**Наращивание экономической и статистической информации в двухструктурных реляционных базах данных**

СОДЕРЖАНИЕ

Введение

1. Понятие информационной системы.

2. Понятие базы данных.

3. Эволюция концепций баз данных

4. Требования, которым должна удовлетворять организация базы данных.

4.1. Установление многосторонних связей

4.2. Производительность

4.3. Минимальные затраты

4.4. Минимальная избыточность

4.5. Возможности поиска

4.6. Целостность

4.7. Безопасность и секретность

4.8. Связь с прошлым

4.9. Связь с будущим

4.10. Простота использования

5. Модели представления данных

5.1. Иерархическая модель данных

5.2. Сетевая модель данных

5.3. Реляционная модель данных

5.3.1. Таблицы

5.3.2. Ключевые поля

5.3.3. Индексы

5.3.4. Отношения предок/потомок

5.3.5. Внешние ключи

5.3.6. Реляционная алгебра

5.3.7. Нормализация базы данных

5.3.7.1. Первая нормальная форма

5.3.7.2. Вторая нормальная форма

5.3.7.3. Третья нормальная форма

5.3.7.4. Четвертая нормальная форма

5.3.7.5. Пятая нормальная форма

6. Язык SQL как стандартный язык баз данных.

6.1. Язык SQL

6.2. Достоинства SQL

6.2.1. Независимость от конкретных СУБД

6.2.2. Переносимость с одной вычислительной системы на другие

6.2.3. Стандарты языка SQL

6.2.4. Одобрение SQL компанией IBM (СУБД DB2)

6.2.5. Протокол ODBC и компания Microsoft

6.2.6. Реляционная основа

6.2.7. Высокоуровневая структура, напоминающая английский язык

6.2.8. Интерактивные запросы

6.2.9. Программный доступ к базе данных

6.2.10. Различные представления данных

6.2.11. Полноценный язык для работы с базами данных

6.2.12. Динамическое определение данных

6.2.13. Архитектура клиент/сервер

7. Архитектуры баз данных

7.1. Локальные базы данных и архитектура "файл-сервер"

7.2. Удаленные базы данных и архитектура "клиент-сервер"

8. Среда Delphi как средство для разработки СУБД

8.1. Высокопроизводительный компилятор в машинный код

8.2. Мощный объектно-ориентированный язык

8.3. Объектно-ориентированная модель программных компонент

8.4. Библиотека визуальных компонент

8.5. Формы, модули и метод разработки “Two-Way Tools”

8.6. Масштабируемые средства для построения баз данных

8.7. Настраиваемая среда разработчика

8.8. Незначительные требования к аппаратным средствам

9. Проектирование базы данных

Инфологическая модель данных

9.2. Инфологическая модель данных "сущность-связь"

9.3. Даталогическая модель данных

9.4. Переход от ER – модели к реляционной.

9.5. Физическая модель данных

9.6. Этапы проектирования базы данных

10. Практическая часть

10.1. Предметная область и задачи, возложенные на базу данных

10.2. Определение объектов базы данных

10.3. Инфологическая и даталогическая модели базы данных

10.4. Физическое описание модели

10.5. Програмная реализация

Заключение

Список литературы

**ВВЕДЕНИЕ**

Опыт применения компьютеров для построения прикладных систем обработки данных показывает, что самым эффективным инструментом здесь являются системы управления базами данных (*СУБД, англ. DBMS – DataBase Management System* ).

Потоки информации, циркулирующие в мире, который нас окружает, огромны. Во времени они имеют тенденцию к увеличению. Поэтому в любой организации, как большой, так и маленькой, возникает проблема такой организации управления данными, которая обеспечила бы наиболее эффективную работу. Некоторые организации используют для этого шкафы с папками, но большинство предпочитают компьютеризированные способы – базы данных, позволяющие эф­фективно хранить, структурировать и систематизировать большие объемы дан­ных. И уже сегодня без баз данных невозможно представить работу большинства финансовых, промышленных, торговых и прочих организаций. Не будь баз данных, они бы просто захлебнулись в информационной лавине.

Существует много веских причин перевода существующей информации на компьютерную основу. Сейчас стоимость хранения информации в файлах на компьютере дешевле, чем на бумаге. Базы данных позволяют хранить, структурировать информацию и извлекать оптимальным для пользователя образом. Использование клиент/серверных технологий позволяют сберечь значительные средства, а главное и время для получения необходимой информации, а также упрощают доступ и ведение, поскольку они основываются на комплексной обработке данных и централизации их хранения. Кроме того компьютер позволяет хранить любые форматы данных текст, чертежи, данные в рукописной форме, фотографии, записи голоса и т.д.

Для использования столь огромных объемов хранимой информации, помимо развития системных устройств, средств передачи данных, памяти необходимы средства обеспечения диалога человек-компьютер, которые позволяют пользователю вводить запросы, читать файлы, модифицировать хранимые данные, добавлять новые данные или принимать решения на основании хранимых данных. Для обеспечения этих функций созданы специализированные средства – системы управления базами данных (СУБД). Современные СУБД - многопользовательские системы управления базой данных, которые специализируется на управлении массивом ин­формации одним или множеством одновременно работающих пользователей.

Наращивание экономической и статической информации происходит ежедневно и ежесекундно. Если раньше, в связи с недостаточной компьютеризацией экономики, информации в электронном виде было очень мало, то сегодня это уже обычное дело. В связи с этим возникает новая проблема – поиск и отбор нужной информации среди того океана данных, которые мы можем сегодня наблюдать в Интернете и локальных корпоративных сетях. Поэтому правильная организация наращивания экономической и статической информации для дальнейшего её быстрого извлечения и эффективного использования - очень актуальная тема сегодня.

Цель данной дипломной работы – дать оценку новым технологиям организации накопления, сбережения, быстрого поиска, отбора и извлечения информации, которые базируются на реляционной концепции моделей данных, и на конкретном примере показать преимущества одной из рассмотренных технологий.

Реализация данной задачи проводится в системе программирования Delphi 5.0, располагающей широкими возможностями по созданию приложений баз данных, необходимым набором драйверов для доступа к самым известным форматам баз данных, удобными и развитыми средствами для доступа к информации, расположенной как на локальном диске, так и на удаленном сервере, а также большим коллекцией визуальных компонент для построения отображаемых на экране окон, что необходимо для создания удобного интерфейса между пользователем и исполняемым кодом.

# Понятие информационной системы

Веками человечество накапливало знания, навыки работы, сведения об окружающем мире, другими словами – собирало информацию. Вначале информация передавалась из поколения в поколение в виде преданий и устных рассказов. Возникновение и развитие книжного дела позволило передавать и хранить информацию в более надежном письменном виде. Открытия в области электричества привели к появлению телеграфа, телефона, радио, телевидения – средств, позволяющих оперативно передавать и накапливать информацию. Развитие прогресса обусловило резкий рост информации, в связи с чем, вопрос о ее сохранении и переработке становился год от года острее. С появлением вычислительной техники значительно упростились способы хранения, а главное, обработки информации. Развитие вычислительной техники на базе микропроцессоров приводит к совершенствованию компьютеров и программного обеспечения. Появляются программы, способные обработать большие потоки информации. С помощью таких программ создаются ***информационные системы***. Целью любой информационной системы является обработка данных об объектах и явлениях реального мира и предоставление человеку нужной информации о них.[11].

Если мы рассмотрим совокупность некоторых объектов, то сможем выделить объекты, обладающие одинаковыми свойствами. Такие объекты выделяют в отдельные классы. Внутри выделенного класса объекты можно упорядочивать как по общим правилам классифицирования, например по алфавиту, так и по некоторым конкретным общим признакам, например по цвету или материалу. Группировка объектов по определенным признакам значительно облегчает поиск и отбор информации. Все эти сведения накапливаются в совокупности файлов называемой базой данных, а для управления этими файлами создаются специальные программы – системы управления базами данных (СУБД).[10].

Информационные системы (ИС) можно условно разделить на фактографические и документальные.

В ***фактографических ИС*** регистрируются факты – конкретные значения данных (атрибутов) об объектах реального мира. Основная идея таких систем заключается в том, что все сведения об объектах (фамилии людей и названия предметов, числа, даты) сообщаются компьютеру в каком-то заранее обусловленном формате (например дата – в виде комбинации ДД.ММ.ГГГГ). Информация, с которой работает фактографическая ИС, имеет четкую структуру, позволяющую машине отличать одно данное от другого, например фамилию от должности человека, дату рождения от роста и т.п. Поэтому фактографическая система способна давать однозначные ответы на поставленные вопросы.

***Документальные ИС*** обслуживают принципиально иной класс задач, которые не предполагают однозначного ответа на поставленный вопрос. Базу данных таких систем образует совокупность неструктурированных текстовых документов (статьи, книги, рефераты и т.д.) и графических объектов, снабженная тем или иным формализованным аппаратом поиска. Цель системы, как правило, - выдать в ответ на запрос пользователя список документов или объектов, в какой-то мере удовлетворяющих сформулированным в запросе условиям.

Указанная классификация ИС в известной мере устарела, так как современные фактографические системы часто работают с неструктурированными блоками информации (текстами, графикой, звуком, видео), снабженными структурированными описателями. При известных факторах фактографическая система может превратиться в документальную (и наоборот).[1,11].

Для систем обработки экономической и статистической информации больше подходят фактографические ИС, которые используются буквально во всех сферах человеческой деятельности.

# Понятие базы данных.

Существует хорошо известное, но трудно реализуемое на практике понятие базы данных как большого по объему хранили­ща, в которое организация помещает все необходимые ей данные и из которого различные пользователи могут эти данные получать. Устройства памяти, в которых хранятся все данные, могут быть расположены в одном или нескольких местах; в последнем слу­чае они должны быть связаны средствами передачи данных. К дан­ным должны иметь доступ программы.

Действительно, большинство существующих на сегодняшний день баз данных предназначено для ограниченного ряда приложений. Часто на одном компьютере создается несколько баз данных. Со временем базы данных, предназначенные для реализа­ции отдельных родственных функций, можно будет объединить, если такое объединение будет способствовать увеличению эффективности и интенсивности использования всей системы.

***Базу данных*** *можно определить как совокупность взаимосвя­занных хранящихся вместе данных при наличии такой минималь­ной избыточности, которая допускает их использование оптималь­ным обра**зом для одного или нескольких приложений; данные за­поминаются так, чтобы они были независимы от программ,* *использующих эти данные; для добавления новых или модифика­**ции существующих данных, а также для поиска данных в базе* *данных применяется общий управляемый способ.* [1,12].

