*Основные особенности архитектуры клиент-сервер

Вычислительная модель «клиент—сервер» исходно связана с парадигмой открытых систем, которая появилась в 90-х годах и быстро эволюционировала.

Сам термин «клиент-сервер» исходно применялся к архитектуре программного обеспечения, которое описывало распределение процесса выполнения по принципу взаимодействия двух программных процессов, один из которых в этой модели назывался «клиентом», а другой — «сервером».

Клиентский процесс запрашивал некоторые услуги, а серверный процесс обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов.

Ранее приложение (пользовательская программа) не разделялась на части, оно выполнялось некоторым монолитным блоком.

Но возникла идея более рационального использования ресурсов сети.

Действительно, при монолитном исполнении используются ресурсы только одного компьютера, а остальные компьютеры в сети рассматриваются как терминалы. Но теперь, 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 (Database Logic),
- функции управления информационными ресурсами (Database Manager System);
- служебные функции, играющие роль связок между функциями первых четырех групп.

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

Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения, к этой же части относится все то, что выводится пользователю на экран как результаты решения некоторых промежуточных задач либо как справочная информация. Поэтому основными задачами презентационной логики являются:

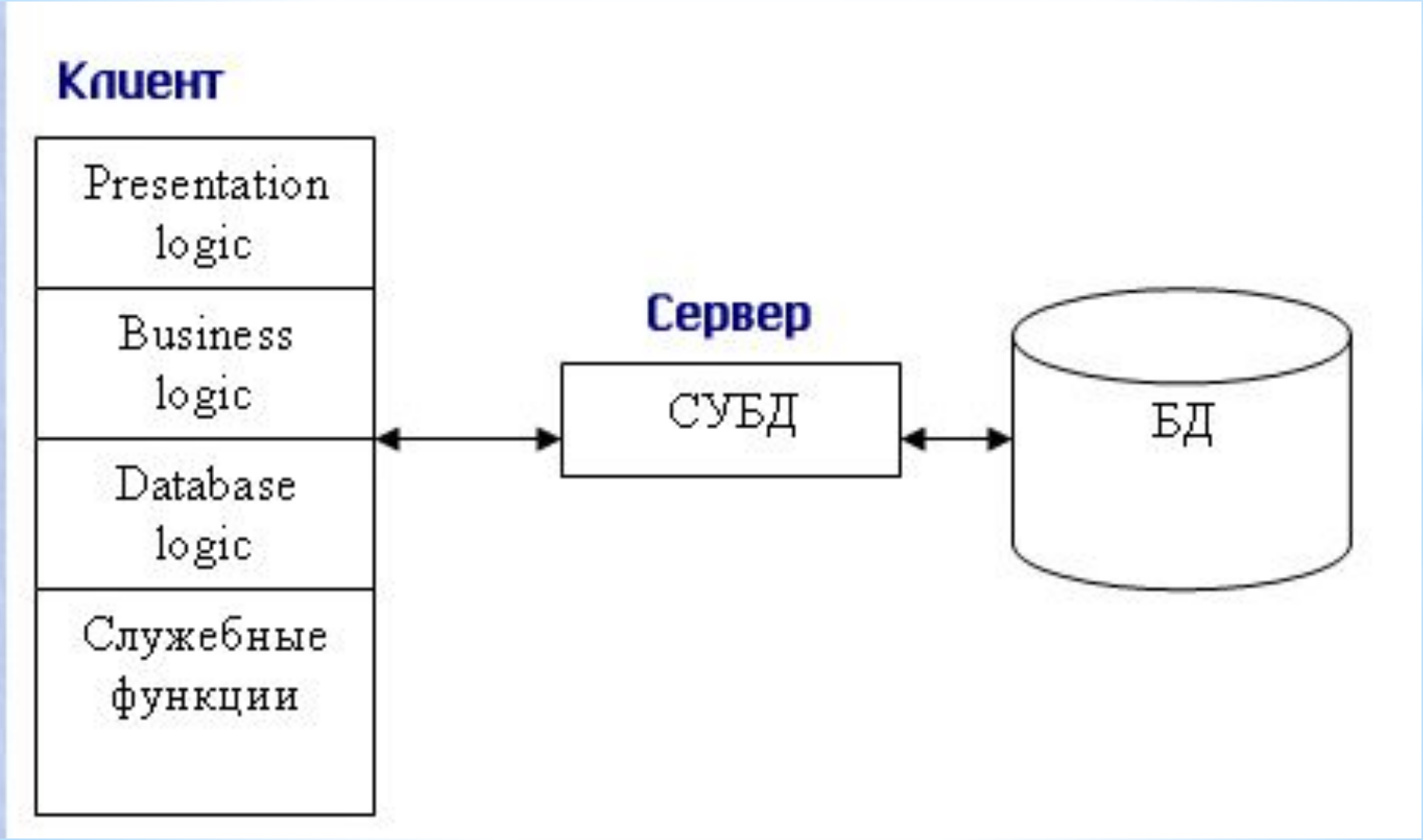
- формирование экранных изображений;
- чтение и запись в экранные формы информации;
- управление экраном;
- обработка движений мыши и нажатие клавиш клавиатуры.

