Иерархические, сетевые и реляционные модели данных

База данных (БД) - это данные, организованные в виде набора записей определенной структуры и хранящиеся в файлах, где помимо самих данных, содержится описание их структуры.

Система управления базами данных (СУБД) - это система, обеспечивающая ввод данных в БД, их хранение и восстановление в случае сбоев, манипулирование данными, поиск и вывод данных по запросу пользователя.

По моделям представления данных базы данных делят на:

- иерархические;
- сетевые;
- реляционные;
- объектно-реляционные.

Иерархические базы данных - это самая первая модель представления данных, в которой все записи базы данных представлены в виде дерева, с отношениями предок-потомок (см. рис. 1.1). Для того, чтобы знать сколько деталей каждого вида надо заказать, строилось дерево (см. рис.1.1). Поскольку список составных частей изделия представлял из себя дерево, то для его хранения в базе данных наилучшим образом подходила иерархическая модель организации данных.

Рисунок 1.1 – Иерархическая база данных

Физически данные отношения реализуются в виде указателей на предков и потомков, содержащихся в самой записи. Такая модель представления данных связана с тем, что на ранних этапах базы данных часто использовались для планирования производственного процесса: каждое выпускаемое изделие состоит из узлов, каждый узел из деталей и т.п.

Однако иерархическая модель не является оптимальной. Допустим, что один и тот же тип болтов используется в автомобиле 300 раз в различных узлах. При использовании иерархической модели, данный тип болтов будет фигурировать в базе данных не 1 раз, а 300 раз (в каждом узле - отдельно). Налицо дублирование информации. Чтобы устранить этот недостаток была введена **сетевая модель** представления данных.

Сетевая база данных - это база данных, в которой одна запись может участвовать в нескольких отношениях предок-потомок (см. рис. 1.2).

