Архитектура клиент-сервер

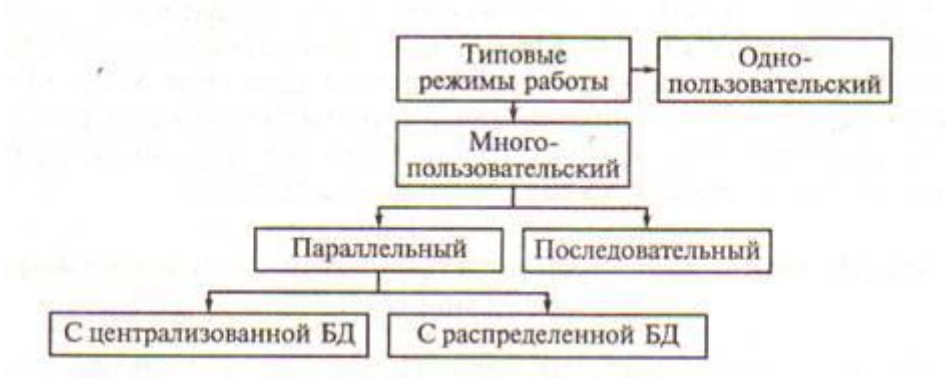
Вспомним, что современные БД должны удовлетворять определенному набору ***требований***:

- адекватность базы данных предметной области;
- интегрированность данных;
- независимость данных;
- минимальная избыточность хранимых данных;
- целостность базы данных;
- обеспечение защиты от несанкционированного доступа или случайного уничтожения данных;
- гибкость и адаптивность структуры базы данных;
- динамичность данных и способность к расширению;
- возможность поиска по многим ключам.

А современная СУБД выполняет следующие функции:

- ввод информации в БД и обеспечение ее логического контроля;
- возможность исправления информации;
- удаление устаревшей информации;
- контроль непротиворечивости данных;
- защита данных от разрушения;
- поиск информации с заданными свойствами;
- автоматическое упорядочение информации в соответствии с определенными требованиями;
- обеспечение коллективного доступа к данным нескольких пользователей одновременно;
- защита данных от несанкционированного доступа.

Системы управления базами данных, обеспечивающие возможность одновременного доступа к информации различным пользователям называют системами управления распределенными базами данных. Повторим основные понятия, применяемые в системах управления распределенными базами данных.



Пользователь БД — программа или человек, обращающийся к базе данных.

Запрос — процесс обращения пользователя к БД с целью ввода, получения или изменения информации в БД.

Транзакция — последовательность операций модификации данных в БД, переводящая БД из одного непротиворечивого состояния в другое непротиворечивое состояние.

Логическая структура БД — определение БД на физически независимом уровне; ближе всего соответствует концептуальной модели БД.

Топология БД, или *структура распределенной БД,* — схема распределения физической организации базы данных в сети.

Локальная автономность означает, что информация локальной БД и связанные с ней определения данных принадлежат локальному владельцу и им управляются.

Удаленный запрос — запрос, который выполняется с использованием модемной связи.

Возможность реализации удаленной транзакции — обработка одной транзакции, состоящей из множества SQL-запросов, на одном удаленном узле.

Поддержка распределенной транзакции допускает обработку транзакции, состоящей из нескольких запросов SQL, которые выполняются на нескольких узлах сети (удаленных или локальных), но каждый запрос в этом случае обрабатывается только на одном узле.

Распределенный запрос — запрос, при обработке которого используются данные из БД, расположенные в разных узлах сети.

Системы распределенной обработки данных в основном связаны с первым поколением БД, которые строились на мультипрограммных операционных системах и использовали централизованное хранение БД на устройствах внешней памяти центральной ЭВМ и терминальный многопользовательский режим доступа. При этом пользовательские терминалы не имели собственных ресурсов, т. е. процессоров и памяти, которые могли бы использоваться для хранения и обработки данных. Первой полностью реляционной системой, работающей в многопользовательском режиме, была СУБД SYSTEM R фирмы IBM. Именно в ней были реализованы как язык манипулирования данными SQL, так и основные принципы синхронизации, применяемые при распределенной обработке данных, которые до сих пор являются базисными практически во всех коммерческих СУБД.