Говорят, что система содержит совокупность баз данных, если эти базы дан­ных структурно полностью самостоятельны. В системах с простой организацией данных для каждого при­ложения создается своя совокупность записей. Назначение базы данных заключается в том, чтобы одну и ту же совокупность дан­ных можно было использовать для максимально возможного числа приложений. Исходя из этого, базу данных часто разрабатывают в качестве хранилища такой информации, необходимость в котором возникает в процессе выполнения определенных функ­ций на заводе, правительственном учреждении или какой-либо другой организации. Такая база данных должна обеспечивать возможность не только получения информации, но также постоян­ной ее модификации, необходимой для процессов управления в данной организации, может оказаться, что для получения информации для целей планирования или ответов на вопросы потребу­ется осуществлять поиск в базе данных. Совокупностью данных могут пользоваться несколько ведомств независимо от того, име­ются ли при этом между ними ведомственные барьеры.[12].

База данных может разрабатываться для пакетной обработки данных, обработки в реальном времени или оперативной обработ­ки (в этом случае обработка каждого запроса завершается к определенному моменту времени, но при этом на время обра­ботки не накладывается жестких ограничений, существующих в системах реального времени). Во многих базах данных преду­смотрена совокупность этих методов обработки, а во многих си­стемах с базами данных обслуживание терминалов в реальном времени происходит одновременно с пакетной обработкой данных.[2].

Большая часть дисковых или ленточных библиотек, которые существовали до использования средств управления базами дан­ных, содержали большое количество повторяющейся информации. При запоминании многих элементов данных допускалась избы­точность, так как на носители информации для различных целей записывались одни и те же данные и, кроме того, хранились различные варианты модификаций одних и тех же данных. База данных предоставляет возможность в значительной степени изба­виться от такой избыточности. Базу данных иногда определяют как ***неизбыточную*** совокупность элементов данных. Однако в действительности для уменьшения времени доступа к данным или упрощения способов адресации во многих базах данных избыточ­ность в незначительной степени присутствует. Некоторые записи повторяются для того, чтобы обеспечить возможность восстанов­ления данных при их случайной потере. Чтобы база данных была неизбыточной и удовлетворяла другим требованиям, приходится идти на компромисс. В этом случае говорят об управляемой, или минимальной, избыточности или о том, что хорошо разрабо­танная база данных свободна от излишней избыточности.

Неуправляемая избыточность имеет несколько недостатков. Во-первых, хранение нескольких копий данных приводит к допол­нительным затратам. Во-вторых, при обновлении, по крайней мере, нескольких избыточных копий необходимо выполнять многократные операции обновления. Из­быточность поэтому обходится значительно дороже в тех случаях, когда при обработке файлов обновляется большое количество ин­формации или, что еще хуже, часто вводятся новые элементы или уничтожаются старые. В-третьих, вследствие того, что различные копии данных могут соответствовать различным стадиям обнов­ления, информация, выдаваемая системой, может быть противо­речивой.[12].

Если не использовать базы данных, то при обработке большого количества информации появится так много избыточных данных, что фактически станет невозможным сохранять их все на одном и том же уровне обновления. Очень часто пользователи обнаружи­вают явные противоречия в данных и поэтому испытывают недо­верие к полученной от компьютера информации. Невозможность хране­ния избыточных данных на одинаковом уровне обновления явля­ется основным препятствием в обработке данных с помощью компьютера.

Одной из наиболее важных характеристик большинства баз данных является их постоянное изменение и расширение. По мере добавления новых типов данных или при появлении новых прило­жений должна быть обеспечена возможность быстрого изменения структуры базы данных. Реорганизация базы данных должна осу­ществляться по возможности без перезаписи прикладных про­грамм и в целом вызывать минимальное количество преобразова­ний. Простота изменения базы данных может оказать большое влияние на развитие приложений баз данных в управлении про­изводством.[10].

О ***независимости данных*** часто говорят как об одном из основ­ных свойств базы данных. Под этим подразумевается независи­мость данных и использующих их прикладных программ друг от друга в том смысле, что изменение одних не приводит к измене­нию других. В частности, прикладной программист изолирован от влияния изменений данных и их организации, а также от изме­нения характеристик физических устройств, на которых они хра­нятся. В действительности же полностью независимыми данные бывают так же редко, как и полностью неизбыточными. Как мы увидим ниже, независимость данных определяется с различных точек зрения. Сведения, которыми должен располагать програм­мист для доступа к данным, различны для различных баз данных. Тем не менее, независимость данных—это одна из основных причин использования систем управления базами данных.

В том случае, когда один набор элементов данных использу­ется для многих приложений, между элементами этого набора устанавливается множество различных взаимосвязей, необходи­мых для соответствующих прикладных программ. Организация базы данных в значительной степени зависит от реализации ***взаи­мосвязей*** между элементами данных и записями, а также от того, как и где эти данные хранятся. В базе данных, используемой мно­гими приложениями, должны быть установлены многочисленные промежуточные взаимосвязи между элементами. В этом случае при хранении и использовании данных контролировать их пра­вильность, обеспечивать их защиту и секретность труднее, чем при хранении данных в простых, несвязанных файлах. Что касается обеспечения секретности данных и восстановления их после сбоев, то этот вопрос является очень важным при конструировании баз данных.[8].

В некоторых системах средства управления базами данных применяются для того, чтобы пользователи могли использовать данные таким путем, который не был предусмотрен разработчи­ками системы. Администраторы или сотрудники могут обращаться к вы­числительной системе с вопросами, которые заранее в ней не пре­дусматривались. Наличие этой возможности означает такую орга­низацию данных в системе, при которой доступ к ним можно осу­ществлять по различным путям, причем одни и те же данные могут использоваться для ответов на различные вопросы. Вся существенная информация об объектах запоминается одновременно и полностью, а не только та ее часть, которая необходима для одного приложения. [10].

В настоящее время существуют СУБД, реализующие эти возможности как на уровне локальных баз данных, расположенных на одном диске (Paradox, Dbase), так и промышленных баз данных (Acsess, Oracle, FoxPro).

# Эволюция концепций баз данных

Понятие *база данных* появилось в конце 60-х годов. До этого в сфере обработки данных говорили о файлах данных и о наборах данных.

До появления компьютеров третьего поколения (первые из них были установлены в 1965 г.) программное обеспечение обработки данных осуществляло в основном операции ввода-вывода. 0б организации данных приходилось заботиться при напи­сании прикладных программ, и делалось это элементарным спо­собом, т. е. данные обычно организовывались в виде простых последовательных файлов на магнитной ленте. Независимость данных отсутствовала. Если организация данных или запоминаю­щие устройства изменялись, прикладной программист должен был соответствующим образом модифицировать программы, заново их компилировать и затем отлаживать.  Для того чтобы обновить файл, нужно было записать новый. Старый файл сохранялся и назывался исходным. Предыдущий вариант также сохранялся, а нередко сохранялись и более ранние версии файла. Многие файлы использовались для одного приложения. Для других приложений часто использовали те же самые данные, но обычно в другой форме, с другими полями, и поэтому приходилось из одних и *т**ех* же данных создавать различные файлы. Вследствие этого уровень избыточности в системе был очень высок и существовали различные файлы, содержащие одни и те же элементы дан­ных.

Иногда использовались файлы с произвольным досту­пом к данным, которые позволяли пользователю получить непосредственный доступ к любой записи в файле вместо того, чтобы последовательно просматривать весь файл. Средства адресации записей обеспечивались прикладным программистом при написа­нии программы. Если изменялись запоминающие устройства, в прикладную программу необходимо было вносить большие изме­нения. На практике изменение запоминающих устройств неизбеж­но. Новая технология привела к значительному уменьшению за­трат на хранение одного бита информации, а размеры файлов сегодня часто превышают по объему использовавшиеся ранее за­поминающие устройства.[7].

Этап 2 (конец 60-х годов) характеризуется изменением по сравнению с этапом 1 как природы файлов, так и устройств, на которых они запоминались. Предпринимается попытка оградить прикладного программиста от влияния изменений в аппаратуре. Программное обеспечение допускает возможность изменения физического расположения данных без изменения при этом их логического представления при условии, что содержимое записей или основная структура файлов не изменяется.

Файлы, соответствующие этому этапу развития средств обра­ботки данных, подобно файлам этапа 1, предназначаются для одного приложения или для тесно связанных между собой прило­жений.

По мере развития средств обработки коммерческих данных становилось ясно, что прикладные программы желательно сделать независимыми не только от изменений в аппаратных средствах хранения файлов и от увеличения размеров файлов, но также и от добавления к хранимым данным новых полей и новых взаимо­связей.[7].

Известно, что база данных представляет собой постоянно развивающийся объект, который используется возрастающим количеством приложений. К базе данных добавляются новые записи, а в существующие записи включаются новые элементы данных. Структура базы данных будет изменяться с целью повышения эффективности ее функциони­рования и при добавлении новых типов запросов. Пользователи будут изменять требования и модифицировать типы запросов на данные.

Структура базы данных является менее статичной, чем файло­вая структура. Элементы хранимых данных и способы их запоми­нания непрерывно изменяются. Если на организацию данных со стороны вычислительной системы накладывается ограничение в виде требования постоянства файловой структуры, то это приво­дит к тому, что в случае ее изменения программисты тратят много времени на модификацию существующих программ, вместо того чтобы заниматься разработкой новых приложений.

В одном случае может сообщаться только имя элемента данных или записи, которую он хочет по­лучить. В другом случае (при наличии другого программного обеспечения) он должен был сообщать идентификацию элемента данных и имя набора, в котором этот элемент данных содержится. Добавление новых элементов данных в записи без изменения прикладных программ возможно при том условии, что программ­ное обеспечение связано с данными на уровне элементов данных (полей), а не на уровне записей. Это часто приводит к созданию сложных структур дан­ных. Однако хорошее программное обеспечение баз данных из­бавляет прикладного программиста от трудностей, связанных со сложностью структуры. Независимо от того, каким образом дан­ные организованы на самом деле, прикладной программист дол­жен представлять себе файл в виде сравнительно простой струк­туры, которая спланирована в соответствии с его требованиями.

Программное обеспечение баз данных этапа 3 (начало 70-х годов) распо­лагало средствами отображения файловой структуры прикладного программиста в такую физическую структуру данных, которая запоминается на реальном носителе и наоборот.

В зависимости от уровня программного обеспече­ния прикладной программист эле­мента данных должен также знать организацию файла данных. В этом случае ему, возможно, придется задать машинный адрес данных. Если отсутствует независимость данных, прикладному программисту необходимо знать точный физический формат запи­си. Самый худший вариант — это случай, когда программист дол­жен быть “навигатором”.[7].

Процесс преобразования обращения прикладного программи­ста к логической записи или к элементам логической записи в машинные обращения к физической записи и ее элементам называется ***привязкой****.* Привязка — это связь физического представле­ния данных с программой, которая эти данные использует. После выполнения процесса привязки программа уже не будет незави­симой от физических данных.[7, 3].

