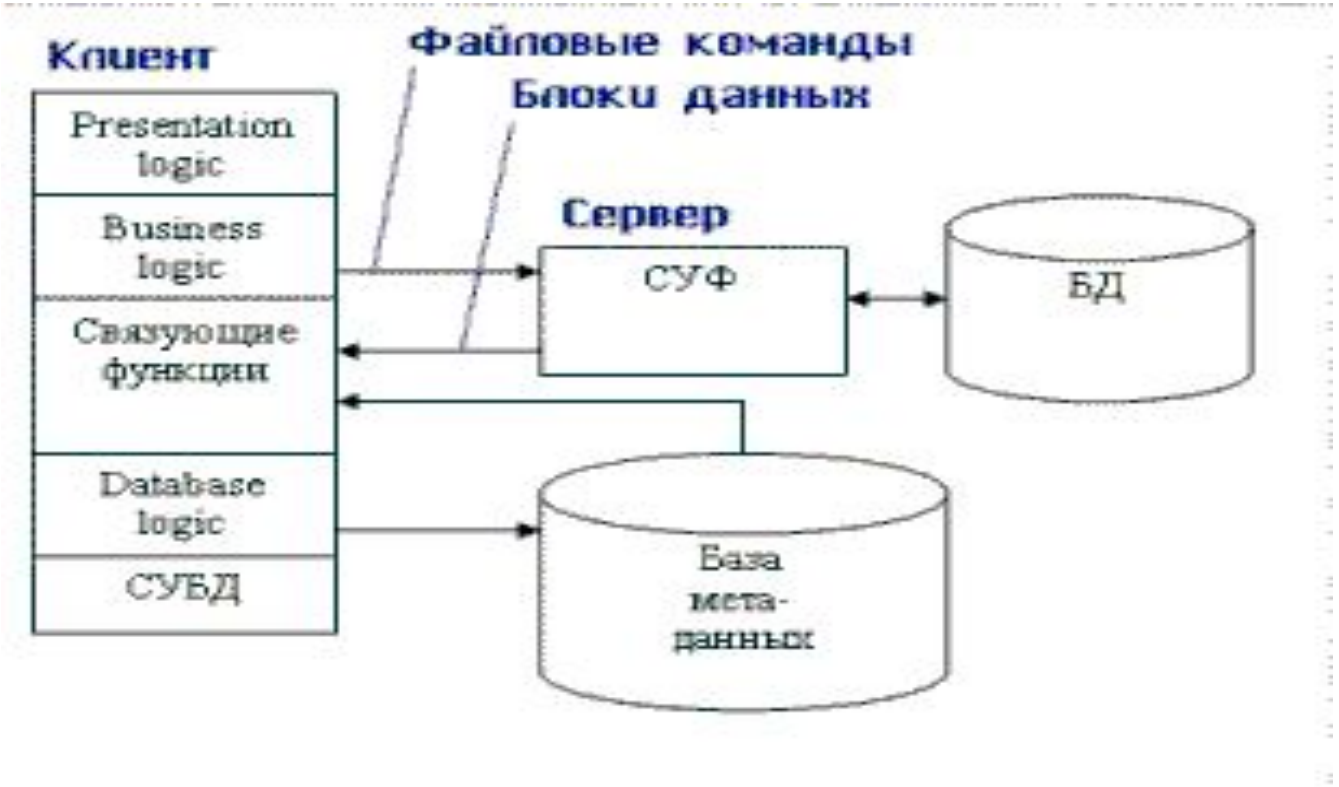
✔ бизнес-логика;

✔функции управления информационными ресурсами.

❖На сервере располагаются:

✔ файлы с данными и поддерживается доступ к этим файлам.

Модель файлового сервера



Модель файлового сервера

- В этой модели файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами (база мета-данных), находится на клиенте.

Модель файлового сервера

Достоинства модели:

✔ разделение монопольного приложения на два взаимодействующих процесса;

✔ сервер (серверный процесс) обслуживает множество клиентов, которые обращаются к нему с запросами;

✔ отсутствие высоких требований к производительности сервера (главное – требуемый объем дискового пространства).

❖ СУБД находится в этой модели на клиенте.

❖ Программные компоненты СУБД не распределены, т.е. никакая часть СУБД на сервере не инсталлируется и не размещается.

Алгоритм выполнения запроса клиента в модели файлового сервера

1).Клиент формулирует запрос в командах ЯМД.

2).СУБД переводит этот запрос в последовательность файловых команд.

3).Каждая файловая команда вызывает перекачку блока информации на клиента. С помощью функций операционной системы в оперативную память клиентской установки на время сеанса работы копируется файлы базы данных.

4). На клиенте СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации .

- Перекачка информации с сервера на клиент производится до тех пор, пока не будет получен ответ на запрос клиента.