Для решения различных задач используются различные модели баз данных. Процесс выбора базы данных, которая наиболее подходит для построения конкретного приложения, называется *масштабированием*.

Рассмотрим некоторые модели баз данных.

Автономные базы данных.

Автономные локальные базы данных хранят свои данные в локальной файловой системе на том компьютере, на котором они установлены, при этом, система управления и машина базы данных, осуществляющая доступ к ней, находится на том же самом компьютере. Сеть в данном случае не используется.

Разработчик автономной базы данных не имеет дело с проблемой параллельного доступа, когда два человека пытаются одновременно изменить одну и ту же запись.

Автономные базы данных не используются для приложений, требующих значительной вычислительной мощности.

К приложениям, использующим автономные базы данных, можно отнести приложения, обрабатывающие документацию небольшого подразделения или конторы, кадровый состав небольшого предприятия, бухгалтерские документы небольшой бухгалтерии и т. п. Каждый пользователь такого приложения манипулирует своими собственными данными на своем компьютере и не имеет доступа к данным другого пользователя.

Файл-серверные базы данных.

Файл-серверные базы данных отличаются от автономных тем, что они могут быть доступны многим клиентам через сеть. Сама база данных хранится на сетевом файл-сервере в единственном экземпляре. Для каждого клиента во время работы создается локальная копия данных, с которой он манипулирует. При этом возникают и решаются проблемы, связанные с возможным одновременным доступом нескольких пользователей к одной и той же информации.

Одним из недостатков файл-серверных баз данных является непроизводительная загрузка сети. При каждом запросе клиента данные в его локальной копии полностью обновляются из базы данных на сервере. Даже если запрос относится всего к одной записи, обновляются все записи данных. Если записей в базе данных много, то даже при небольшом числе клиентов сеть будет загружена очень основательно, что серьезно скажется на скорости выполнения запросов.

К приложениям, использующим файл-серверные базы данных, можно отнести приложения, обслуживающие крупные учреждения. В таких приложениях администраторы отдельных подразделений обращаются к общим данным и не создают свои локальные базы данных (при этом сохраняется конфиденциальность информации, и каждый администратор имеет доступ только к той информации, которая касается деятельности только его подразделения).

Базы данных типа клиент-сервер.

Для больших баз данных с множеством пользователей часто используются базы данных на платформе клиент/сервер. В этом случае

доступ к базе данных для группы клиентов выполняется специальным компьютером – сервером. Клиент дает задание серверу выполнить те или иные операции поиска или обновления базы данных. Мощный сервер, ориентированный на операции с запросами клиентов, самым оптимальным способом выполняет такие запросы и сообщает клиенту результаты своей работы.

Подобная организация работы повышает эффективность выполнения приложений за счет использования мощности сервера, разгружает сеть и обеспечивает хороший контроль целостности данных.

В базах данных такого типа возникает проблема проектирования приложения таким образом, чтобы оно максимально использовало возможности сервера и минимально нагружало сеть, передавая через нее только минимум информации.

Модели "клиент—сервер" в технологии баз данных

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.

И как в промышленности, здесь возникает древняя как мир идея распределения обязанностей, разделения труда. Конвейеры Форда сделали в свое время прорыв в автомобильной промышленности, показав наивысшую производительность труда именно из-за того, что весь процесс сборки был разбит на мелкие и максимально простые операции и каждый рабочий специализировался на выполнении только одной операции, но эту операцию он выполнял максимально быстро и качественно.

Конечно, в вычислительной технике нельзя было напрямую использовать технологию автомобильного или любого другого механического производства, но идею использовать было можно. Однако для воплощения идеи необходимо было разработать модель разбиения единого

монолитного приложения на отдельные части и определить принципы взаимосвязи между этими частями.

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 приложения, работающего с базой данных приведена на рисунке

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.

Некоторые возможности для организации презентационной логики приложений предоставляет знако-ориентированный пользовательский интерфейс, задаваемый моделями CICS (Customer Control Information System) и IMS/DC фирмы IBM и моделью TSO (Time Sharing Option) для централизованной main-фреймовой архитектуры. Модель GUI — графического пользовательского интерфейса, поддерживается в

операционных средах Microsoft's Windows, Windows NT, в OS/2 Presentation Manager, X-Windows и OSF/Motif.