Итак, для 3-го этапа:

* Различные логические файлы могли быть получены из одних и тех же физических данных.
* Доступ к одним и тем же данным осуществлялся различными приложениями различными путями, отвечающими требованиям этих приложений.
* Программное обеспечение содержало средства уменьшения избыточно­сти данных.
* Элементы данных являлись общими для различных приложений.
* Физическая структура данных независима от прикладных программ. Ее можно было изменять с целью повышения эффективности базы данных, не вызывая при этом модификации прикладных программ,
* Данные адресуются на уровне полей или групп. [7].

По мере накопления опыта использования первых систем  управления базами данных довольно скоро стало очевидно, что не­обходим дополнительный уровень независимости данных. Общая логическая структура данных, как правило, сложная, и по мере роста базы данных она неизбежно изменяется. Поэтому важно обеспечить возможность изменения общей логической структуры без изменения используемых при этом многочисленных приклад­ных программ. В некоторых системах изменение общей логиче­ской структуры данных составляет форму ее существования, т. е. эта структура находится в состоянии постоянного развития. По­этому требуются два уровня независимости данных. Их называют ***логической*** и ***физической*** независимостью данных.

***Логическая независимость данных*** означает, что общая логи­ческая структура данных может быть изменена без изменения при­кладных программ (изменение, конечно, не должно заключаться в удалении из базы данных таких элементов, которые использу­ются прикладными программами).

***Физическая независимость данных*** означает, что физическое расположение и организация данных могут изменяться, не вызы­вая при этом изменений ни общей логической структуры данных, ни прикладных программ.[7, 8, 3].

Этап 4 характеризуется идей логической и физи­ческой независимости данных; логическая структура данных может сильно отличаться от физической структуры данных и от их пред­ставлений в конкретных прикладных программах. Программное обеспечение баз данных будет фактически преобразовывать пред­ставление данных прикладного программиста в общее логическое представление, а затем будет отображать логическое представле­ние в физическое представление данных.

Назначение такой структуры обеспечивает максимум свободы в изменении структур данных без переделки при этом выполненной ранее работы по формированию и использованию базы данных.

* База данных может развиваться без больших затрат на ведение.
* Средства, предусмотренные для администратора данных, позволяют ему выполнять функции контроллера и обеспечивать сохранность данных.
* Обеспечиваются эффективные процедуры управления защитой секрет­ности, целостности и безопасности данных.
* В некоторых системах используются инвертированные файлы, позво­ляющие осуществлять быстрый поиск данных в базе данных.
* Базы данных конструируются для выдачи ответов на не планируемые заранее информационные запросы.
* Обеспечиваются средства перемещения данных.[7].

# Требования, которым должна удовлетворять организация базы данных.

Изучением этого вопроса долгое время занимались различные группы людей в учреждениях, использующих компьютеры, в правитель­ственных комиссиях, на вычислительных центрах коллективного пользования. Комитет CODASYL опубликовал отчеты на эту тему (CODASYL—организация, разработавшая язык КОБОЛ). Организации пользователей IBM SHARE и GUIDE в своем отчете сформулировали требования к системе управления базами дан­ных. Организация ACiM (Association for Computing Machi­nery) также занималась изучением этого вопроса.

Ниже перечислены основные требования к организации базы данных.

## Установление многосторонних связей

Различным программистам требуются различные логические файлы. Эти файлы получаются из одной и той же совокупности данных. Между элементами запоминаемых данных могут суще­ствовать различные связи. Некоторые базы данных будут содер­жать сложные переплетения взаимосвязей. Метод организации данных должен быть таким, чтобы обеспечивалась возможность удобного представления этих взаимосвязей и быстрого согласова­ния вносимых в них изменений. Система управления базами дан­ных должна обеспечивать возможность получения требуемых логи­ческих файлов из имеющихся данных и существующих между ними связей. Необходимо, чтобы существовало хотя бы небольшое сходство между представлением логического файла в прикладной программе и способом физического хранения данных.[7, 10, 11].

## Производительность

Базы данных, специально разработанные для использования их оператором терминала, обеспечивают время ответа, удовлет­ворительное для диалога человека — терминал. Кроме того, система баз данных должна обеспечивать соответствующую пропуск­ную способность. В системах, рассчитанных на небольшой поток запросов, пропускная способность накладывает незначительные ограничения на структуру базы данных. В системах с большим потоком запросов, например в системах резервирования авиа­билетов, пропускная способность оказывает решающее влияние на выбор организации физического хранения данных.

В системах, предназначенных только для пакетной обработки, время ответа не так важно и метод физической организации мо­жет выбираться из условий обеспечения эффективной пакетной обработки.[7, 10, 11].

## Минимальные затраты

Для уменьшения затрат на создание и эксплуатацию базы данных выбираются такие методы организации, которые миними­зируют требования к внешней памяти. При использовании этих методов физическое представление данных в памяти может сильно отличаться от того представления, которое использует прикладной программист. Преобразование одного представления в другое осу­ществляют программное обеспечение либо, если возможно, аппа­ратные или микропрограммные средства. В таких случаях прихо­дится выбирать между затратами на алгоритм преобразования и экономией памяти.[7, 10, 11].

## Минимальная избыточность

В системах обработки, существовавших до использования си­стем управления базами данных, информационные фонды облада­ли очень высоким уровнем избыточности. Большинство ленточных библиотек содержало большое количество избыточных данных. Даже при использовании баз данных по мере возрастания инфор­мации, объединяемой в интегрированные базы данных, потен­циальная возможность появления избыточных данных постепенно увеличивается. Избыточные данные дороги в том смысле, что они занимают больше памяти, чем это необходи­мо, и требуют более одной операции обновления. Целью организации базы данных должно быть уничтожение избыточных данных там, где это выгодно, и контроль за теми про­тиворечиями, которые вызываются наличием избыточных данных.[7, 10, 11].

## Возможности поиска

Пользователь базы данных может обращаться к ней с самыми различными вопросами по поводу хранимых данных. В большин­стве современных коммерческих приложений типы запросов предо­пределены, и физическая организация данных разрабатывается для их обработки с требуемой скоростью. Возросшие требования к системам заключаются в обеспечении обработки таких запро­сов или формирования таких ответов, которые заранее не запла­нированы. [7, 10, 11].

## Целостность

Если база данных содержит данные, используемые многими пользователями, очень важно, чтобы элементы данных и связи между ними не разрушались. Необходимо учитывать возможность возникновения ошибок и различного рода случайных сбоев. Хра­нение данных, их обновление, процедуры включения данных должны быть такими, чтобы система в случае возникновения сбоев могла восстанавливать данные без потерь. Необходимо, чтобы вы­числительная система гарантировала целостность хранимых в ней данных.[7, 10, 11].

## Безопасность и секретность

Данные в системах баз данных должны храниться в тайне и сохранности. Запоминаемая информация иногда очень важна для использующего ее учреждения. Она не должна быть утеряна или похищена. Для увеличения жизнестойкости информации в базе данных важно защищать ее от аппаратных или программных сбоев, от катастрофических и криминальных ситуаций, от неком­петентного или злонамеренного использования лицами, которые могут ее неправильно употребить.

*Под* ***безопасностью*** *данных понимают защиту данных от слу­чайного или преднамеренного доступа к ним лиц, не имеющих на это право, от неавторизованной модификации данных или их унич­**тож**ения.*

***Секретность*** *определяют как право отдельных лиц или орга­низаций определять, когда, как и какое количество соответствую­щей информации может быть передано другим лицам или орга­низациям.*[7, 10, 11].

## Связь с прошлым

Организации, которые в течение какого-то времени эксплуати­руют системы обработки данных, затрачивают значительные сред­ства на написание программ, процедур и организацию хранения данных. В том случае, когда фирма начинает использовать на вычислительной установке новое программное обеспечение управ­ления базами данных, очень важно, чтобы при этом она могла работать с уже существующими на этой установке программами, обрабатываемые данные можно было бы соответствующим образом преобразовывать. Такое условие требует наличия програм­мной и информационной совместимости, и ее отсутствие может стать основным сдерживающим фактором при переходе к новым системам управления базами данных. Важно, однако, чтобы про­блема связи с прошлым не сдерживала развитие средств управ­ления базами данных. [7, 10, 11].

## Связь с будущим

Особенно важной представляется связь с будущим. В будущем данные и среда их хранения изменятся по многим направлениям. Любая коммерческая организация со временем претерпевает из­менения. Особенно дорогими эти изменения оказываются для пользователей системами обработки данных. Огромные затраты, которые требуются для реализации самых простых изменений, сильно тормозят развитие этих систем. Эти затраты расходуются на преобразование данных, перезапись и отладку прикладных программ, явившихся результатом внесения изменений. Со време­нем число прикладных программ в организации растет, и поэтому перспектива перезаписи всех этих программ кажется нереальной. Одна из самых важных задач при разработке баз данных—запла­нировать базу данных таким образом, чтобы изменения ее можно было выполнять без модификации прикладных программ.[7, 10, 11].

## Простота использования

Средства, которые используются для представления общего логического описания данных, должны быть простыми и изящны­ми.

Интерфейс программного обеспечения должен быть ориентирован на конечного пользователя и учитывать возможность того, что пользователь не имеет необходимой базы знаний по теории баз данных. [7, 10, 11].

# Модели представления данных

С ростом популярности СУБД в 70-80-х годах появилось множество различных моделей данных. У каждой из них имелись свои достоинства и недостатки, которые сыграли ключевую роль в развитии реляционной модели данных, появившейся во многом благодаря стремлению упростить и упорядочить первые модели данных.

Современные БД основываются на  использовании моделей данных (МД), позволяющих описывать объекты предметных областей и взаимосвязи между ними существуют три основные МД и их комбинации, на которых основываются БД: реляционная модель данных (РМД), сетевая модель данных (СМД), иерархическая модель данных (ИМД).

Основное различие между этими моделями данных состоит в способах описания взаимодействий между объектами и  атрибутами. Взаимосвязь выражает отношение между множествами данных.

Используют взаимосвязи "один к одному", "один ко многим" и "многие ко многим". ***"Один к одному"*** - это взаимно однозначное соответствие, которое устанавливается между одним объектом и одним  атрибутом. ***"Один ко многим"*** - это соответствие между одним объектом и многими атрибутами. ***"Многие ко многим"*** - это соответствие между многими объектами и многими атрибутами. [10, 11, 12].

Рассмотрим эти модели данных более подробно.

## Иерархическая модель данных

ИМД основана  на  понятии деревьев, состоящих из вершин и ребер. Вершине  дерева  ставится в соответствие совокупности атрибутов данных, характеризующих некоторый объект. Вершины и ребра  дерева  как бы образуют иерархическую древовидную структуру, состоящую из n уровней.