Недостатки модели файлового сервера

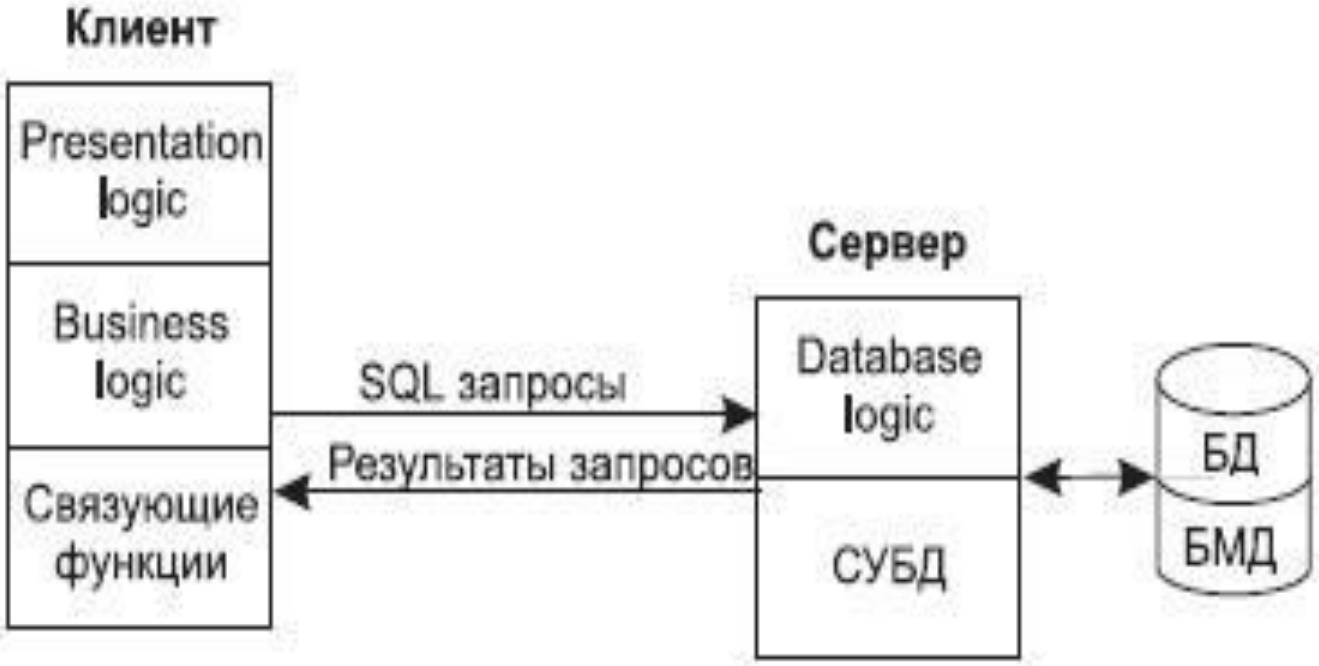
- ✔ высокий сетевой трафик, связанный с передачей по сети множества блоков и файлов, требуемых приложению;
- ✔ узкий спектр операций манипулирования с данными, который определяется только файловыми командами;
- ✔ отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы).

Модель удаленного доступа к данным

- В модели удаленного доступа (Remote Data Access, RDA) база данных хранится на сервере.
- На сервере находится и ядро СУБД.

  На клиенте располагается презентационная логика и бизнес-логика приложения.
- Клиент обращается к серверу с запросами на языке SQL.
- Компонент доступа реализуется в виде самостоятельной программной части СУБД, называемой *SQL-сервером*, и инсталлируется на вычислительной установке сервера системы.

Модель удаленного доступа к данным (RDA)



- Обращение клиента за сервисом управления данными происходит через среду передачи с помощью операторов языка SQL или вызова специальной библиотеки API (Application Programming Interface – интерфейс прикладного программирования).

Преимущества модели удаленного доступа

✔перенос компонента представления и прикладного компонента на клиентский компьютер разгрузил сервер БД;

✔процессор или процессоры сервера целиком загружаются операциями обработки данных, запросов и транзакций;

✔уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на SQL, и их объем существенно меньше.

В ответ на запросы клиент получает только данные, релевантные запросу, а не блоки файлов, как в FS-модели.

Модель удаленного доступа

- *Основное достоинство RDA-модели — унификация интерфейса «клиент-сервер», стандартом при общении приложения-клиента и сервера становится язык SQL.*

Недостатки модели удаленного доступа (RDA)

- ✔запросы на языке SQL при интенсивной работе клиентских приложений загружают сеть;
- ✔ так как в этой модели на клиенте располагается бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики дублируется;
- ✔сервер играет пассивную роль, т.к. функции управления информационными ресурсами выполняются на клиенте.
  - Например, если необходимо выполнять контроль страховых запасов товаров на складе, то каждое приложение, которое связано с изменением состояния склада, после выполнения операций модификации данных, имитирующих продажу или удаление товара со склада, должно выполнять проверку на объем остатка.

Модель сервера баз данных

 Для устранения недостатков модели удаленного доступа должны быть соблюдены следующие условия:

1. Необходимо, чтобы БД в каждый момент отражала текущее состояние предметной области, то есть данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.
2. БД должна отражать правила предметной области, законы, по которым она функционирует (business rules).
3. Необходим постоянный контроль за состоянием БД, отслеживание всех изменений и адекватная реакция на них: например, при уменьшении товарного запаса ниже допустимой нормы должна быть сформирована заявка конкретному поставщику на поставку соответствующего товара.

- Эту модель поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server.
- Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища.

 Модель сервера баз данных имеет вид:

Модель активного сервера БД

- В этой модели бизнес-логика разделена между клиентом и сервером.
- На сервере бизнес-логика реализована в виде хранимых процедур - специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД.
- Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все изменения в БД, которые в ней предусмотрены.
- Сервер возвращает клиенту данные, релевантные его запросу, требующиеся клиенту либо для вывода на экран, либо для выполнения части бизнес-логики, которая расположена на клиенте.
- Трафик обмена информацией между клиентом и сервером резко уменьшается.

- Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров.
- Триггеры являются частью БД.
- Ядро СУБД проводит мониторинг всех событий, которые вызывают созданные и описанные триггеры в БД, и при возникновении соответствующего события сервер запускает соответствующий триггер.
- Каждый триггер представляет собой некоторую программу, которая выполняется над базой данных.
- Триггеры могут вызывать хранимые процедуры.

- В этой модели сервер является активным, т.к. не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.
- И хранимые процедуры, и триггеры хранятся в словаре БД, они могут быть использованы несколькими клиентами, что уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.
- Для написания хранимых процедур и триггеров используется расширение стандартного языка SQL - встроенный SQL.

Недостатки модели серверов БД

Недостаток модели - большая загрузка сервера, т.к. сервер обслуживает множество клиентов и выполняет следующие функции:

- ✔ осуществляет мониторинг событий, связанных с описанными триггерами;
- ✔ обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
- ✔ обеспечивает исполнение внутренней программы каждого триггера;
- ✔ запускает хранимые процедуры по запросам пользователей;
- ✔ запускает хранимые процедуры из триггеров;
- ✔ возвращает требуемые данные клиенту;
- ✔ обеспечивает все функции СУБД: доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД.

- Если переложить на сервер большую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшаются.
- Эту модель называют моделью с «тонким клиентом», в отличие от предыдущих моделей, где на клиента возлагались гораздо более серьезные задачи. Эти модели называются моделями с «толстым клиентом».

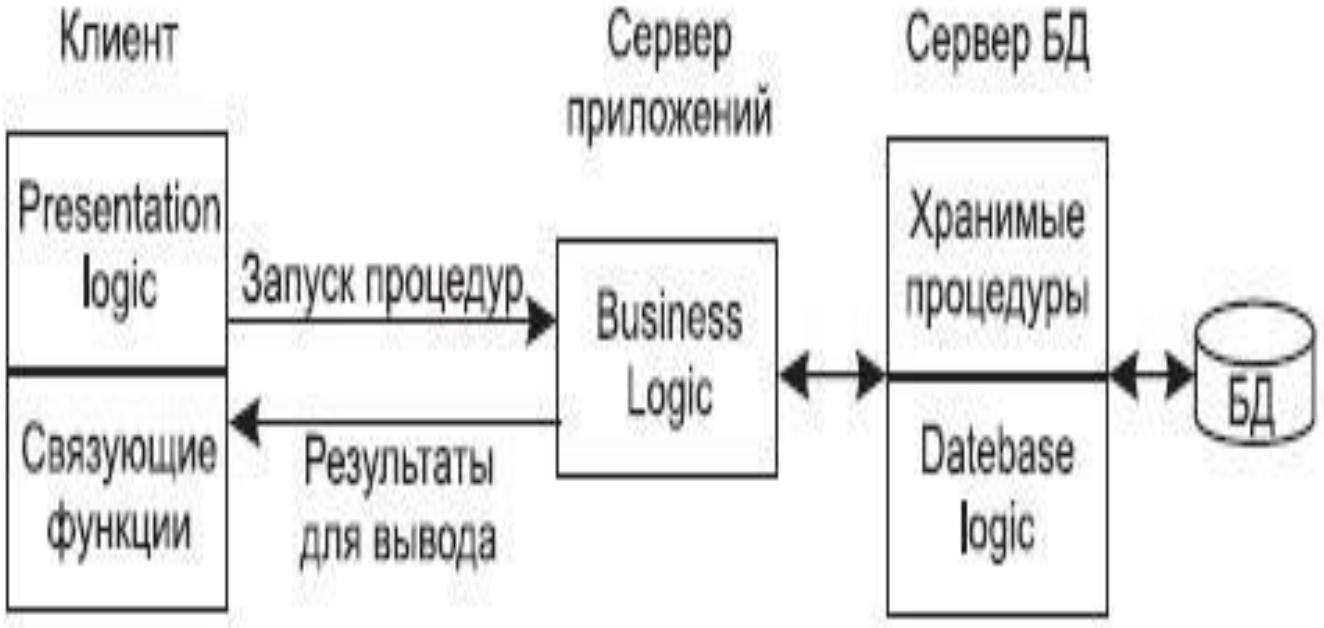
❖ Для разгрузки сервера была предложена *трехуровневая модель.*

Модель сервера приложений

- Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером.
- Этот промежуточный уровень содержит один или несколько серверов приложений.

❖ Архитектура трехуровневой модели имеет вид:

Модель сервера приложений



Клиент в модели сервера приложений

- В этой модели компоненты приложения делятся между тремя исполнителями:
  - *Клиент* обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на компьютере-клиенте.
  - Клиент исполняет коммуникационные функции front-end части приложения, обеспечивающие доступ клиенту в локальную или глобальную сеть.

Серверы приложений

- *Серверы приложений* составляют новый промежуточный уровень архитектуры.
- Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, поддерживают сетевую доменную операционную среду, хранят и исполняют наиболее общие правила бизнес-логики, поддерживают каталоги с данными, обеспечивают обмен сообщениями и поддержку запросов.

Серверы баз данных

 Серверы баз данных в этой модели занимаются исключительно функциями СУБД:

- ✔ обеспечивают функции создания и ведения БД;
- ✔ поддерживают целостность реляционной БД;
- ✔ обеспечивают функции хранилищ данных (warehouse services);
- ✔ создают резервные копии БД;
- ✔ восстанавливают БД после сбоев;
- ✔ управляют выполнением транзакций.