Бизнес-логика, или логика собственно приложений (Business processing Logic), — это часть кода приложения, которая определяет собственно алгоритмы решения конкретных задач приложения. Обычно этот код пишется с использованием различных языков программирования, таких как C, C++, Cobol, SmallTalk, Visual-Basic.

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

Обычно операторы языка SQL встраиваются в языки 3-го или 4-го поколения (3GL, 4GL), которые используются для написания кода приложения.

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.

В централизованной архитектуре (Host-based processing) 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. В зависимости от характера распределения можно выделить следующие 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 компонентов приложения в моделях клиент — сервер

Модели распределений	Компоненты приложения						Пользователь
	Функции логики представления		Функции бизнес-логики		Функции управления данными		
Распределенное представление (DP)		DP	RP	DBL	RDM	DDM	
	■						Клиент
		■	■	■	■	■	Сервер
Удаленное представление (RP)							
	■	■					Клиент
			■	■	■	■	Сервер
Распределенная бизнес-логика (DBL)							
	■	■	■				Клиент
				■	■	■	
Удаленное управление данными (RDM)							
	■	■	■	■			Клиент
					■	■	Сервер
Распределенное управление данными (DDM)							
	■	■	■	■	■		Клиент
						■	Сервер
Совмещение DBL и DDM							
	■	■	■		■		Клиент
				■		■	Сервер

Примечание. Фоном выделено наличие соответствующих компонентов приложения.

Двухуровневые модели

Двухуровневая модель фактически является результатом распределения пяти указанных выше функций между двумя процессами, которые выполняются на двух платформах: на клиенте и на сервере. В чистом виде почти никакая модель не существует, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели: модели удаленного управления данными и модели удаленного доступа к данным.

Модель удаленного управления данными.

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

Рис. 10.3. М

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

Достоинство этой модели заключается в том, что приложение разделено 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 клиентском компьютере.

Алгоритм выполнения клиентского запроса сводится к следующему.

1. Запрос формулируется в командах ЯМД.

2. СУБД переводит этот запрос в последовательность файловых команд.

3. Каждая файловая команда вызывает перекачку блока информации на компьютер клиента, а СУБД анализирует полученную информацию; если в полученном блоке не содержится ответ на запрос, то принимается решение о перекачке следующего блока информации, и т.д.

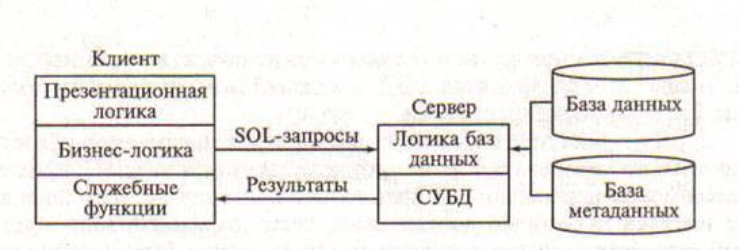
4. Перекачка информации с сервера на клиентский компьютер производится до тех пор, пока не будет получен ответ на запрос клиента.

Данная модель имеет следующие 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 данным (защита только на уровне файловой системы).

Модель удаленного доступа к данным.

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



Преимущества данной модели заключаются в следующем:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгружает сервер БД, сводя к минимуму общее число выполняемых процессов в операционной системе;

- « сервер БД освобождается от несвойственных ему функций; процессор или процессоры 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, а их объем существенно меньше. В ответ на запросы клиент получает только данные, соответствующие запросу, а не блоки файлов.

Основное достоинство RDA-модели — унификация интерфейса клиент—сервер (стандартом при общении приложения-клиента и сервера становится язык SQL).

Данная модель имеет следующие недостатки:
- запросы на языке SQL при интенсивной работе клиентской части приложения 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 приложения;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте.

Распределенная база данных, параллельный сервер баз данных

Есть много причин, которые могут подтолкнуть проектировщика к принятию решения о построении распределенной базы данных. Приведем некоторые из них:

- Размещение часто используемых данных ближе к клиенту, что позволяет минимизировать сетевой трафик.
- Расположение часто меняющихся данных в одном месте, обеспечивающее минимизацию затрат по синхронизации копий данных.
- Увеличение надежности комплекса к единичным отказам серверов (горячее резервирование).
- Понижение стоимости системы за счет использования группы небольших серверов вместо одного мощного центрального сервера.