Первую вершину называют корневой вершиной. Он  удовлетворяет условиям:

1. Иерархия начинается с корневой вершины.
2. Каждая вершина  соответствует одному или нескольким атрибутам.
3. Hа уровнях с большим номером находятся зависимые вершины. Вершин  предшествующего уровня является начальной для новых зависимых вершин.
4. Каждая вершина, находящаяся на  уровне i, соединена  с одной и только одной вершиной уровня i-1, за  исключением корневой вершины.
5. Корневая вершина  может быть связана  с одной или несколькими зависимыми вершинами.
6. Доступ к каждой вершине происходит через корневую по единственному пути
7. Существует произвольное количество вершин каждого уровня.

Иерархическая модель данных состоит из нескольких деревьев, т.е. является лесом. Каждая корневая вершин  образует начало записи логической базы данных. В ИМД вершины, находящиеся на  уровне i, называют порожденными вершин ми н  уровне i-1.

Операции в ИМД имеют  нелогичный позаписный характер. Аппарат перемещения по структуре в графовых моделях служит для установки тех объектов данных, к которым будет применяться очередная операция манипулирования данными. Такие объекты называются текущими. Механизмы доступа  к данным и перемещения по структуре данных в таких моделях достаточно сложны и существенным образом опираются на концепцию текущего состояния механизма  доступа.[7, 10, 11, 12].

Основные достоинства  ИМД: простота  построения и использования, обеспечение определенного уровня независимости данных, простота  оценки операционных характеристик. Основные недостатки: отношение "многие ко многим" реализуется очень сложно, дает громоздкую структуру и требует хранения избыточных данных, что особенно нежелательно на  физическом уровне, иерархическая упорядоченность усложняет операции удаления и включения, доступ к любой вершине возможен только через корневую, что увеличивает время доступа.

К числу СУБД иерархического типа  можно отнести PC/Focus, Team-Up, Data Edge,   также разработанную в нашей стране систему HИКА, преемницу широко распространенной советской системы ИHЕС для ЕС ЭВМ.

Одной из наиболее важных сфер применения первых иерархических СУБД было планирование производства для компаний, занимающихся выпуском продукции. Например, если автомобильная компания хотела выпустить 10000 машин одной модели и 5000 машин другой модели, ей необходимо было знать, сколько деталей следует заказать у своих поставщиков. Чтобы ответить на этот вопрос, необходимо определить, из каких деталей состоят эти части и т.д. Например, машина состоит из двигателя, корпуса и ходовой части; двигатель состоит из клапанов, цилиндров, свеч и т.д. Работа со списками составных частей была как будто специально предназначена для компьютеров.

Список составных частей изделия по своей природе является иерархической структурой. Для хранения данных, имеющих такую структуру, была разработана *иерархическая* модель данных, которую иллюстрирует рис. 1.

В этой модели каждая ***запись*** базы данных представляла конкретную деталь. Между записями существовали ***отношения предок*/*потомок***, связывающие каждую часть с деталями, входящими в неё.Чтобы получить доступ к данным, содержащимся в базе данных, программа могла:

*Рис. 1. Иерархическая база данных, содержащая информацию о составных частях*

Автомобиль

Двигатель

Корпус

Ходовая часть

Крыша

Днище

Правая дверь

Левая дверь

Замок

Окно

Ручка

Записи

* найти конкретную деталь (правую дверь) по её номеру;
* перейти "вниз" к первому потомку (ручка двери);
* перейти "вверх" к предку (корпус);
* перейти "в сторону" к другому потомку (правая дверь).

Таким образом, для чтения данных из иерархической базы данных требовалось перемещаться по записям, за один раз переходя на одну запись вверх, вниз или в сторону.

*Ограничения целостности.*

Автоматически поддерживается целостность ссылок между предками и потомками. *Основное правило: никакой потомок не может существовать без своего родителя.* Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается. [7, 9].

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии.

## Сетевая модель данных

Сетевая модель данных замышлялась как инструмент для пользователей баз данных - программистов. В связи с этим в СМД больше внимания уделяется структуризации данных, чем развитию ее операционных возможностей.

В СМД элементарные данные и отношения между ними представляются в виде ориентированной сети (вершины - данные, дуги - отношения).[7].

Если структура данных оказывалась сложнее, чем обычная иерархия, простота структуры иерархической базы данных становилась её недостатком. Например, в базе данных для хранения заказов один заказ мог участвовать в трёх различных отношениях предок/потомок, связывающих заказ с клиентом, разместившим его, со служащим, принявшим его, и с заказанным товаром, что иллюстрирует рис. 2. Такие структуры данных не соответствовали строгой иерархии IMS.

Acme

Mfg.

Bill

Adams

Size 4

Widget

#112963

Заказы

Клиенты

Ñëóæàèå

Товары

*Рис. 2. Множественные отношения предок/потомок*

В связи с этим для таких приложений, как обработка заказов, была разработана новая ***сетевая*** модель данных. Она являлась улучшенной иерархической моделью, в которой одна запись могла участвовать в нескольких отношениях предок/потомок. В сетевой модели такие отношения назывались ***множествами***. В 1971 году на конференции по языкам систем данных был опубликован официальный стандарт сетевых баз данных, который известен как модель CODASYL. Компания IBM не стала разрабатывать собственную сетевую СУБД и вместо этого продолжала наращивать возможность IMS. Но в 70-х годах независимые производители программного обеспечения реализовали сетевую модель в таких продуктах, как IDMS компании Cullinet, Total компании Cincom и СУБД Adabas, которые приобрели большую популярность.

Сетевые базы данных обладали рядом преимуществ:

* *Гибкость.* Множественные отношения предок/потомок позволяли сетевой базе данных хранить данные, структура которых была сложнее простой иерархии.
* *Стандартизация.* Появление стандарта CODASYL популярность сетевой модели, а такие поставщики мини-компьютеров, как Digital Equipment Corporation и Data General, реализовали сетевые СУБД.
* *Быстродействие.* Вопреки своей большой сложности, сетевые базы данных достигали быстродействия, сравнимого с быстродействием иерархических баз данных. Множества были представлены указателями на физические записи данных, и в некоторых системах администратор мог задать кластеризацию данных на основе множества отношений.

Конечно, у сетевых баз данных были недостатки. Как и иерархические базы данных, сетевые базе данных были очень жесткими. Наборы отношений и структуру записей приходилось задавать наперёд. Изменение структуры базы данных обычно означало перестройку всей базы данных.

Как иерархическая, так и сетевая база данных были инструментами программистов. Чтобы получить ответ на вопрос типа "Какой товар наиболее часто заказывает компания Acme Manufacturing?", программисту приходилось писать программу для навигации по базе данных. Реализация пользовательских запросов часто затягивалась на недели и месяцы, и к моменту появления программы информация, которую она предоставляла, часто оказывалась бесполезной.[7, 10].

*Ограничения целостности.*

В принципе их поддержание не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

## Реляционная модель данных

Недостатки иерархической и сетевой моделей привели к появлению новой, *реляционной* модели данных, созданной Коддом в 1970 году и вызвавшей всеобщий интерес. Реляционная модель была попыткой упростить структуру базы данных. В ней отсутствовали явные указатели на предков и потомков, а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы. На рис. 3. показана реляционная версия сетевой базы данных, содержащей информацию о заказах и приведенной на рис. 2.

К сожалению, практическое определение понятия "реляционная база данных" оказалось гораздо более расплывчатым, чем точное математическое определение, данное этому термину Коддом в 1970 году. В первых реляционных СУБД не были реализованы некоторые из ключевых частей модели Кодда, и этот пробел был восполнен только впоследствии. По мере роста популярности реляционной концепции реляционными стали называться многие базы данных, которые на деле таковыми не являлись.



|  |  |  |
| --- | --- | --- |
| Таблица CUSTOMERS | | |
| COMPANY | CUST\_REP | CREDIT\_LIMIT |
| JSP Inc. | 103 | $50,000.00 |
| First Corp. | 101 | $65,000.00 |

*Рис. 3. Реляционная база данных, содержащая информацию о заказах*

В ответ на неправильное использование термина "реляционный" Кодд в 1985 году написал статью, где сформулировал 12 правил, которым должна удовлетворять любая база данных, претендующая на звание реляционной. С тех пор двенадцать правил Кодда считаются определением реляционной СУБД. Однако можно сформулировать и более простое определение:

***Реляционной*** *называется база данных, в которой все данные, доступные пользователю, организованны в виде таблиц, а все операции над данными сводятся к операциям над этими таблицами*.

Приведенное определение не оставляет места встроенным указателям, имеющимся в иерархических и сетевых СУБД. Несмотря на это, реляционная СУБД также способна реализовать отношения предок/потомок, однако эти отношения представлены исключительно значениями данных, содержащихся в таблицах.

Поскольку в программной реализации дипломной работы избран реляционный подход, как наиболее подходящий, опишем его более подробно.[3, 7, 8, 12].

### Таблицы

***Таблицы*** – фундаментальные объекты реляционной базы данных, в которых хранится основная часть данных приложения. Отдельная таблица чаще всего хранит информацию по конкретной теме (например, сведения о служащих компании или адреса заказчиков). Информация в таблице организуется в строки (записи) и столбцы (поля). Таблице присущи два компонента: структура таблицы и данные таблицы.

***Структура таблицы*** (также называется определением таблицы) специфицируется при создании таблицы. Структура таблицы должна быть спроектирована и создана перед вводом в таблицу каких-либо данных. Она определяет, какие данные таблица будет хранить, а также правила, ассоциированные с вводом, изменением или удалением данных (*бизнес-правила*, или *ограничения*).

Структура таблицы включает следующую информацию:

* ***Имя таблицы*** - Имя, по которому к таблице можно обратиться в свойствах, методах и операторах SQL.
* ***Столбцы таблицы* -** Категории информации, сохраненной в таблице. Каждый столбец имеет имя и тип данного.
* ***Табличные и столбцовые ограничения* -** Ограничения целостности, определенные на уровне таблицы или на уровне столбца.[3, 7, 8, 12].