- Эта модель обладает большей гибкостью, чем двухуровневые модели.
- Заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных, которые относятся к области OLAP-приложений (On-line analytical processing.)
- В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках программирования, таких как C, C++, SmallTalk, Cobol, что повышает переносимость системы, ее масштабируемость.

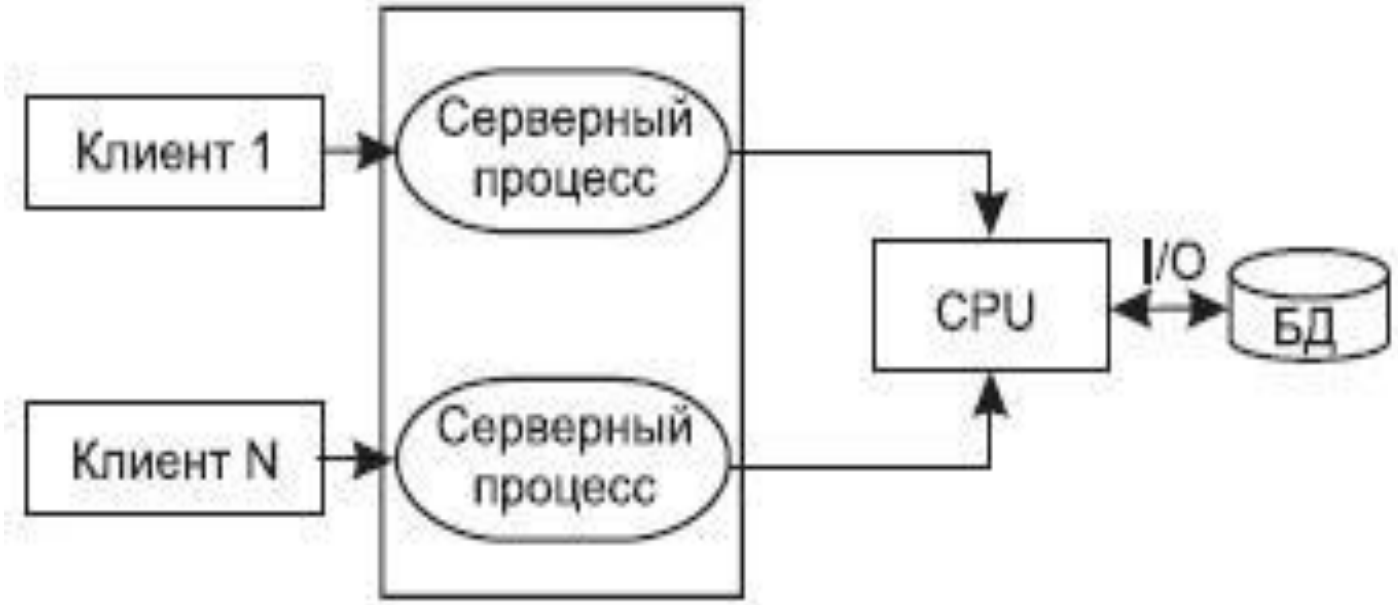
Модели серверов баз данных

- Изначально в архитектуре систем не было адекватного механизма организации взаимодействия процессов типа «клиент» и процессов типа «сервер».
- В современных же СУБД технология «клиент-сервер» является основополагающей и от эффективности ее реализации зависит эффективность работы системы в целом.
- Рассмотрим эволюцию технологии «клиент-сервер».

  В основном этот механизм определяется структурой реализации серверных процессов, и называется архитектурой сервера баз данных.

- Первоначально существовала модель, когда управление данными (функция сервера) и взаимодействие с пользователем были совмещены в одной программе.
- Это нулевой этап развития серверов БД.
- Затем функции управления данными были выделены в самостоятельную группу - сервер, однако модель взаимодействия пользователя с сервером соответствовала парадигме «один-к-одному», то есть сервер обслуживал запросы только одного пользователя (клиента), и для обслуживания нескольких клиентов нужно было запустить эквивалентное число серверов.

Взаимодействие серверных и клиентских процессов в модели «один - к – одному»



CPU (Central Processing Unit)- центральное обрабатывающее устройство.

- Выделение сервера в отдельную программу было революционным шагом, позволившим поместить сервер на одну машину, а программный интерфейс с пользователем - на другую, осуществляя взаимодействие между ними по сети.
- Однако необходимость запуска большого числа серверов для обслуживания множества пользователей ограничивала возможности такой системы.

- Для обслуживания большого числа клиентов на сервере должно быть запущено большое количество одновременно работающих серверных процессов, что повышало требования к ресурсам ЭВМ.
- Кроме того, каждый серверный процесс в этой модели запускался как независимый, поэтому если один клиент сформировал запрос, который был только что выполнен другим серверным процессом для другого клиента, то запрос выполнялся повторно.
- В такой модели сложно обеспечить взаимодействие серверных процессов.
- Эта модель самая простая, и исторически она появилась первой.