Если в информационной системе наличествуют все перечисленные факторы, то распределенная база данных может оказаться хорошим решением. Часто распределение базы данных изначально определяется требованиями бизнеса, например, когда объединяется информация нескольких крупных подразделений одной компании и требуется постоянный обмен данными между филиалами.

Решение о распределенной базе данных оправданно и для систем, где есть четко выраженные группа меняющихся данных и группа устойчивых данных, по которым выполняются отчеты. Тогда в самом простом варианте работают два сервера: один обслуживает часто меняющиеся данные - это, как правило, OLTP (On-Line Transaction Processing. - Прим. ред.), второй - отчеты, то есть DSS (Decision Support System. - Прим. ред.). Ряд СУБД не очень хорошо совмещает обработку OLTP- и DSS-потоков запросов, поскольку для этих двух типов потоков запросов оптимальные параметры конфигурации серверов будут различаться. Решение такой базы данных как распределенной может оказаться более выгодным.

Преимущества распределенной базы данных имеют свою цену - должны быть обеспечены:

- Непротиворечивость данных, независимо от того, какой клиент к какому серверу обратился.
- Производительность распределенной базы данных должна удовлетворять требованиям заказчика, а это может оказаться более сложной задачей, чем в случае централизованной базы данных.
- Надежность распределенной базы данных не должна уступать надежности централизованной базы данных.

Современные СУБД позволяют осуществлять запрос данных, находящихся на разных узлах распределенной сети в одном SQL-запросе. Прозрачность доступа СУБД также обеспечивают - в каких-то реализациях хуже, в каких-то лучше. Одна из самых распространенных проблем - более сложная обработка транзакций. Практически все промышленные реализации поддерживают только двухфазную фиксацию транзакций, а в этом режиме не все типы отказов разрешимы. Распределенный deadlock (взаимоблокировка) также может представлять значительную проблему, и большинство реализаций используют только таймауты для его детектирования.

Как бы то ни было, стратегия распределения данных должна определяться не политикой предприятия (что обычно пытается навязать руководство), а требованиями надежности и производительности системы. При построении распределенной базы данных следует уделить особое внимание проектированию топологии сети. Узлы распределенной базы данных должны быть соединены скоростными надежными линиями связи. Это же касается линий соединения узлов базы данных и серверов приложений. Наличие низкоскоростного и ненадежного канала связи между узлами распределенной базы данных резко повышает количество детектируемых отказов сети.

В распределенной базе данных могут быть использованы следующие типы вызовов:

- Удаленные DDL- и DML- операции, а также выборка данных.
- Синхронные удаленные вызовы процедур.
- Асинхронные удаленные вызовы процедур.
- Непротиворечивые снимки.

- Асинхронная симметричная репликация.
- Синхронная симметричная репликация.
- Вызов распределенного запроса (запрашивает данные на чтение и модификацию с нескольких узлов).

Распределенная база данных может обеспечить горизонтальную фрагментацию; например, в филиале чаще всего используют данные о клиентах, находящихся в городе N (CLIENT_PLACE = "N"). В этом случае на узле распределенной базы данных этого филиала может быть расположен фрагмент таблицы данных, выделенный согласно условию CLIENT_PLACE = "N".

Стратегия распределения данных для каждой СУБД определяется по-своему. Определение стратегии преследует две цели: сократить нагрузку на сеть и сервер и/или повысить уровень готовности данных.

Следует отметить, что проектировщики должны четко представлять себе особенности реализации распределенной базы данных используемой СУБД. Не владеющий этой информацией проектировщик рискует создать нерабочую или плохо работающую схему, причем проявится это даже не на моделях, а в реальной эксплуатации.

Если вы решили строить распределенную базу данных, позаботьтесь о наличии в штате квалифицированного администратора. Есть одно простое правило: проектировщик не может принять окончательного решения об определении стратегии распределения данных без администратора баз данных. Редко встречаются проектировщики, которые имеют практический опыт администрирования 2-3 серверов баз данных и которые реально работали на них с распределенными базами данных. Для каждой СУБД принципы, влияющие на детали распределения базы данных, индивидуальны.