Более наглядно структуру таблицы иллюстрирует рис 4., на котором изображена таблица STUDENTS. Каждая горизонтальная *строка* этой таблицы представляет отдельную физическую сущность - одного студента. Все данные, содержащиеся в конкретной строке таблицы, относятся к студенту, который описывается этой строкой.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица STUDENTS | | |  |  |  |
| NUMBER | FAMILY | NAME | DATE | GROUPE | PODGRP |
| 22 | Denver | Western | 1.08.77 | 1 | 1 |
| 11 | ИВАНОВ | АЛЕКСЕЙ | 10.06.78 | 1 | 1 |
| 12 | ЖДАНОВ | СЕРГЕЙ | 14.03.71 | 1 | 2 |
| 13 | ПЕТРОВ | ВАДИМ | 25.10.75 | 3 | 2 |
| 21 | КРОХИН | МАКСИМ | 30.12.77 | 2 | 3 |

Фамилия

студента

Дата рождения

Номер подгруппы

Данные о подгруппе для ИВАНОВА

Данные о подгруппе для

КРОХИНА

*Рис. 4. Структура реляционной таблицы.*

Каждый вертикальный ***столбец*** таблицы STUDENTS представляет один элемент данных для каждого из студентов. Например, в столбце GROUP содержатся номера групп, в которых расположены студенты. В столбце DATE содержатся даты рождения каждого студента.

***Данные таблицы*** – информация, которая сохранена в таблице. Все данные таблицы хранятся в строках, каждая из которых содержит порции информации в столбцах, определенных в структуре таблицы. Данные – та часть таблицы, к которой обычно должны иметь доступ пользователи приложения (например, данные таблицы могут выводиться в элементах управления, размещенных в формах и отчетах).

На пересечении каждой строки с каждым столбцом таблицы содержится в точности одно значение данных. Например, во второй строке в столбце FAMILY содержится значение "ИВАНОВ". В столбце PODGRP той же строки содержится значение 1, которое является номером подгруппы, в которой находится данный студент.

Все значения, содержащиеся в одном и том же столбце, являются данными одного типа. Например, в столбце FAMILY содержатся только слова, в столбце DATE содержатся даты, а в столбце NUMBER содержатся целые числа, представляющие идентификаторы студентов. Множество значений, которые могут содержаться в столбце, называется ***доменом*** этого столбца. Доменом столбца FAMILY является множество фамилий студентов. Доменом столбца DATE является любая дата.

У каждого столбца в таблице есть своё ***имя***, которое обычно служит заголовком столбца. Все столбцы в одной таблице должны иметь уникальные имена, однако разрешается присваивать одинаковые имена столбцам, расположенным в различных таблицах. На практике такие имена столбцов, как NUMBER, FAMILY, NAME, GROUP, DATE, PODGRP, часто встречаются в различных таблицах одной базы данных.

Столбцы таблицы упорядочены слева направо, и их порядок определяется при создании таблицы. В любой таблице всегда есть как минимум один столбец. В стандарте ANSI/ISO не указывается максимально допустимое число столбцов в таблице, однако почти во всех коммерческих СУБД этот предел существует и обычно составляет примерно 255 столбцов.

В отличие от столбцов, строки таблицы не имеют определённого порядка. Это значит, что если последовательно выполнить два одинаковых запроса для отображения содержимого таблицы, нет гарантии, что оба раза строки будут перечислены в одном и том же порядке.

В таблице может содержаться любое количество строк. Вполне допустимо существование таблицы с нулевым количеством строк. Такая таблица называется ***пустой***. Пустая таблица сохраняет структуру, определённую её столбцами, просто в ней не содержится данные. Стандарт ANSI/ISO не накладывает ограничений на количество строк в таблице, и во многих СУБД размер таблиц ограничен лишь свободным дисковым пространством компьютера. В других СУБД имеется максимальный предел, однако он весьма высок - около двух миллиардов строк, а иногда и больше.[12].

### Ключевые поля

Мощь реляционных баз данных заключается в том, что с их помощью можно быстро найти и связать данные из разных таблиц при помощи запросов; форм и отчетов. Для этого каждая таблица должна содержать одно или несколько полей, однозначно идентифицирующих каждую запись в таблице. Эти поля называются ***ключевыми полями*** таблицы. Ключевые поля ещё также называют ***первичным ключом***. Можно выделить три типа ключевых полей: счетчик, простой ключ и составной ключ.

Поскольку строки в реляционной таблице не упорядочены, нельзя выбрать строку по ее номеру в таблице. В таблице нет "первой", "последней" или "тринадцатой" строки. Тогда каким же образом можно указать в таблице конкретную строку, например строку для студента с фамилией Иванов?

Ключевое поле можно задать таким образом, чтобы при добавлении каждой записи в таблицу в это поле автоматически вносилось порядковое число, т.е. организовать ***счётчик***. Это наиболее простой способ создания ключевых полей.

Если поле содержит уникальные значения, такие как коды или инвентарные номера, то это поле можно определить как ***простой ключ***. Если выбранное поле содержит повторяющиеся или пустые значения, то оно не будет определено как ключевое. Для определения записей, содержащих повторяющиеся данные, можно выполнить запрос на поиск повторяющихся записей. Если устранить повторы путем изменения значений невозможно, то следует либо добавить в таблицу поле счетчика и сделать его ключевым, либо определить составной ключ.[3, 7, 8, 12].

На первый взгляд, первичным ключом таблицы STUDENTS могут служить и столбец FAMILY. Однако в жизни довольно часто встречаются однофамильцы, следовательно, столбец FAMILY больше не может выполнять роль ключа. На практике в качестве первичных ключей таблиц обычно следует выбирать идентификаторы, такие как идентификатор студента NUMBER в таблице STUDENTS.

Таблица ORDERS, фрагмент которой показан на рис. 5., является примером таблицы, в которой первичный ключ представляет собой комбинацию столбцов. Такой первичный ключ называется ***составным ключом****.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица ORDERS | | |  |  |
| NSTUD | NORDER | TYPEORDER | DATE | MOTIVE |
| 10 | 2A45C | Отчислить | 07.09.94 | 1 |
| 22 | 4100Y | Восстановить | 27.02.88 | 2 |
| 22 | KX47 | Предоставить А/О | 30.05.87 | 3 |
| 37 | 41672 | Отчислить | 18.10.90 | 0 |

Первичный ключ

*Рис. 5. Пример таблицы с составным первичным ключом*

Он применяется в случаях, когда невозможно гарантировать уникальность значений каждого отдельного поля. Чаще всего такая ситуация возникает для таблицы, используемой для связывания двух таблиц в отношении "многие-ко-многим".

Столбец NSTUD содержит идентификаторы студентов, перечисленных в таблице, а столбец NORDER содержит номера, приказам. Может показаться, что столбец NORDER мог бы и один выполнять роль первичного ключа, однако ничто не мешает одному студенту несколько раз попасть под отчисление и затем восстановиться на факультете. Таким образом, в качестве первичного ключа таблицы ORDERS необходимо использовать комбинацию столбцов NSTUD и NORDER. Для каждого из студентов, содержащихся в таблице, комбинация значений в этих столбцах будет уникальной.

Первичный ключ для каждой строки таблицы является уникальным, поэтому в таблице с первичным ключом нет двух совершенно одинаковых строк. Таблица, в которой все строки отличаются друг от друга, в математических терминах называется ***отношением****.* Именно этому термину реляционные базы данных и обязаны своим названием, поскольку в их основе лежат отношения (таблицы с отличающимися друг от друга строками).

Хотя первичные ключи являются важной частью реляционной модели данных, в первых реляционных СУБД (System/R, DB2, Oracle и других) не была обеспечена явным образом их поддержка. Как правило, проектировщики базы данных сами следили за тем, чтобы у всех таблиц были первичные ключи, однако в самих СУБД не было возможности определить для таблицы первичный ключ. И только в СУБД DB2 Version 2, появившейся в апреле 1988 года, компания IBM реализовала поддержку первичных ключей. После этого подобная поддержка была добавлена в стандарт ANSI/ISO.[3, 7, 8, 12].

### Индексы

***Индексы*** *– объекты базы данных, которые обеспечивают быстрый доступ к отдельным строкам в таблице.* Индекс создается с целью повышения производительности операций запросов и сортировки данных таблицы. Индексы также используются для поддержания в таблицах некоторых типов ключевых ограничений; эти индексы часто создаются автоматически при определении ограничения.

Индекс – независимый объект, логически отдельный от таблицы; создание или удаление индекса никак не воздействует на определение или данные индексированной таблицы. Он хранит высоко оптимизированные версии всех значений одного или больше столбцов таблицы. Когда значение запрашивается из индексированного столбца, процессор (ядро) базы данных использует индекс для быстрого нахождения требуемого значения. Индексы должны постоянно поддерживаться, чтобы отражать последние изменения индексированных столбцов таблицы. Процедуры обновления индекса при вставке, модификации или удалении значения в индексированный столбец автоматически выполняются процессором базы данных. Хотя эти операции не требуют никаких действий со стороны пользователя, они, однако, снижают эффективность некоторых операций манипулирования данными (кроме запросов на выборку). Однако уменьшение производительности, ассоциированное с поддержанием индекса, в большинстве случаев с лихвой компенсируется преимуществами повышения быстродействия доступа к данным, которое обеспечивает индекс. Индексы обеспечивают наибольшие выгоды для относительно статичных таблиц, по которым часто выполняются запросы.

Создать индексы, как и ключи, можно по одному или нескольким полям. Составные индексы позволяют при отборе данных группировать записи, в которых первые поля могут иметь одинаковые значения. Индексировать поля требуется для выполнения частых поисков, сортировок или объединений с полями из других таблиц в запросах. Ключевые поля таблицы индексируются автоматически. Нельзя индексировать поля с типом данных поле МЕМО, гиперссылка или объект OLE. Для остальных полей индексирование используется, если поле имеет текстовый, числовой, денежный тип или тип даты/времени и требуется осуществлять поиск и сортировку значений в поле. Если предполагается, что будет часто выполняться сортировка или поиск одновременно по двум и более полям, можно создать составной индекс. Например, если для одного и того же запроса часто устанавливается критерий для полей Имя и Фамилия, то для этих двух полей имеет смысл создать составной индекс. При сортировке таблицы по составному индексу сначала осуществляется сортировка по первому полю, определенному для данного индекса. Если в первом поле содержатся записи с повторяющимися значениями, то сортировка осуществляется по второму полю и т. д.[3, 7, 8, 12].

### Отношения предок/потомок

Одним из отличий реляционной модели от первых моделей представления данных было то, что в ней отсутствовали явные указатели, используемые для реализации отношений предок/потомок в иерархической модели данных. Однако вполне очевидно, что отношения предок/потомок существуют и в реляционных базах данных. Например, в нашей базе данных каждой оценке на экзамене соответствует дисциплина, поэтому ясно, что между строками таблицы DISCIPLS и таблицы EXAMINE существует отношение.

Как следует из рис.6, это никоим образом не приводит к потере информации. На рисунке изображено несколько строк из таблиц DISCIPLS и EXAMINE. Обратим внимание на то, что в столбце NUMBER таблицы EXAMINE содержится идентификатор студента. Доменом этого столбца (множеством значений, которые могут в нем храниться) является множество идентификаторов студентов, содержащихся в столбце NUMBER таблицы EXAMINE.