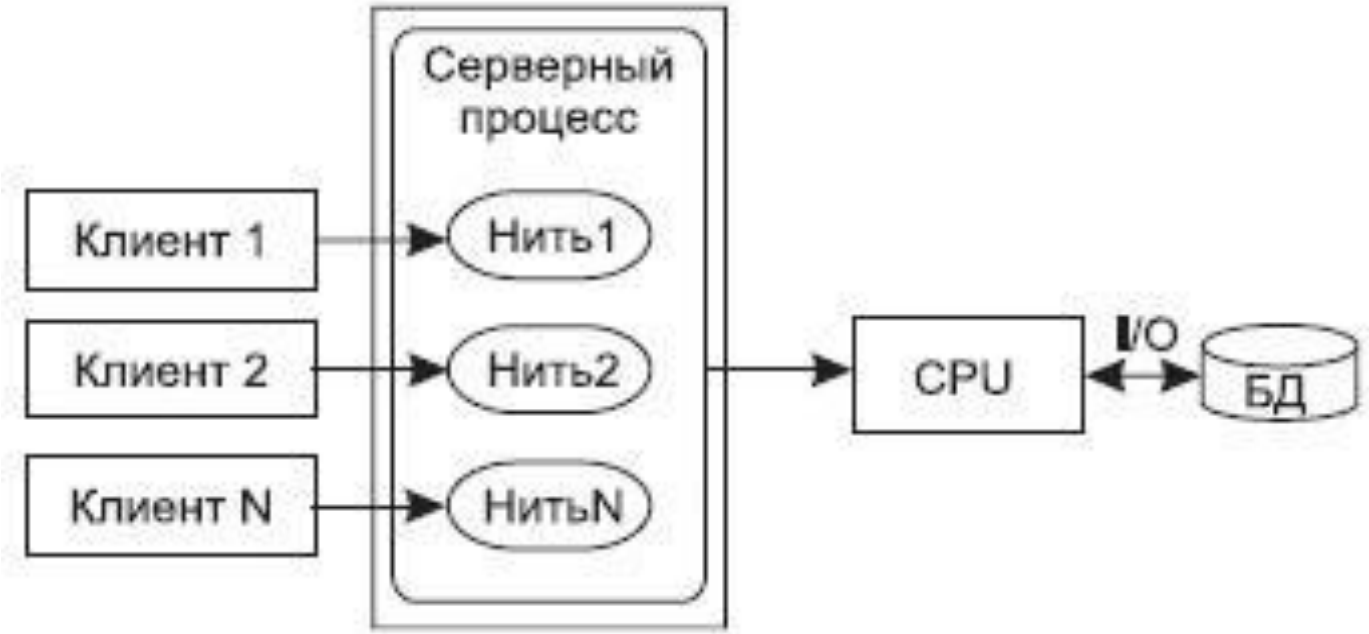
Многопотоковая односерверная архитектура

- Проблемы, возникающие в модели «один-к-одному», решаются в архитектуре «систем с выделенным сервером», обрабатывающий запросы от многих клиентов.
- Сервер единственный обладает монополией на управление данными и взаимодействует одновременно со многими клиентами.

Многопотоковая односерверная архитектура

- Логически каждый клиент связан с сервером отдельной нитью («thread»), или потоком, по которому пересылаются запросы.
- Такая архитектура называется многопотоковой односерверной («multi-threaded») и позволяет уменьшить нагрузку на операционную систему.

Многопотоковая односерверная архитектура

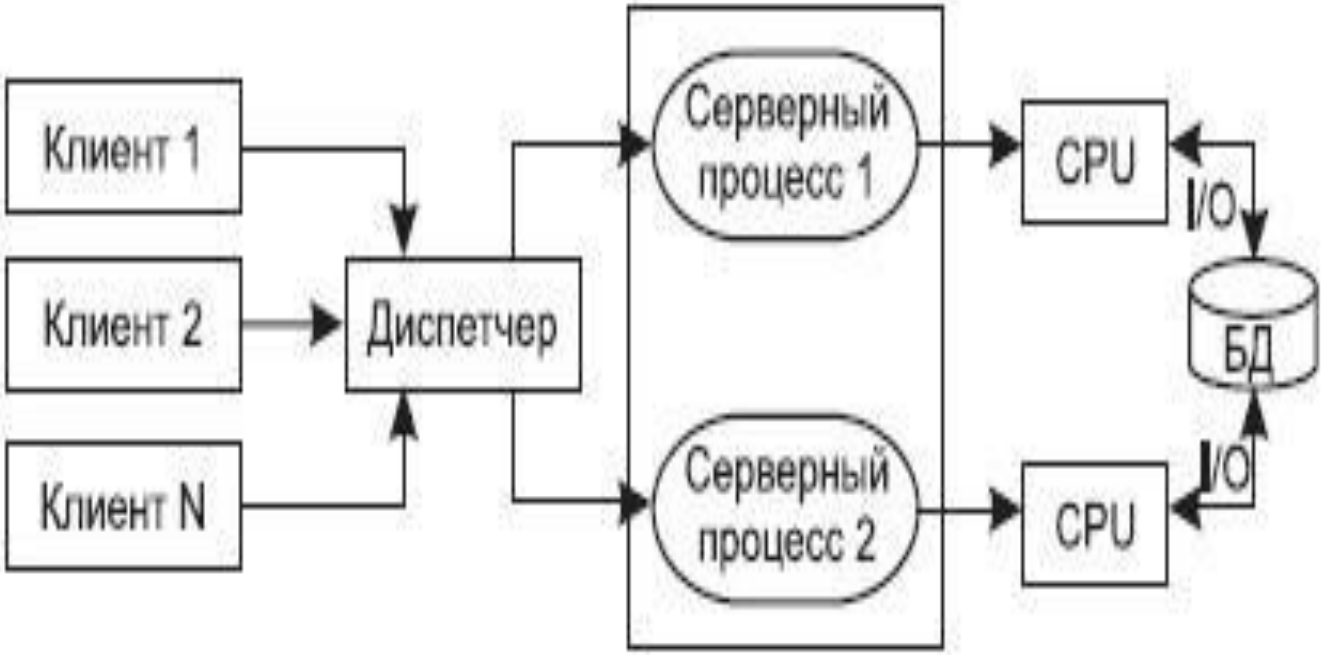


- Кроме того, возможность взаимодействия с одним сервером многих клиентов позволяет в полной мере использовать разделяемые объекты, что уменьшает потребности в памяти.
- Однако такое решение имеет свои недостатки.
- Так как сервер может выполняться только на одном процессоре, возникает естественное ограничение на применение СУБД для мультипроцессорных платформ. Если компьютер имеет, например, четыре процессора, то СУБД с одним сервером используют только один из них, не загружая оставшиеся три.