Особый интерес представляет неоднородная распределенная база данных. Это то, что постоянно, но безуспешно пытаются изживать и проектировщики, и администраторы баз данных. Никто не любит иметь дело со шлюзами данных. Но если неоднородной распределенной базы избежать не удалось, уделите особое внимание выбору шлюзов данных, а также ПО, обеспечивающему синхронизацию информации базах данных под управлением разных СУБД. Если синхронизация данных на узлах под управлением одной СУБД может быть обеспечена самой СУБД (что реализуется большинством производителей), то синхронизация данных на узлах под управлением разных СУБД обеспечивается, как правило, специальным ПО. Тщательно оцените, что будет дешевле: использовать это ПО или импортировать данные и схему и сделать распределенную базу данных однородной.

Следует также отметить, что в подавляющем большинстве реализаций каждый из узлов распределенной базы данных может обслуживаться параллельным сервером баз данных, и это является важным критерием повышения производительности системы. Промышленные СУБД также используют параллельный доступ к хранилищам данных; RAID, например,

могут размещать данные, используя не файловую систему операционной системы, а свою файловую систему. Реализация всех этих возможностей существенно различается у различных СУБД. Это означает, что выбор параллельного сервера баз данных должен осуществляться только после серьезных консультаций с администраторами тех СУБД, которые рассматриваются как возможные претенденты на роль сервера баз данных в проекте.

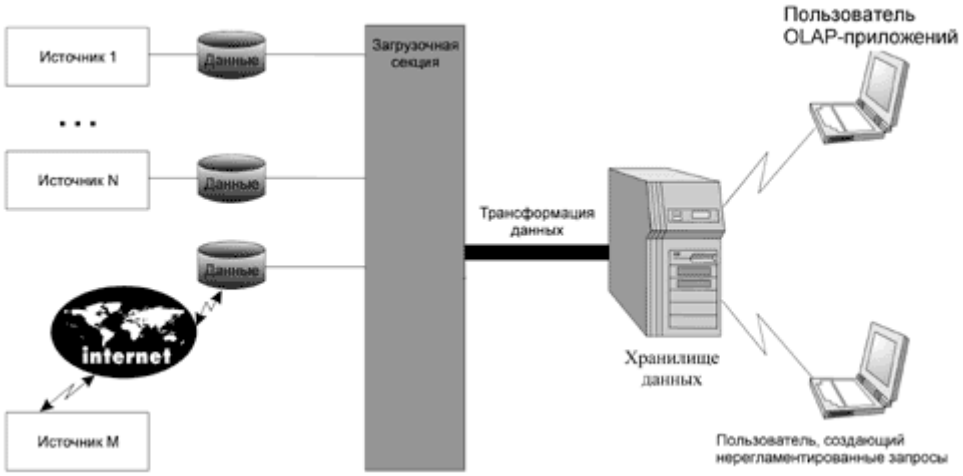
Метод распределения данных по дискам для поддержки параллельных вычислений во многом зависит и от особенностей реализации СУБД. Администратор базы данных не может выбрать такой метод в отрыве от особенностей конкретной информационной системы. Еще одна возможность параллельной обработки данных, предоставляемая СУБД: обработка одного запроса несколькими менеджерами ресурсов. В реализациях также имеется возможность использования одного хранилища данных несколькими серверами баз данных (Parallel Server). Такая архитектура может быть эффективно использована на кластерах.

Хранилище данных

Информационные системы такого вида сегодня одни из самых актуальных. Хранилище данных характеризуется большим объемом редко изменяемой информации. Хранилище данных обменивается информацией с другими хранилищами; как правило, речь идет о пакетной загрузке. Здесь решаются задачи обработки и хранения больших объемов информации (VLDB, Very Large Databases). Хранилище данных формирует сводное представление обо всех данных информационной системы.

В проектировании хранилищ данных важное место занимают интерфейсы обмена данными и интерфейсы конвертации данных. Следует четко выделить те системы, которые поставляют данные в хранилище, и те, что забирают данные из хранилища.