Отношение предок/потомок, существующее между дисциплинами и оценками за экзамен, в реляционной модели не потеряно; просто оно реализовано в виде одинаковых значений данных, хранящихся в двух таблицах, а не в виде явного указателя. Все отношения, существующие между таблицами реляционной базы данных, реализуются в таком виде.

### Внешние ключи

|  |  |  |
| --- | --- | --- |
| **Таблица** DISCIPLS | |  |
| NDIS | NAME | TYPEDIS |
| 22 | МАТ.АНАЛИЗ | Общеобразоват |
| 11 | ФИЗИКА | Естеств |
| 12 | ИН.ЯЗЫК | ГУМАНИТАР |

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица** EXAMINE | |  |  |
| NUMBER | NUM\_DIS | SEMESTR | RESULT |
| 105 | 22 | 1 | 3 |
| 109 | 11 | 1 | 3 |
| 102 | 11 | 2 | 4 |
| 106 | 12 | 5 | 5 |

***Рис. 6. Отношение предок/потомок в реляционной базе данных***

Столбец одной таблицы, значения в котором совпадают со значениями столбца, являющегося первичным ключом другой таблицы, называется ***внешним ключом****.* На рис. 7 столбец NUM\_DIS представляет собой внешний ключ для таблицы DISCIPLS. Значения, содержащиеся в этом столбце, представляют собой идентификаторы изучаемых дисциплин. Эти значения соответствуют значениям в столбце NDIS, который является первичным ключом таблицы DISCIPLS. Совокупно первичный и внешний ключи создают между таблицами, в которых они содержатся, такое же отношение предок/потомок, как и в иерархической базе данных.

Внешний ключ, как и первичный ключ, тоже может представлять собой комбинацию столбцов. На практике внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей. На рис. 7. показаны три внешних ключа таблицы ORDERS из учебной базы данных:

столбец REP является внешним ключом для таблицы SALESREPS и  
связывает каждый заказ со служащим, принявшим его;

столбец CUST является внешним ключом для таблицы CUSTOMES и  
связывает каждый заказ с клиентом, разместившим его;

столбцы MRF и PRODUCT совокупно представляют собой составной внешний ключ для таблицы PRODUCTS, который связывает каждый заказ с заказанным товаром.

|  |  |
| --- | --- |
| Таблица CUSTOMERS | |
| CUST\_NUM | COMPANY |
| … |  |
| 2117 | J.P. Sinclair |
| … |  |

|  |  |
| --- | --- |
| Таблица SALESREPS | |
| EMPL\_NUM | NAME |
| … |  |
| 106 | Sam Clark |
| … |  |

|  |  |  |
| --- | --- | --- |
| Таблица PRODUCTS | |  |
| MFR\_ID | PRODUCT\_ID | DESCRIPTION |
| … |  |  |
| REI | 2A44L | Left Hinge |
| … |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица ORDERS | |  |  |  |  |
| ORDER\_NUM | ORDER\_DATE | CUST | REP | MFR | PRODUCT |
| … |  |  |  |  |  |
| 112967 | 17-DEC-89 | 2117 | 106 | REI | 2A44L |
| … |  |  |  |  |  |

*Рис. 7. Множественные отношения предок/потомок в реляционной базе данных*

Реляционная модель данных обладает всеми возможностями сетевой модели по части выражения сложных отношений.

Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют отношения между таблицами базы данных. К несчастью, как и в случае с первичными ключами, поддержка внешних ключей отсутствовала в первых реляционных СУБД. Она была введена в системе DB2 Version 2 и теперь имеется во всех коммерческих СУБД.

### Реляционная алгебра

Основная идея реляционной алгебры состоит в том, что коль скоро таблицы являются множествами, то средства манипулирования ими могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для баз данных.

Используется немного расширенный начальный вариант алгебры, который был предложен Коддом. В этом варианте набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса - теоретико-множественные операции и специальные реляционные операции. В состав ***теоретико-множественных*** операций входят операции:

* объединения таблиц;
* пересечения таблиц;
* взятия разности таблиц;
* прямого произведения таблиц.

***Специальные*** реляционные операции включают:

* ограничение таблицы;
* проекцию таблицы;
* соединение таблиц;
* деление таблиц.

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующей таблицы.

*Ограничения целостности.*

Существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление записи, на которую существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся записи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении записи, на которую имеются ссылки, во всех ссылающихся записях значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении записи из таблицы, на которую ведет ссылка, из ссылающейся таблицы автоматически удаляются все ссылающиеся записи.[7, 12, 13].

### Нормализация базы данных

Процесс трансформации данных в реляционную форму называется ***нормализацией***[9]. Говоря проще, нормализация - это удаление избыточных данных из каждой таблицы в базе данных. У нормализации двойная цель - удалить лишние копии данных и обеспечить максималь­ную гибкость как в структурах таблиц, так и в интерфейсных приложениях на случай возможных будущих изменений в базах данных.

О нормализации таблиц в базе данных нужно заботится на раннем этапе проектирования приложения, так как при “живых” данных довольно трудно менять структуру базы. Иногда процесс нормализации порождает добавочные таблицы, которые были не включены в первона­чальный проект. Узнав об этом как можно раньше, не придется зря тратить силы на их разработку.

Нормализация обычно подразделяется на пять форм или стадий— от *первой нормальной формы* по *пятую нормальную форму.* То есть просто пять установок реляционного критерия, который либо обнаруживает таблицу, либо нет. Каждая последующая стадия строится на предыдущей. Формально существует пять форм, но на практике, как правило, используется только первые три. Последние две считаются слишком специальными, чтобы их применять к обычным проектам баз данных.[7, 8, 10].

#### Первая нормальная форма

Для того чтобы таблица считалась нормализованной к первой нормальной форме, каждое из ее полей должно быть неделимым и не должно содержать никаких повторяющихся групп.

Поле считается неделимым, если оно содержит только один элемент данных. Например, поле Address, которое содержит не только название улицы, но также и города, почтовый код, не является неделимым. Чтобы соответствовать первой нормальной форме, такие столбцы должны быть разбиты на несколько полей.[7, 8, 10].

***Повторяющаяся группа*** *— это поле, которое повторяется внутри определения записи с целью хранения нескольких значений для атрибута.*

#### Вторая нормальная форма

Для того чтобы привести таблицу ко второй нормальной форме, нужно, чтобы все не ключевые поля полностью зависели от первичного ключа таблицы и от каждого поля в первичном ключе, если последний состоит из нескольких полей. Это зна­чит, что каждое не ключевое поле должно уникально определяться первичным ключом и полями, его составляющими.[7, 8, 10].

#### Третья нормальная форма

Для того чтобы таблица была приведена к третьей нормальной форме, нужно, чтобы все не ключевые поля полностью зависели от первичного ключа таблицы и не зависели друг от друга. Таким образом, к квалификации второй нормальной формы добавляется требование независимости каждого не ключевого поля таблицы от других не ключевых полей.[7, 8, 10].

#### Четвертая нормальная форма

Четвертая нормальная форма запрещает хранить независимые элементы в одной и той же таблице, когда между этими элементами существуют взаимоотношения типа многие-ко-многим. Четвертая нормальная форма требует, чтобы запомнили такие элементы в отдельных таблицах и создали таблицу от­ношений для организации связей между таблицами, характеризующихся взаимоотношениями типа многие-ко-многим.

Конечно же, поскольку два столбца находятся во взаимоотношении многие-ко-многим, то они уже не являются неза­висимыми, и тем самым уже нарушают третью нормальную форму. По этой причине четвертая нормальная форма рас­сматривается больше теоретически, т.к. частично она перекрывается третьей нормальной формой.[7, 8, 10].

#### Пятая нормальная форма

Пятая нормальная форма требует, чтобы вы имели возможность перестраивать свои данные в нормализованных таб­лицах, в которые они были переведены. Это значит, что если вы начинаете с ненормализованных таблиц, то у вас не должно быть препятствий к вырезке и вставке данных и после нормализации. Это осуществимые, если есть гарантия, что в процессе нормализации не будет потери данных.

На практике идея сохранения всех элементов в базе данных в процессе нормализации воплощается чисто интуитивно. Ведь вряд ли будут слепо выбрасывать из таблиц элементы данных. Но тем не менее, пятая нормальная форма при­звана застраховать вас от такого несчастного случая.[7, 8, 10].

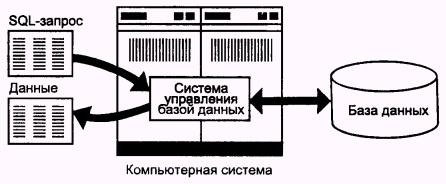
# Язык SQL как стандартный язык баз данных.

Стремительный рост популярности SQL является одной из самых важных тенденций в современной компьютерной промышленности. За несколько последних лет SQL стал *единственным* языком баз данных. На сегодняшний день SQL поддерживают свыше ста СУБД, работающих как на персональных компьютерах, так и на больших ЭВМ. Был принят, а затем дополнен официальный международный стандарт на SQL. Язык SQL является важным звеном в архитектуре систем управления базами данных, выпускаемых всеми ведущими поставщиками программных продуктов, и служит стратегическим направлением разработок компании Microsoft в области баз данных. Зародившись в результате выполнения второстепенного исследовательского проекта компании IBM, SQL сегодня широко известен и в качестве мощного рыночного фактора.[13]

## Язык SQL

***QL*** является инструментом, предназначенным для обработки и чтения данных, содержащихся в компьютерной базе данных. SQL - это сокращенное название ***структурированного языка запросов*** *(Structured Query Language*). Как следует из названия, SQL является *языком программирования*, который применяется для организации взаимодействия пользователя с базой данных. На самом деле SQL работает только с базами данных реляционноготипа*.* На рис. 8 изображена схема работы SQL. Согласно этой схеме, в вычислительной системе имеется *база данных,* в которой хранится важная информация. Если вычислительная система относится к сфере бизнеса, то в базе данных может храниться информация о материальных ценностях, выпускаемой продукции, объемах продаж и зарплате. В базе данных на персональном компьютере может храниться информация о выписанных чеках, телефонах и адресах или информация, извлеченная из более крупной вычислительной системы. Компьютерная программа, которая управляет базой данных, называется ***системой управления базой данных****,* или ***СУБД***.