- Эта проблема решается вводом промежуточного слоя - диспетчера.
- Такая архитектура называется архитектурой виртуального сервера («virtual server»).
- В этой архитектуре клиенты подключаются не к реальному серверу, а к промежуточному звену, называемому диспетчером, который выполняет только функции диспетчеризации запросов к актуальным серверам.
- В этом случае нет ограничений на использование многопроцессорных платформ. Количество актуальных серверов может быть согласовано с количеством процессоров в системе.

- Но эта архитектура имеет недостатки, т.к. в систему добавляется новый слой, размещаемый между клиентом и сервером, что увеличивает трату ресурсов на поддержку баланса загрузки актуальных серверов («load balancing») и ограничивает возможности управления взаимодействием «клиент-сервер».
- Во-первых, становится невозможным направить запрос от конкретного клиента конкретному серверу;
- во-вторых, серверы становятся равноправными - нет возможности устанавливать приоритеты для обслуживания запросов.

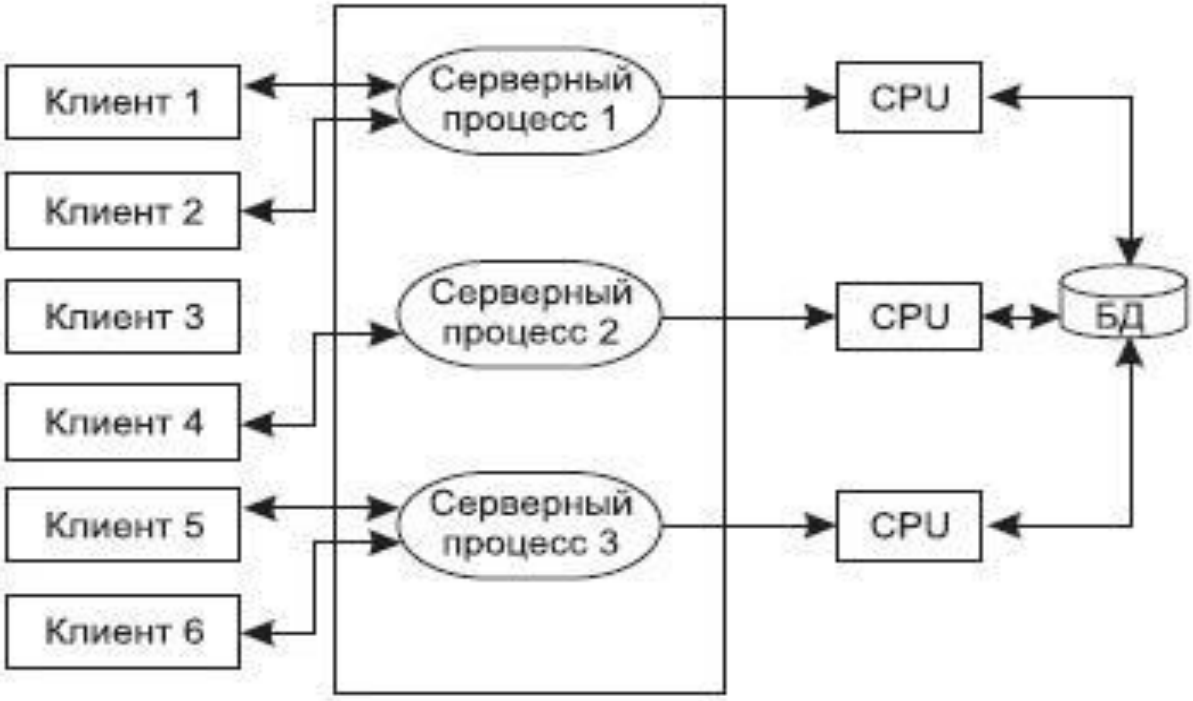
Архитектура с виртуальным сервером



- Современное решение проблемы СУБД для мультипроцессорных платформ - возможность запуска нескольких серверов базы данных, в том числе и на различных процессорах.
- При этом каждый из серверов должен быть многопотоковым.
- Если эти два условия выполнены, то это многопотоковая архитектура с несколькими серверами.

  Такая архитектура также может быть названа многонитевой мультисерверной архитектурой.
- Эта архитектура связана с вопросами распараллеливания выполнения одного пользовательского запроса несколькими серверными процессами.

Многопотоковая мультисерверная архитектура



Распараллеливание запроса

- Существует несколько возможностей распараллеливания выполнения запроса.