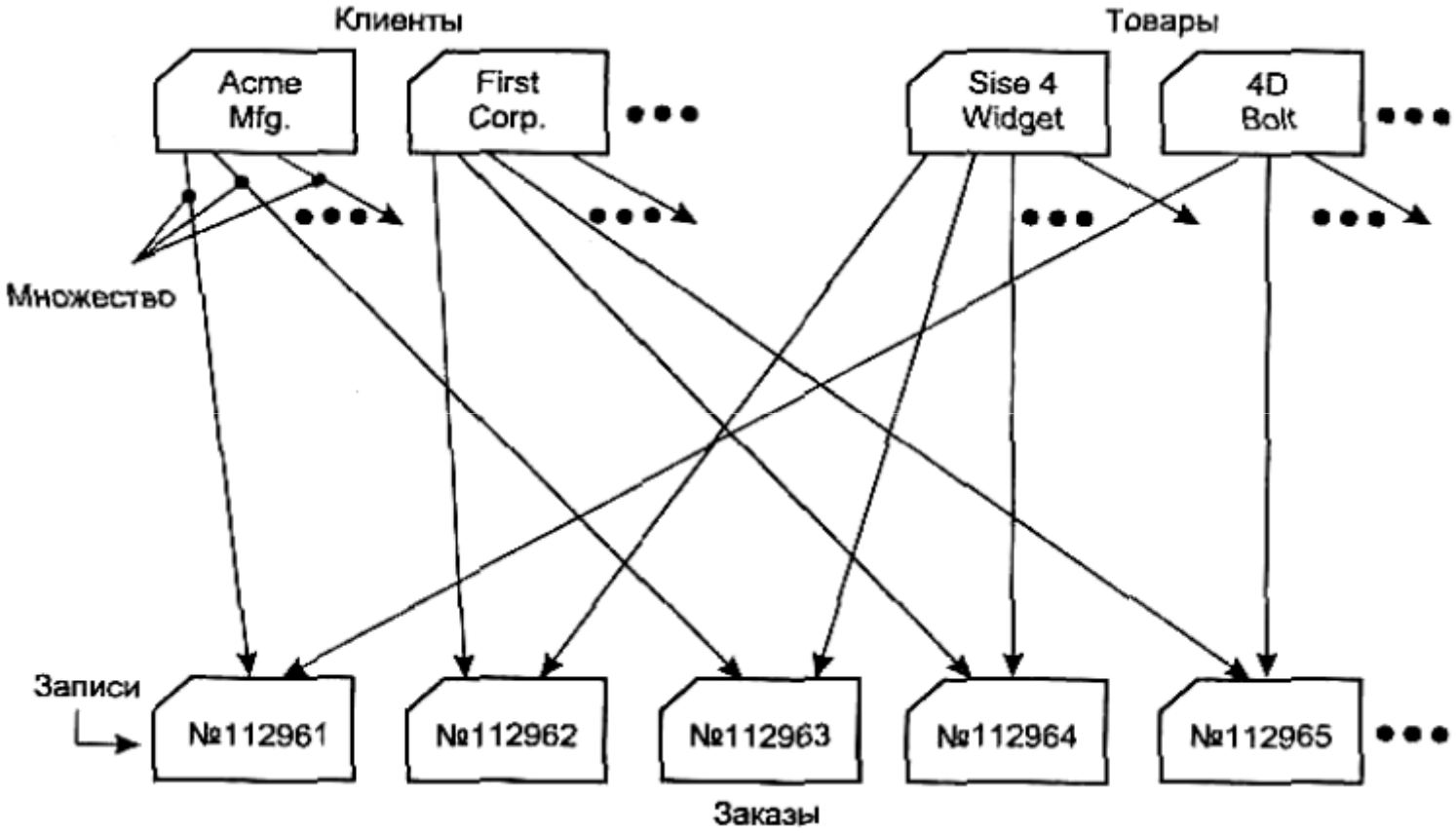

Рисунок 1.2 – Сетевая база данных

Физически данная модель также реализуется за счет хранящихся внутри самой записи указателей на другие записи, только, в отличие от иерархической модели, число этих указателей может быть произвольным.

И иерархическая, и сетевая модель достаточно просты, однако они имеют общий недостаток: для того, чтобы получить ответ даже на простой вопрос, программист был должен написать программу, которая просматривала базу данных, двигаясь по указателям от одной записи к другой.

Написание программы занимало некоторое время, и часто к тому моменту, когда такая программа была написана, необходимость в получении данных уже отпадала. Поэтому в середине 80-х годов 20 века произошел практически повсеместный переход к **реляционным базам данных.**

В ***реляционной базе данных*** вся информация представляется в виде таблиц и любые операции над данными - это операции над таблицами.

Таблицы состоят из строк и столбцов. Строки - это записи, а столбцы имеют определенный тип данных и длину данных). Строки в таблице не упорядочены - не существует первой или десятой строки. Однако поскольку на строки надо как-то ссылаться, то вводится понятие "первичный ключ".

Первичный ключ - это столбец, значения которого во всех строках разные. Используя первичный ключ можно однозначно сослаться на какую-либо строку таблицы. Первичный ключ может состоять и из нескольких столбцов (составной первичный ключ).

Некоторые СУБД требуют в явном виде указать первичный ключ таблицы, а некоторые позволяют пользователю не задавать для таблицы первичный ключ - в таком случае СУБД сама добавляет в таблицу столбец - первичный ключ, не отображаемый на экране (так например, в СУБД Oracle у любой таблицы существует псевдо-столбец ROWID, формируемый Oracle, который содержит уникальный адрес каждой строки). Отношения предок-потомок в реляционных БД реализуются при помощи внешних ключей.

Внешний ключ - это столбец таблицы, значения которого совпадают со значениями первичного ключа некоторой другой таблицы.

Так, например, на рис. 1.3 столбец "Ответственный" таблицы "Мероприятия" является внешним ключом для таблицы "Сотрудники" (первичный ключ - столбец "Фамилия").