Бизнес-логика, или логика собственно приложений (Business processing Logic), — это часть кода приложения, которая определяет собственно алгоритмы решения конкретных задач приложения.

Логика обработки данных (Data manipulation Logic) — это часть кода приложения, которая связана с обработкой данных внутри приложения. Данными управляет собственно СУБД. Для обеспечения доступа к данным используются язык запросов и средства манипулирования данными стандартного языка SQL.

Процессор управления данными (Database Manager System Processing) — это собственно СУБД, которая обеспечивает хранение и управление базами данных. В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако для рассмотрения архитектуры приложения нам надо их выделить в отдельную часть приложения.

Структура типового приложения, работающего с базой данных



В централизованной архитектуре эти части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы.

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

Модель удаленного управления данными. Модель файлового сервера

Модель удаленного управления данными также называется моделью файлового сервера (File Server, FS). В этой модели презентационная логика и бизнес-логика располагаются на клиенте. На сервере располагаются файлы с данными и поддерживается доступ к файлам. Функции управления информационными ресурсами в этой модели находятся на клиенте.

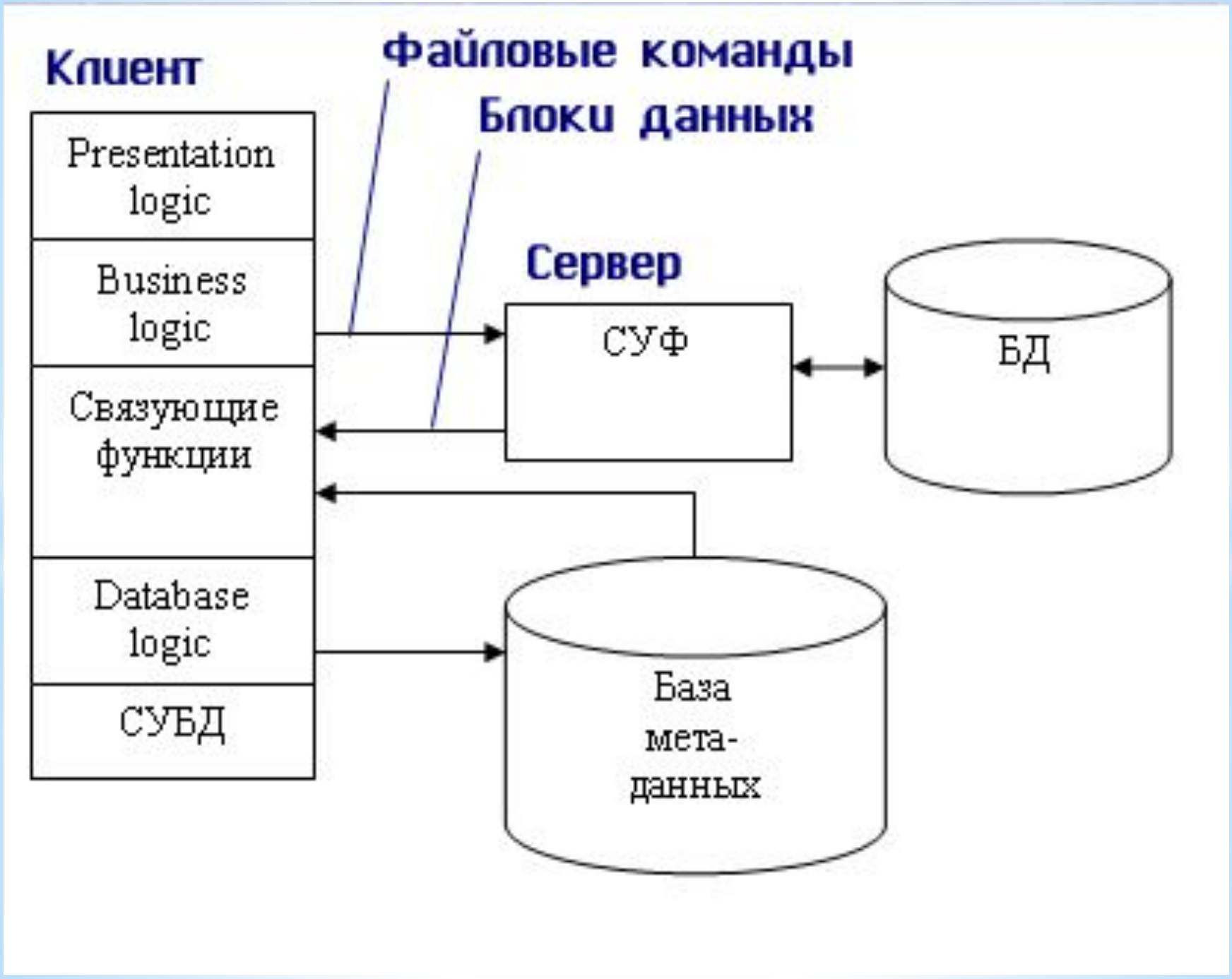
Достоинства этой модели в том, что мы *уже* имеем разделение монопольного приложения на два взаимодействующих процесса. При этом сервер (серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на клиенте.