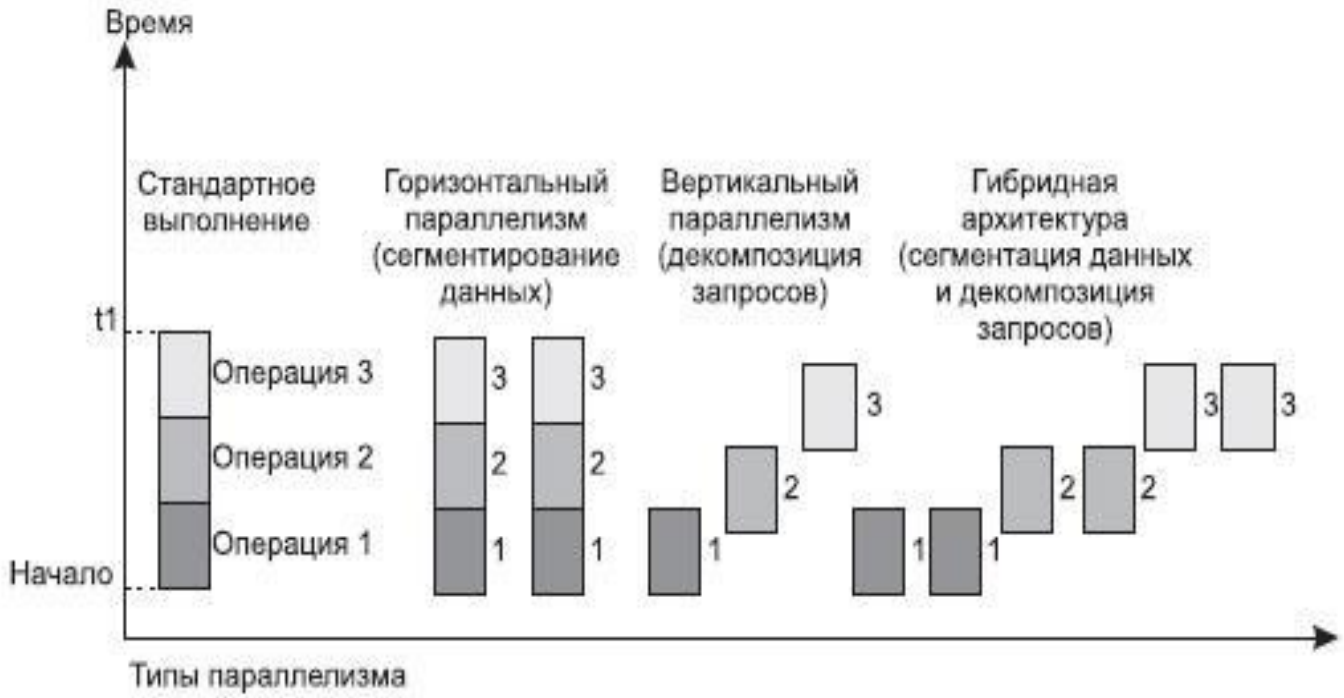
  В этом случае пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса.
- Тогда для обеспечения оперативности выполнения запросов их подзапросы могут быть направлены отдельным серверным процессам, а потом полученные результаты объединены в общий результат.

Распараллеливание запроса

- В данном случае серверные процессы не являются независимыми процессами, такими, как рассматривались ранее.

  Эти серверные процессы принято называть нитями (treads), и управление нитями множества запросов пользователей требует дополнительных расходов от СУБД, однако при оперативной обработке информации в хранилищах данных такой подход наиболее перспективен.

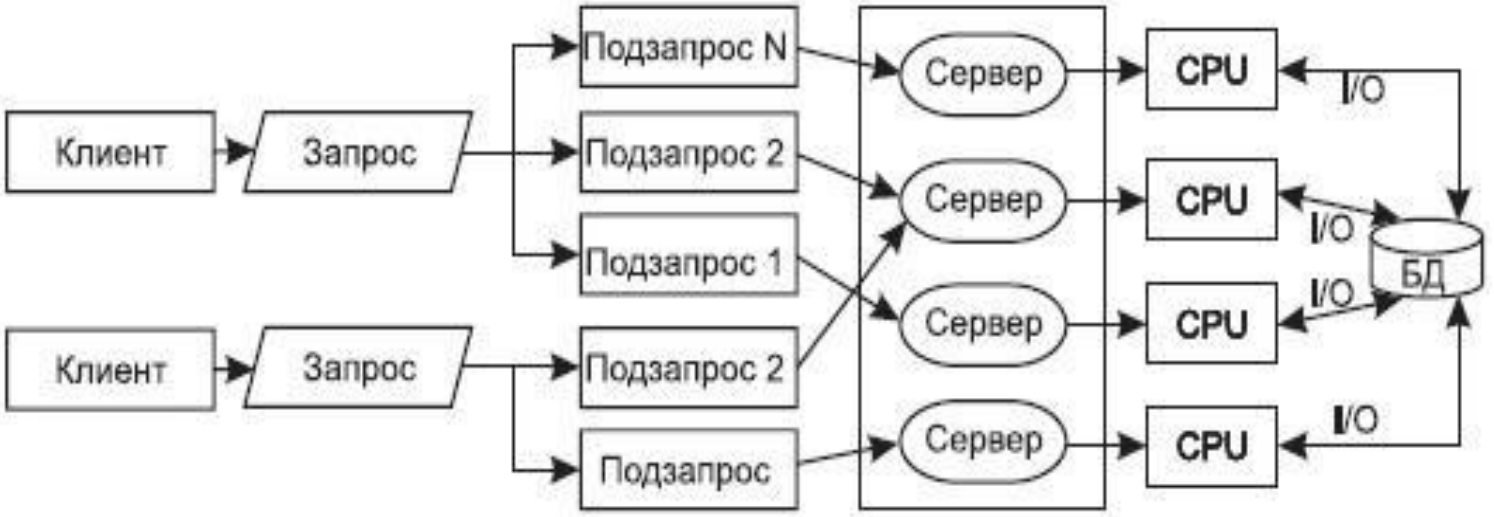
Многонитевая мультисерверная архитектура