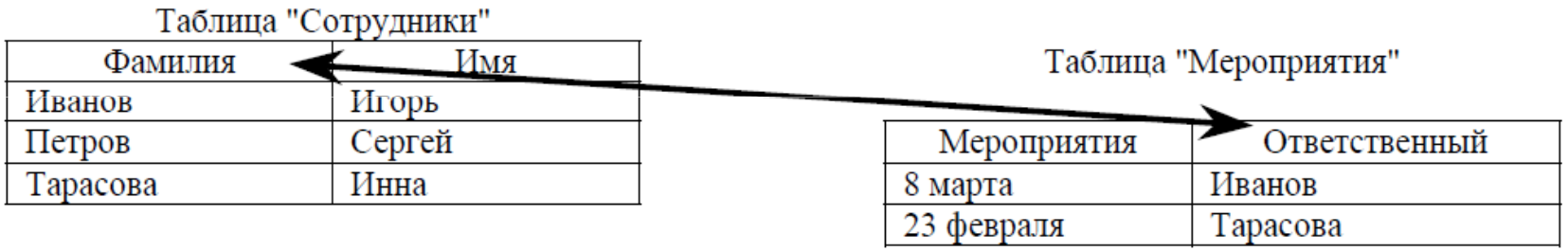

Рисунок 1.3 - Отношения предок-потомок в реляционных базах данных.

Важным моментом является также использование значения NULL в таблицах реляционной базы данных. NULL - это отсутствующее значение, отсутствие информации по данной позиции. Не допускается использовать 0 или "пробел" вместо NULL: понятно, что "нулевой" объем продаж - это не тоже самое, что "неизвестный" объем продаж. По этой же причине ни одно значение NULL не равно другому значению NULL. В реляционной базе данных, при запросах, группировке, сравнениях и т.д. - значения NULL обрабатываются особым образом.

Одъектно-реляционные базы данных появились в последнее время у значительного числа производителей СУБД (Oracle, Informix, PostgreSQL) и сочетают в себе реляционную модель данных с концепциями объектно-ориентированного программирования (полиморфизм, инкапсуляция, наследование).

Существующие архитектуры СУБД

По способу организации взаимодействия с базой данных через сеть СУБД делят на:

- СУБД с централизованной архитектурой;
- СУБД с архитектурой файл-сервер;
- СУБД с архитектурой клиент-сервер;
- СУБД с трехуровневой архитектурой: "тонкий клиент" - сервер приложений - сервер базы данных.

В СУБД с ***централизованной архитектурой***, СУБД и сама база данных размещается и функционирует центральном миникомпьютере (мэйнфрейме), а пользователи получают доступ к базе данных при помощи обычных терминалов - компьютер рассматривается просто как устройство ввода и отображения информации: на мэйнфрейм передаются нажатия клавиш, в обратном направлении передаются данные, отображаемые непосредственно на мониторе пользователя. Примерами СУБД с централизованной архитектурой являются ранние версии СУБД DB2, ранние версии СУБД Oracle и Ingres.

В СУБД с ***архитектурой файл-сервер*** база данных хранится на сервере, а копии СУБД устанавливаются на компьютерах пользователей. Файл базы данных, находящийся на сервере, совместно используется всеми пользователями одновременно, при помощи сетевого программного обеспечения и самой операционной системы.

Ярким примером такой архитектуры является СУБД MS Access: копии СУБД установлены на компьютере каждого пользователя, а сам файл базы данных находится на сервере в сетевой папке.

Архитектура файл-сервер позволяет добиться приемлемой производительности, т.к. в распоряжении каждой копии СУБД находятся все ресурсы компьютера пользователя. С другой стороны, производительность такой схемы для каждого пользователя, напрямую зависит от характеристик компьютера пользователя.

Кроме того, такая схема работы значительно загружает сеть. Допустим, что пользователю необходимо отобрать строки таблицы с товарами, по которым объем продаж не превышает 100 тыс. руб. Поскольку строки в таблице не упорядочены, то скорее всего по сети будут переданы **все**, строки таблицы, из которых СУБД уже "на месте" (на компьютере пользователя) отберет **нужные**.

Очевидно, что такая схема нерациональна при больших объемах обрабатываемой информации или большом числе пользователей базы данных. Поэтому, для таких БД целесообразнее применять архитектуру **клиент-сервер**.

При ***архитектуре клиент-сервер*** база данных хранится на сервере, а СУБД подразделяется на две части: клиентскую и серверную. **Клиентская часть** СУБД выполняется на стороне клиента и обеспечивает интерактивное взаимодействие с пользователем и формирование запросов к базе данных (на языке SQL). **Серверная часть** работает на сервере и взаимодействует с базой данных, обеспечивая выполнение запросов клиентской части.