Внутренняя структура хранилища рассчитана на выполнение сложных запросов. Как правило, хранилище данных обслуживается выделенным сервером, а нередко - параллельным сервером баз данных. Хранилища данных используются для прогнозирования развития бизнеса. Они оснащаются средствами аналитической обработки данных (OLAP, On-Line Analytical Processing). На данный момент существуют специально ориентированные на OLAP версии серверов баз данных.



ПО хранилища баз данных состоит из следующих компонентов:

– ПО источников данных.

Здесь формируются входные пакеты данных, которые в состоянии интерпретировать конвертор и загрузчик данных. Это ПО входит в состав внешних систем (если повезло) или разрабатывается специально; в последнем случае основная его задача - достать данные из внешней системы.

– Загрузочная секция (отвечает и за трансформацию данных).

ПО принимает входные пакеты данных в необработанном формате, проверяет целостность пакета (например, контрольную сумму и т.п.), выполняет первичную обработку пакета (например, расшифровку), а также переводит данные во внутренний формат. Теперь они готовы для передачи в хранилище. Данные загрузочной секции хранятся во внутреннем формате системы, обеспечивающей пересылку данных.

– ПО трансформации данных.

Здесь обеспечиваются пересылка данных в хранилище и логика обработки данных. Данные проходит проверку на корректность, переводятся в нужный формат и интегрируются в хранилище.

– Собственно хранилище данных.

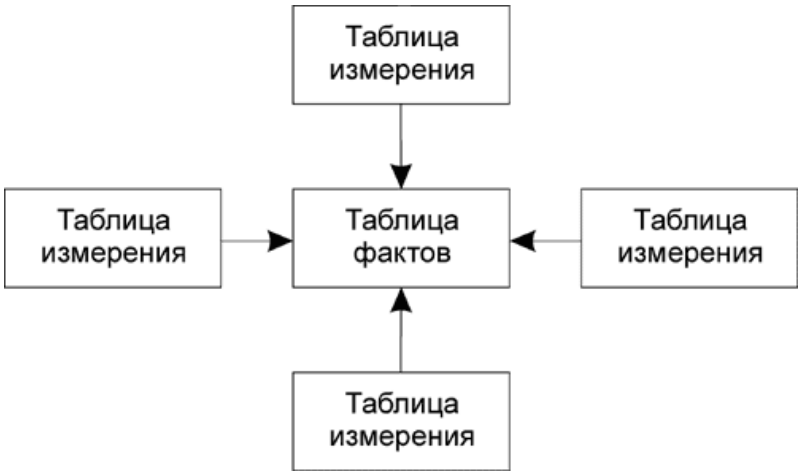
Преимущественно это один или несколько мощных параллельных серверов баз данных, которые обеспечивают обработку информации в хранилище. Само хранилище может быть распределенной среды (распределенная база данных) или трехуровневой среды (централизованная или распределенная база данных и сервер приложений). Собственно база данных хранилища редко представляет собой объединение рабочих баз данных. База данных не обязательно должна быть реляционной. Здесь могут с успехом использоваться расширения, предоставляемые современными СУБД, например Spatial Data-расширения, которые поддерживают многомерные данные.

– Интерфейс клиента.

ПО этого уровня обеспечивает взаимодействие приложений-клиентов, запрашивающих информацию из хранилища. Здесь запрещены операции модификации данных.

Рассмотрим один из приемов проектирования хранилищ данных.

Звездообразная схема (она же многомерная схема, соединение по схеме "звезда", куб данных) состоит из центральной таблицы фактов и нескольких таблиц измерений.



Факты представляют собой основные виды деятельности и факторы, влияющие на бизнес. В таблице измерений хранятся данные, способные оказывать определенное влияние либо порождать те или иные пути развития фактов.

Часто таблица фактов разбивается на секции для повышения скорости доступа к хранящимся в ней записям, в зависимости от запроса, поскольку оптимизатор в этом случае сканирует только часть таблицы/индекса (определенную секцию), а не весь объем данных. Отношение между таблицей фактов и таблицей измерения должно быть как можно более простым, так как должен существовать единственно возможный путь их соединения.

Список источников

- http://bourabai.ru/dbt/dbms/1002.htm
- https://studopedia.ru/20_24677_modeli-klientserver-v-tehnologii-raspredelennih-baz-dannih.html
- http://www.interface.ru/home.asp?artId=3696