Горизонтальный параллелизм

- Существует несколько путей распараллеливания запросов. Параллелизм возникает тогда, когда хранимая в БД информация распределяется по нескольким физическим устройствам хранения — нескольким дискам. При этом информация из одного отношения разбивается на части по горизонтали. Этот вид параллелизма называют распараллеливанием или сегментацией данных. И параллельность здесь достигается путем выполнения одинаковых операций, например фильтрации, над разными физическими хранимыми данными. Эти операции могут выполняться параллельно разными процессами, они независимы. Результат выполнения целого запроса складывается из результатов выполнения отдельных операций.

Выполнение запроса при горизонтальном параллелизме



Время выполнения такого запроса при соответствующем сегментировании данных существенно меньше, чем время выполнения этого же запроса традиционными способами одним процессом.

Вертикальный параллелизм

- Этот параллелизм достигается конвейерным выполнением операций, составляющих запрос пользователя. Этот подход требует серьезного усложнения в модели выполнения реляционных операций ядром СУБД.
- Он предполагает, что ядро СУБД может произвести декомпозицию запроса, базируясь на его функциональных компонентах, и при этом ряд подзапросов может выполняться параллельно, с минимальной связью между отдельными шагами выполнения запроса.

Вертикальный параллелизм

- Действительно, если рассмотреть, например, последовательность операций реляционной алгебры:

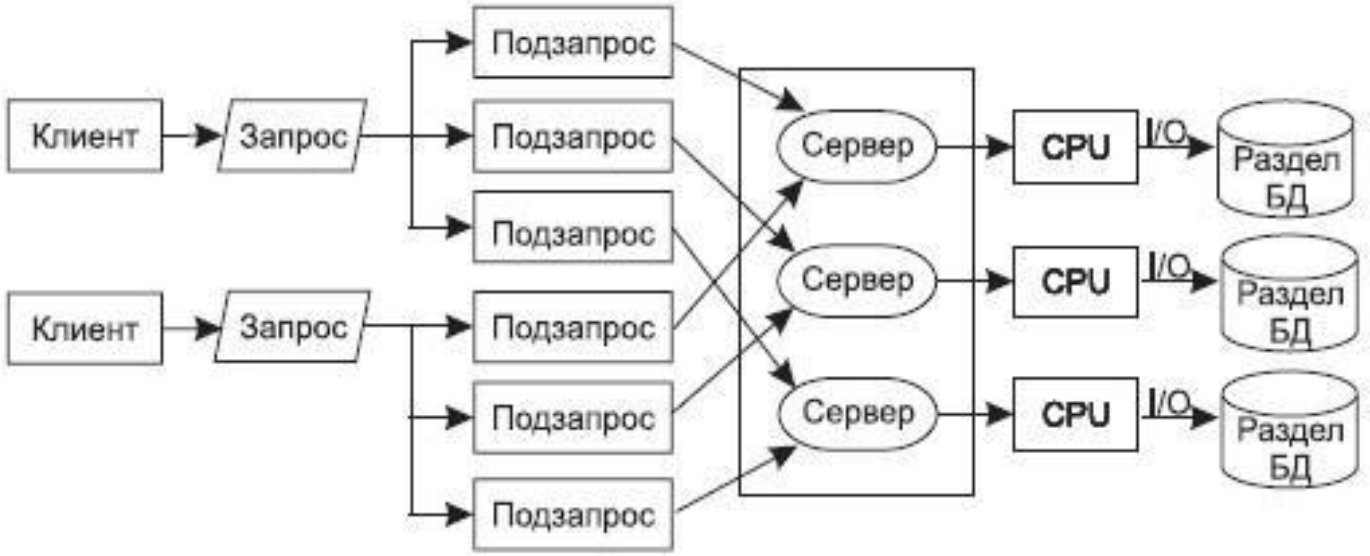
  R5=R1 [A,C]

  R6=R2 [A,B,D]

  R7 = R5[A > 128]

  R8 = R5[A]R6, то операции первую и третью можно объединить и выполнить параллельно с операцией два, а затем выполнить над результатами последнюю четвертую операцию.
- Общее время выполнения подобного запроса будет существенно меньше, чем при традиционном способе выполнения последовательности из четырех операций.

Гибридный параллелизм



Этот вид параллелизма является гибридом горизонтального и вертикального

- Наиболее активно применяются все виды параллелизма в OLAP-приложениях, где эти методы позволяют существенно сократить время выполнения сложных запросов над очень большими объемами данных.

- Существует несколько возможностей распараллеливания выполнения запроса, когда пользовательский запрос разбивается на ряд подзапросов, которые могут выполняться параллельно, а результаты их выполнения потом объединяются в общий результат выполнения запроса.
- Тогда для обеспечения оперативности выполнения запросов их подзапросы могут быть направлены отдельным серверным процессам, а потом полученные результаты объединены в общий результат.
- В данном случае серверные процессы не являются независимыми процессами, такими, как рассматривались ранее.
- Эти серверные процессы принято называть нитями (treads).