Запрос клиента формулируется в командах ЯМД. СУБД переводит этот запрос в последовательность файловых команд. Каждая файловая команда вызывает перекачку блока информации на клиента, далее на клиенте СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации и т. д.

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

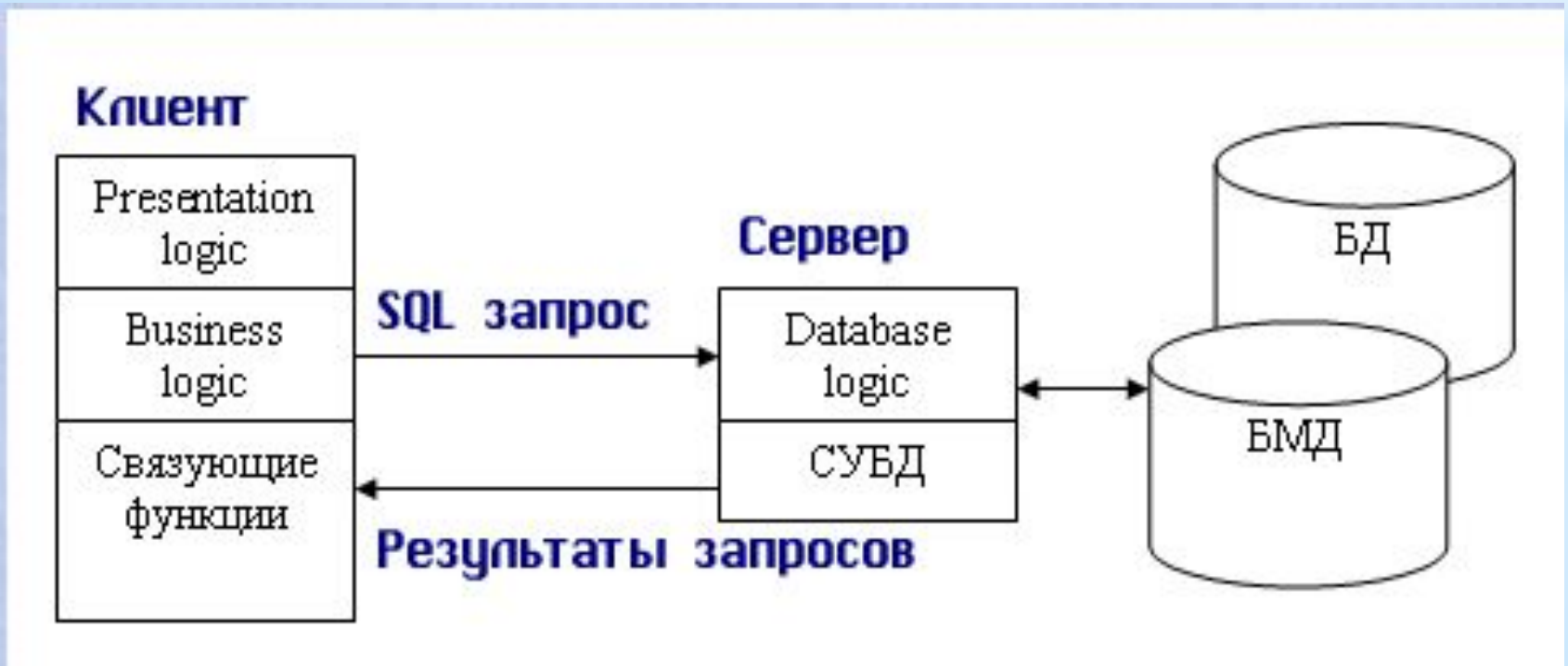
Недостатки:

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, который определяется только файловыми командами;
- отсутствие адекватных средств безопасности доступа к


Клиент
Файловые команды
Блоки данных
Presentation logic
Business logic
Связующие функции
Database logic
СУБД
Сервер
СУФ
БД
База мета-данных

Модель удаленного доступа к данным

В модели удаленного доступа (Remote Data Access, RDA) база данных хранится на сервере. На сервере же находится ядро СУБД. На клиенте располагается презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL.



Преимущества данной модели

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгрузил сервер БД, сводя к минимуму общее число процессов в операционной системе;
- сервер БД освобождается от несвойственных ему функций; процессор сервера целиком загружаются операциями обработки данных, запросов и транзакций;
- резко 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.
- Основное достоинство RDA-модели — унификация интерфейса «клиент-сервер», стандартом при общении приложения-клиента и сервера становится язык SQL.

Недостатки:

- все-таки запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;
- так как в этой модели на клиенте располагается и презентационная логика, и бизнес-логика приложения, то при повторении аналогичных функций в разных приложениях код соответствующей бизнес-логики должен быть повторен для каждого клиентского приложения. Это вызывает излишнее дублирование кода приложений;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте.

Модель сервера баз данных

Для того чтобы избавиться от недостатков модели удаленного доступа, должны быть соблюдены следующие условия:

- Необходимо, чтобы БД в каждый момент отражала текущее состояние предметной области, которое определяется не только собственно данными, но и связями между объектами данных. То есть данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.
- БД должна отражать некоторые правила предметной области, законы, по которым она функционирует (business rules).
- Необходимо, чтобы возникновение некоторой ситуации в БД четко и оперативно влияло на ход выполнения прикладной задачи.

- 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.
- Одной из важнейших проблем СУБД является контроль типов данных. В настоящий момент СУБД контролирует синтаксически только стандартно-допустимые типы данных, то есть такие, которые определены в DDL (data definition language) — языке описания данных, который является частью SQL. Однако в реальных предметных областях у нас действуют данные, которые несут в себе еще и семантическую составляющую, например, рабочая неделя в отличие от реальной имеет сразу после пятницы понедельник.

Данную модель поддерживают большинство современных СУБД. Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры.

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, которые требуются 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 также являются частью БД.

В данной модели сервер является активным, потому что 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. существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

Недостатком данной модели является очень большая загрузка сервера.