*Рис. 8. Применение SQL для доступа к базе данных*



Если пользователю необходимо прочитать данные из базы данных, он запрашивает их у СУБД с помощью SQL. СУБД обрабатывает запрос, находит требуемые данные и посылает их пользователю. Процесс запрашивания данных и получения результата называется ***запросом*** к базе данных: отсюда и название — ***структурированный язык запросов****.*

Однако это название не совсем соответствует действительности. Во-первых, сегодня SQL представляет собой нечто гораздо большее, чем простой инструмент создания запросов, хотя именно для этого он и был первоначально предназначен. Несмотря на то, что чтение данных по-прежнему остается одной из наиболее важных функций SQL, сейчас этот язык используется для реализации всех функциональных возможностей, которые СУБД предоставляет пользователю, а именно:

* ***Организация данных****.* SQL дает пользователю возможность изменять структуру представления данных, а также устанавливать отношения между элементами базы данных.
* ***Чтение данных****.* SQL дает пользователю или приложению возможность читать из базы данных содержащиеся в ней данные и пользоваться ими.
* ***Обработка ванных****.* SQL дает пользователю или приложению возможность изменять базу данных, т.е. добавлять в нее новые данные, а также удалять или обновлять уже имеющиеся в ней данные.
* ***Управление доступом****.* С помощью SQL можно ограничить возможности пользователя по чтению и изменению данных и защитить их от несанкционированного доступа.
* ***Совместное использование данных****.* SQL координирует совместное использование данных пользователями, работающими параллельно, чтобы они не мешали друг другу.
* ***целостность данных****.* SQL позволяет обеспечить целостность базы данных, защищая ее от разрушения из-за несогласованных изменений или отказа системы.

Таким образом, SQL является достаточно мощным языком для взаимодействия с СУБД.

Во-вторых, SQL — это не полноценный компьютерный язык типа COBOL, FORTRAN или С. В SQL нет оператора IF для проверки условий, нет оператора GOTO для организации переходов и нет операторов DO или FOR для создания циклов. SQL является ***подъязыком*** баз данных, в который входит около тридцати операторов, предназначенных для управления базами данных. Операторы SQL встраиваются в базовый язык, например COBOL, FORTRAN или С, и дают возможность получать доступ к базам данных. Кроме того, из такого языка, как С, операторы SQL можно посылать СУБД в явном виде, используя интерфейс вызовов функций*.*

Наконец, SQL — это слабо структурированный язык, особенно по сравнению с такими сильно структурированными языками, как С или Pascal. Операторы SQL напоминают английские предложения и содержат "слова-пустышки", не влияющие на смысл оператора, но облегчающие его чтение. В SQL почти нет нелогичностей, к тому же имеется ряд специальных правил, предотвращающих создание операторов SQL, которые выглядят как абсолютно правильные, но не имеют смысла.

Несмотря на не совсем точное название, SQL на сегодняшний день является единственным *стандартным* языком для работы с реляционными базами данных. SQL — это достаточно мощный и в то же время относительно легкий для изучения язык.[13, 8].

## Достоинства SQL

SQL — это легкий для понимания язык и в то же время универсальное программное средство управления данными.

Успех языку SQL принесли следующие его особенности:

• независимость от конкретных СУБД;

• переносимость с одной вычислительной системы на другую;

• наличие стандартов;

• одобрение компанией IBM (СУБД DB2);

• поддержка со стороны компании Microsoft (протокол ODBC);

• реляционная основа;

• высокоуровневая структура, напоминающая английский язык;

• возможность выполнения специальных интерактивных запросов:

• обеспечение программного доступа к базам данных;

• возможность различного представления данных;

• полноценность как языка, предназначенного для работы с базами данных;

• возможность динамического определения данных;

• поддержка архитектуры клиент/сервер.

Все перечисленные выше факторы явились причиной того, что SQL стал стандартным инструментом для управления данными на персональных компьютерах, мини-компьютерах и больших ЭВМ. Ниже эти факторы рассмотрены более подробно.[13, 8, 17].

### Независимость от конкретных СУБД

Все ведущие поставщики СУБД используют SQL, и ни одна новая СУБД, не поддерживающая SQL, не может рассчитывать на успех. Реляционную базу данных и программы, которые с ней работают, можно перенести с одной СУБД на другую с минимальными доработками и переподготовкой персонала. Программные средства, входящие в состав СУБД для персональных компьютеров, такие как программы для создания запросов, генераторы отчетов и генераторы приложений, работают с реляционными базами данных многих типов. Таким образом, SQL обеспечивает независимость от конкретных СУБД, что является одной из наиболее важных причин его популярности.

### Переносимость с одной вычислительной системы на другие

Поставщики СУБД предлагают программные продукты для различных вычислительных систем: от персональных компьютеров и рабочих станций до локальных сетей, мини-компьютеров и больших ЭВМ. Приложения, созданные с помощью SQL и рассчитанные на однопользовательские системы, по мере своего развития могут быть перенесены в более крупные системы. Информация из корпоративных реляционных баз данных может быть загружена в базы данных отдельных подразделений или в личные базы данных. Наконец, приложения для реляционных баз данных можно вначале смоделировать на экономичных персональных компьютерах, а затем перенести на дорогие многопользовательские системы.

### Стандарты языка SQL

Официальный стандарт языка SQL был опубликован Американским институтом национальных стандартов (American National Standards Institute — ANSI) и Международной организацией по стандартам (International Standards Organization — ISO) в 1986 году и значительно расширен в 1992 году. Кроме того, SQL является федеральным стандартом США по обработке информации (FIPS — Federal Information Processing Standard) и, следовательно, соответствие ему является одним из основных требований, содержащихся в больших правительственных контрактах, относящихся к области вычислительной техники. В Европе стандарт X/OPEN для переносимой среды программирования на основе операционной системы UNIX включает в себя SQL в качестве стандарта для доступа к базам данных. SQL Access Group — консорциум поставщиков компьютерного оборудования и баз данных — определил для SQL стандартный интерфейс вызовов функций, который является основой протокола ODBC компании Microsoft и входит также в стандарт X/OPEN. Эти стандарты служат как бы официальной печатью, одобряющей SQL, и они ускорили завоевание им рынка.[13, 8, 17].

### Одобрение SQL компанией IBM (СУБД DB2)

SQL был придуман научными сотрудниками компании IBM и широко используется ею во множестве пакетов программного обеспечения. Подтверждением этому служит флагманская СУБД DB2 компании IBM. Все основные семейства компьютеров компании IBM поддерживают SQL: система PS/2 для персональных компьютеров, система среднего уровня AS/400. система RS/6000 на базе UNIX, а также операционные системы MVS и VM больших ЭВМ. Широкая поддержка SQL фирмой IBM ускорила его признание и еще в самом начале возникновения и развития рынка баз данных явилась своего рода недвусмысленным указанием для других поставщиков баз данных и программных систем, в каком направлении необходимо двигаться.

### Протокол ODBC и компания Microsoft

Компания Microsoft рассматривает доступ к базам данных как важную часть своей операционной системы Windows. Стандартом этой компании по обеспечению доступа к базам данных является ODBC (Open Database Connectivity — взаимодействие с открытыми базами данных) — программный интерфейс, основанный на SQL. Протокол ODBC поддерживается наиболее распространенными приложениями Windows (электронными таблицами, текстовыми процессорами, базами данных и т.п.), разработанными как самой компанией Microsoft, так и другими ведущими поставщиками. Поддержка ODBC обеспечивается всеми ведущими реляционными базами данных. Кроме того, ODBC опирается на стандарты, одобренные консорциумом поставщиков SQL Access Group, что делает ODBC как стандартом де-факто компании Microsoft, так и стандартом, независимым от конкретных СУБД.[13, 8, 17].

### Реляционная основа

SQL является языком реляционных баз данных, поэтому он стал популярным тогда, когда популярной стала реляционная модель представления данных. Табличная структура реляционной базы данных интуитивно понятна пользователям, поэтому язык SQL является простым и легким для изучения. Реляционная модель имеет солидный теоретический фундамент, на котором были основаны эволюция и реализация реляционных баз данных. На волне популярности, вызванной успехом реляционной модели, SQL стал *единственным* языком для реляционных баз данных.[13, 8, 17].

### Высокоуровневая структура, напоминающая английский язык

Операторы SQL выглядят как обычные английские предложения, что упрощает их изучение и понимание. Частично это обусловлено тем, что операторы SQL *описывают* данные, которые необходимо получить, а не *определяют способ* их поиска. Таблицы и столбцы в реляционной базе данных могут иметь длинные описательные имена. В результате большинство операторов SQL означают именно то, что точно соответствует их именам, поэтому их можно читать как простые, понятные предложения.

### Интерактивные запросы

SQL является языком интерактивных запросов, который обеспечивает пользователям немедленный доступ к данным. С помощью SQL пользователь может в интерактивном режиме получить ответы на самые сложные запросы в считанные минуты или секунды, тогда как программисту потребовались бы дни или недели, чтобы написать для пользователя соответствующую программу. Из-за того, что SQL допускает немедленные запросы, данные становятся более доступными и могут помочь в принятии решений, делая их более обоснованными.[13, 8, 17].

### Программный доступ к базе данных

Программисты пользуются языком SQL, чтобы писать приложения, в которых содержатся обращения к базам данных. Одни и те же операторы SQL используются как для интерактивного, так и для программного доступа, поэтому части программ, содержащие обращения к базе данных, можно вначале тестировать в интерактивном режиме, а затем встраивать в программу. В традиционных базах данных для программного доступа используются одни программные средства, а для выполнения немедленных запросов — другие, без какой либо связи между этими двумя режимами доступа.[13, 8, 17].

### Различные представления данных

С помощью SQL создатель базы может сделать так, что различные пользователи базы данных будут видеть *различные* представления её структуры и содержимого. Например, базу данных можно спроектировать таким образом, что каждый пользователь будет видеть только данные, относящиеся к его подразделению или торговому региону. Кроме того, данные из различных частей базы данных могут быть скомбинированы и представлены пользователю в виде одной простой таблицы. Следовательно, представления можно использовать для усиления защиты базы данных и ее настройки под конкретные требования отдельных пользователей.[13, 8, 17].

### Полноценный язык для работы с базами данных

Первоначально SQL был задуман как язык интерактивных запросов, но сейчас он вышел далеко за рамки чтения данных. SQL является полноценным и логичным языком, предназначенным для создания базы данных, управления ее защитой, изменения ее содержимого, чтения данных и совместного использования данных несколькими пользователями, работающими параллельно. Приемы, освоенные при изучении одного раздела языка, могут затем применяться в других командах, что повышает производительность работы пользователей.[13, 8, 17].

### Динамическое определение данных

С помощью SQL можно динамически изменять и расширять структуру базы данных даже в то время, когда пользователи обращаются к ее содержимому. Это большое преимущество перед языками статического определения данных, которые запрещают доступ к базе данных во время изменения ее структуры. Таким образом, SQL обеспечивает максимальную гибкость, так как дает базе данных возможность адаптироваться к изменяющимся требованиям, не прерывая работу приложения, выполняющегося в реальном масштабе времени.[13, 8, 17].