Т.е., если провести аналогию с рассмотренным выше примером, то **клиентская часть** сформирует и отправит серверной части запрос «Отбери для меня строки таблицы с товарами, по которым объем продаж не превышает 100 тыс. руб», **серверная часть** выполнит данный запрос и отошлет клиентской части только те строки, которые необходимо, не передавая по сети все строки таблицы. Большинство современных СУБД реализованы по архитектуре клиент-сервер: Oracle, MS SQL Server, PostgreSQL, Informix, DB2 и др.

Однако и архитектура клиент-сервер не лишена недостатков. Если деловая логика взаимодействия с базой данных (логика, определяющаяся порядком работы предприятия: какие таблицы и в каком порядке заполнять, что делать при добавлении нового сотрудника и т.д.) изменяется, то приходится заново переписывать клиентские программы (вводить новые формы, менять порядок их заполнения и т.д.).

Если изменения происходят слишком часто, а количество рабочих мест велико, то постоянная переустановка программного обеспечения (которая, кстати, должна осуществляться достаточно быстро) становится серьезной проблемой. В таких случаях следует переходить к **трехуровневой архитектуре: "тонкий клиент" – сервер приложений – сервер базы данных**.

При ***трехуровневой архитектуре*** в функции **клиентской части** ("тонкий клиент") входит только интерактивное взаимодействие с пользователем, а вся деловая логика вынесена на **сервер приложений**, который собственно и обеспечивает формирование запросов к базе данных, передаваемых на выполнение серверу базы данных.

"**Тонкий клиент**" находится на компьютере пользователя и чаще всего представляет из себя Web-браузер (например, Internet Explorer). **Сервер приложений** находится на сервере и может являться специализированной программой (например, Oracle Forms Server) или обычным Web-сервером, вызывающим для обработки HTTP-запроса внешнюю программу через интерфейс CGI (см. рис. 1.4).

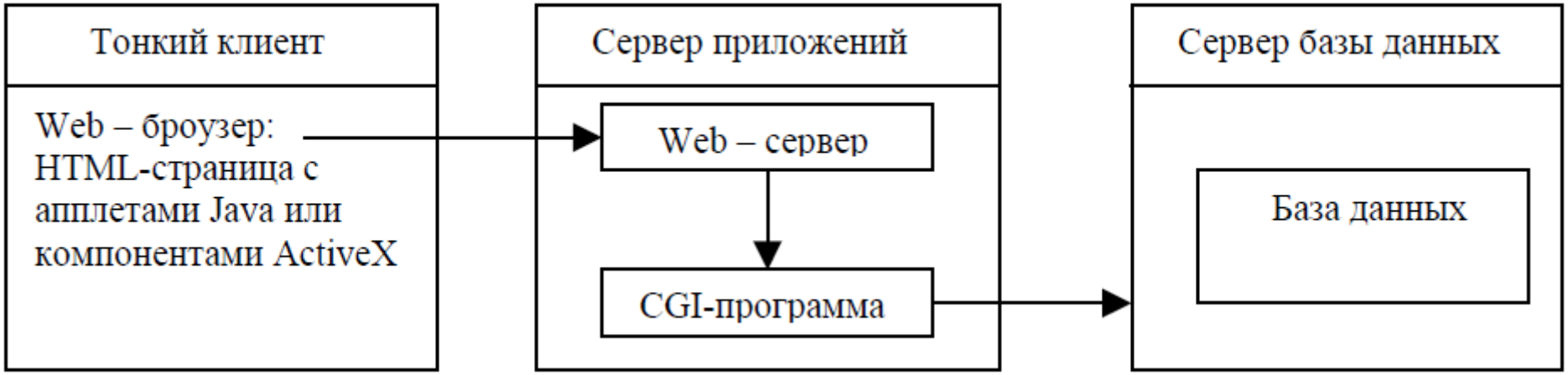

Рисунок 1.4 – Трехуровневая архитектура

Преимущества трехуровневой архитектуры очевидны: при необходимости изменений в деловой логике, изменения вносятся только один раз - на сервере приложений. Изменять или переустанавливать клиентские программы нет никакой необходимости.