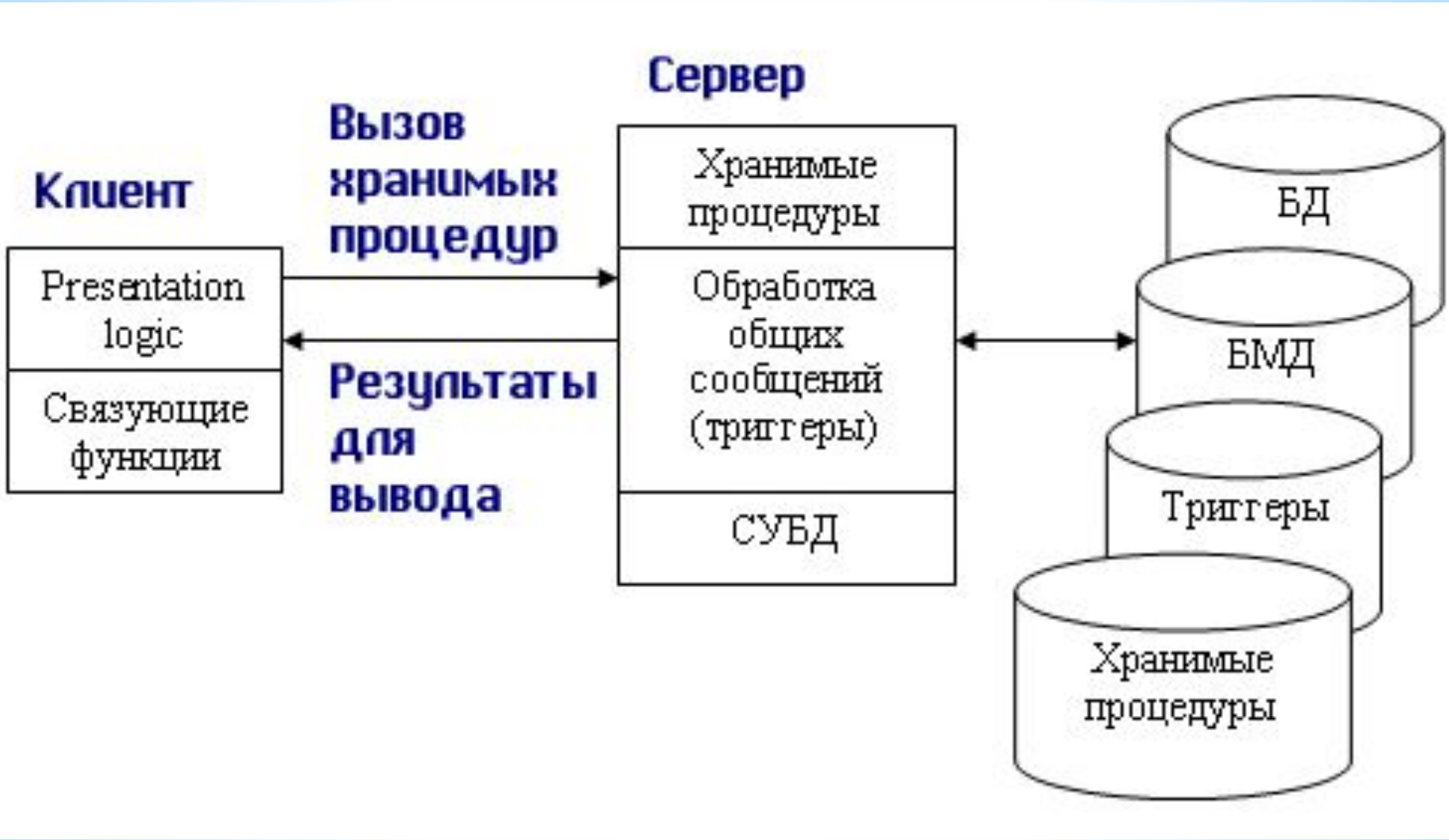
Действительно, сервер обслуживает множество клиентов и выполняет следующие функции:

- 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 и 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

Если мы переложили на сервер большую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшаются.

Иногда такую 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.


Клиент
Presentation logic
Связующие функции
Вызов хранимых процедур
Результаты для вывода
Сервер
Хранимые процедуры
Обработка общих сообщений (триггеры)
СУБД
БД
БМД
Триггеры
Хранимые процедуры

Модель сервера приложений

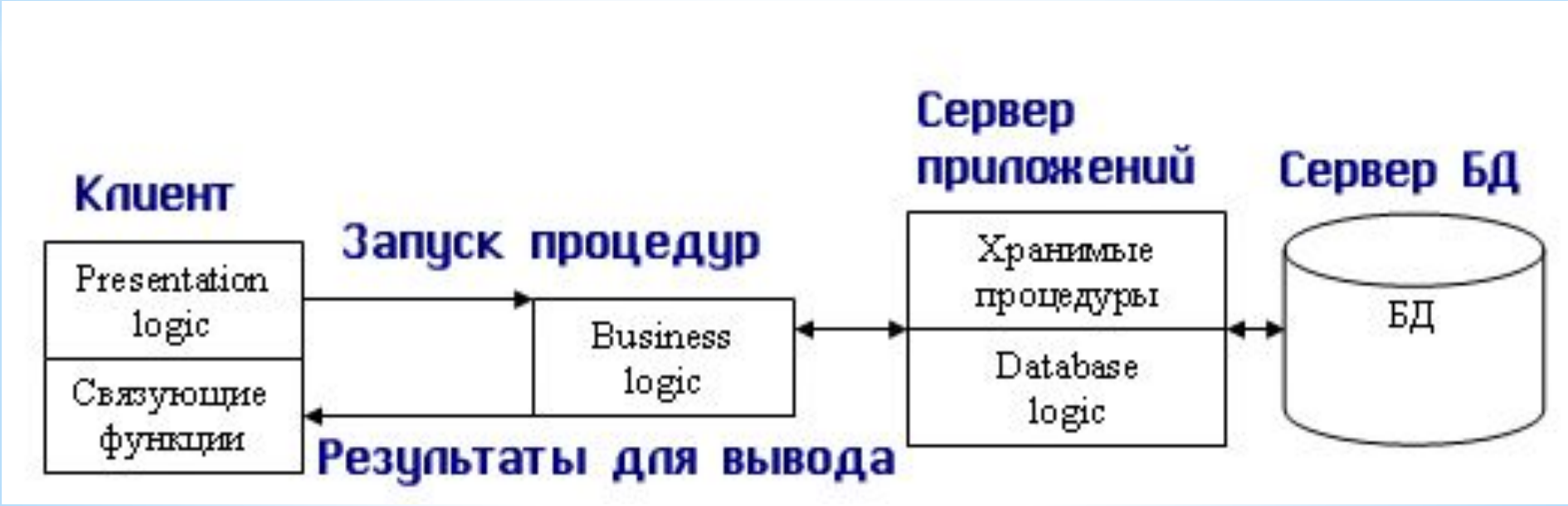
Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Этот промежуточный уровень содержит один или несколько серверов приложений.

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 части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть.

- *Серверы приложений* составляют новый промежуточный уровень архитектуры. Они спроектированы как исполнения общих незагружаемых функций для клиентов. 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, особенно в распределенных транзакциях.
- *Серверы баз данных* в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных. Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших приложений.

Отметим, что эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных. 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. Это повышает переносимость системы, ее масштабируемость.



Клиент — 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 простейшая бизнес-логика: интерфейс авторизации, проверка вводимых значений на допустимость и соответствие формату, несложные операции (сортировка, группировка, подсчет значений) с данными, уже загруженными на терминал.

Сервер приложений располагается на втором уровне. На втором уровне сосредоточена большая часть бизнес-логики. Вне его остаются фрагменты, экспортируемые на терминалы, 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.

В простейшей конфигурации физически сервер приложений может быть совмещён с сервером базы данных на одном компьютере, к которому по сети подключается один или несколько терминалов.

В «правильной» конфигурации сервер базы данных находится на выделенном компьютере, к которому по сети подключены один или несколько серверов приложений, к которым, в свою очередь, по сети подключаются терминалы.

*Достоинства

По сравнению с клиент-серверной или файл-серверной архитектурой можно выделить следующие достоинства трёхуровневой архитектуры:

- Масштабируемость - способность системы справляться с увеличением рабочей нагрузки (увеличивать свою производительность) при добавлении ресурсов.
- конфигурируемость — изолированность уровней друг от друга позволяет (при правильном развертывании архитектуры) быстро и простыми средствами переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней
- высокая безопасность
- высокая надёжность

- низкие требования к скорости канала (сети) между терминалами и сервером приложений
- низкие требования к производительности и техническим характеристикам терминалов, как следствие снижение их стоимости.

*Недостатки

Недостатки вытекают из достоинств. По сравнению с клиент-серверной или файл-серверной архитектурой можно выделить следующие недостатки трёхуровневой архитектуры:

- более высокая сложность создания приложений;
- сложнее в разворачивании и администрировании;
- высокие требования к производительности серверов приложений и сервера базы данных, а, значит, и высокая стоимость серверного оборудования;
- высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.