### Архитектура клиент/сервер

SQL — естественное средство для реализации приложений клиент/сервер. В этой роли SQL служит связующим звеном между клиентской системой, взаимодействующей с пользователем, и серверной системой, управляющей базой данных, позволяя каждой системе сосредоточиться на выполнении своих функций. Кроме того, SQL позволяет персональным компьютерам функционировать в качестве клиентов по отношению к сетевым серверам или более крупным базам данных, установленным на больших ЭВМ; это позволяет получать доступ к корпоративным данным из приложений, работающих на персональных компьютерах.[13, 8, 17].

# Архитектуры баз данных

Для рассмотрения способов организации баз данных нужно определить несколько понятий.

***Ядро БД*** отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций. Ядро БД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL и утилитах БД. Ядро БД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией ***компилятора языка БД*** является компиляция операторов языка БД в некоторую выполняемую программу.

В отдельные ***утилиты БД*** обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра БД, а иногда даже с проникновением внутрь ядра.

# База данных

# Ядро БД

Приложение для работы с БД

**Компьютер пользователя**

Рис. 11. Однопользовательская архитектура при работе с локальными БД

**Приложение**

**Невизуальные компоненты для работы с БД**

**Визуальные компоненты для работы с БД**

# База данных

# Ядро БД

Рис. 10. Общий состав средств для работы готового приложения с БД.

Общий состав средств, необходимых для работы готового приложения с БД, показан на рис. 10. Согласно этой общей схеме, мы имеем цепочку

Приложение —**>** Ядро БД —**>** базы данных. В структуре приложения имеется цепочка Невизуальные компоненты —**>** Визуальные компоненты. ***Невизуальные компоненты*** предоставляют программисту некоторые функции по управлению ядром базы данных, а также самими данными. С помощью ***Визуальных компонент*** данные отображаются на экране (таблицы, списки, выпадающие списки, графики и др.). Местоположение ядра БД и самих баз данных в этой цепочке не отражены.

Местоположение Ядра БД и баз данных зависит от используемой архитектуры. Имеется четыре разновидности архитектур баз данных:

• локальные базы данных; ' архитектура "файл-сервер";

• архитектура "клиент-сервер";

• многозвенная (трехзвенная N-tier или multi-tier) архитектура.

Использование той или иной архитектуры накладывает сильный отпечаток на общую идеологию работы приложения, на программный код в приложении, на состав компонентов для работы с БД, используемых в приложении (прежде всего это касается невизуальных компонентов).[4, 15].

## Локальные базы данных и архитектура "файл-сервер"

При работе с ***локальными базами данных*** сами БД расположены на том же компьютере, что и приложения, осуществляющие доступ к ним. Работа с БД происходит в однопользовательском режиме. Ядро БД распложено на компьютере пользователя. Приложение ответственно за поддержание целостности БД и за выполнение запросов к БД. Общая схема однопользовательской архитектуры показана на рис. 11.

При работе в ***архитектуре "файл-сервер"*** БД и приложение расположены на файловом сервере сети (например, Novell NetWare). Возможна много­пользовательская работа с одной и той же БД, когда каждый пользователь со своего компьютера запускает приложение, расположенное на сетевом сервере. Тогда на компьютере пользователя запускается копия приложения. По каждому запросу к БД из приложения данные из таблиц БД перегоняются на компьютер пользователя, независимо от того, сколько реально нужно данных для выполнения запроса. После этого выполняется запрос.

Приложение для работы с БД

**Компьютер пользователя**

# Ядро БД

# Копия базы данных

# База данных

**Сетевой сервер**

Рис. 12. Архитектура “Файл-Сервер”

# Приложение для работы с БД (Клиент)

# Ядро БД

**Компьютер пользователя**

**База данных**

# Сервер базы данных

**Удаленный сервер**

Рис. 13. Архитектура “Клиент-Сервер”

Каждый пользователь имеет на своем компьютере локальную копию данных, время от времени обновляемых из реальной БД, расположенной на сетевом сервере. При этом изменения, которые каждый пользователь вносит в БД, могут быть до определенного момента неизвестны другим пользователям, что делает актуальной задачу систематического обновления данных на компьютере пользователя из реальной БД. Другой актуальной задачей является блокирование записей, которые изменяются одним из пользователей: это необходимо для того, чтобы в это время другой пользователь не внес изменений в те же данные. В архитектуре "файл-сервер" вся тяжесть выполнения запросов к БД р, управления целостностью БД ложится на приложение пользователя. БД да сервере является пассивным источником данных. Общая схема архитектуру "файл-сервер" показана на рис. 12.

Кардинальных различий с точки зрения архитектуры между одно­пользовательской архитектурой и архитектурой "файл-сервер" нет. И в том. и в ином случае в качестве СУБД применяются так называемые "персональные" (или "локальные") СУБД. такие как Paradox, dBase и пр. Сама база данных в этом случае представляет собой набор таблиц, индексных файлов, файлов полей комментариев (мемо-полей) и пр., хранящихся в одном каталоге на диске в виде отдельных файлов.[4].

## Удаленные базы данных и архитектура "клиент-сервер"

Архитектура "файл-сервер" неэффективна, по крайней мере, в двух отношениях:

1. При выполнении запроса к базе данных, расположенной на файловом сервере, в действительности происходит запрос к локальной копии данных на компьютере пользователя. Поэтому перед выполнением запроса данные в локальной копии обновляются из реальной БД. Данные обновляются в полном объеме. Так, если таблица БД состоит из 1000 записей, а для выполнения запроса (например, выдать сумму премий за октябрь в отделе Y) реально нужно 10 записей, все равно перегоняются все 1000 записей. Таким образом, не нужно иметь слишком много пользователей и запросов от них, чтобы серьезно ''забить" сеть, что, конечно же, не может не сказаться на ее быстродействии.
2. Обеспечение целостности БД производится из приложений. Это потенциальный источник ошибок, нарушающих физическую и логическую целостность БД, поскольку различные приложения могут производить контроль целостности БД по-разному, взаимоисключающими способами, или не проводить такого контроля вовсе. Намного эффективнее управлять БД из единого места и по единым законам, нежели из разных приложений и по потенциально разным законам (все зависит от того, как написано приложение). Поэтому безопасность при работе в архитектуре "файл-сервер" невысока и всегда присутствует элемент неопределенности. Секретность и конфиденциальность при работе с БД в архитектуре "файл-сервер" обеспечить также тяжело - любой, кто имеет доступ в каталог сетевого сервера, где хранится БД, может изменять таблицы БД любым образом, копировать их, заменять и т.д. [4].

Архитектура ***"клиент-сервер"*** разделяет функции приложения пользователя (называемого клиентом) и сервера (Рис. 13.).

Приложение-клиент формирует запрос к серверу, на котором расположена БД, на структурном языке запросов SQL. Удаленный сервер принимает запрос и переадресует его SQL-серверу БД. SQL-сервер – это специальная программа, управляющая удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных. формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети "путешествуют" только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Поскольку выполнение запроса происходит там же, где хранятся данные (на сервере), нет необходимости в пересылке больших пакетов данных. Кроме того, SQL-сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами.

Все это повышает быстродействие системы и снижает время ожидания результата запроса.

При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются в базе данных на сервере и являются едиными для всех приложений, исполь­зующих эту БД. Таким образом, исключается возможность определения противоречивых правил поддержания целостности. Мощный аппарат транзакций, поддерживаемый SQL-серверами, позволяет исключить одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно. Таким образом, функциями приложения-клиента являются:

1. посылка к серверу запросов;
2. интерпретация результатов запросов, полученных от сервера, и представление их пользователю в требуемой форме;
3. реализация интерфейса пользователя.

***SQL-сервер*** - это программа, расположенная на компьютере сетевого сервера. SQL-сервер должен быть загружен на момент принятия запроса от клиента. Функциями сервера БД являются:

1. прием запросов от приложений-клиентов, интерпретация запросов, выполнение запросов в БД, отправка результата выполнения запроса приложению-клиенту;
2. управление целостностью БД, обеспечение системы безопасности, блокировка неверных действий приложений-клиентов;
3. хранение *бизнес-правил*, часто используемых запросов в уже интерпретированном виде;
4. обеспечение одновременно безопасной и отказоустойчивой многопользовательской работы с одними и теми же данными. В архитектуре "клиент-сервер" используются так называемые "удаленные" (или "промышленные") СУБД. Промышленными они называются из-за того. что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия, организации, банка. Локальные СУБД предназначены для однопользовательской работы или для обеспечения работы информационных систем, рассчитанных на небольшие группы пользователей.[4, 15, 11].

К разрядку промышленных СУБД принадлежат: Oracle, Gupta, Informix, Sybase, MS SQL Server, DB2, InterBase и ряд других.

Как правило, SQL-сервер управляется отдельным сотрудником или группой сотрудников (администраторы SQL-сервера). Они управляют физическими характеристиками баз данных, производят оптимизацию, настройку и переопределение различных компонентов БД, создают новые БД, изменяют существующие и т.д., а также выдают привилегии (разрешения на доступ определенного уровня к конкретным БД, SQL-серверу) различным пользователям.

Кроме этого, существует отдельная категория сотрудников, называемых ***администраторами*** баз данных. Как правило, это администраторы сервера, разработчики БД или пользователи, имеющие привилегии на создание, изменение, настройку оптимальных параметров отдельных серверных БД-Администраторы БД также отвечают за предоставление прав на разно­уровневый доступ к сопровождаемым ими БД для других пользователей.[4, 15, 11].

Использование архитектуры "клиент-сервер":

1. резко уменьшает сетевой трафик:
2. понижает сложность приложений-клиентов (поскольку тем уже нет необходимости обеспечивать целостность и безопасность БД и следить за параметрами многопользовательской работы с БД);
3. понижает требования к аппаратным средствам, на которых эти приложения функционируют (т.е. к компьютерам пользователей-клиентов):
4. повышает надежность БД, ее целостность, безопасность и секретность.

# Среда Delphi как средство для разработки СУБД

Поскольку использование баз данных является одним из краеугольных камней, на которых построено существование различных организаций, пристальное внимание разработчиков приложений баз данных вызывают инструменты, при помощи которых такие приложения можно было бы создавать. Выдвигаемые к ним требования в общем виде можно сформулировать как: "быстрота, простота, эффективность, надежность".

Среди большого разнообразия продуктов для разработки приложений Delphi занимает одно из ведущих мест. Delphi отдают предпочтение разработчики с разным стажем, привычками, профессиональными интересами. С помощью Delphi написано колоссальное количество приложений, десятки фирм и тысячи программистов-одиночек разрабатывают для Delphi дополнительные компоненты.[4].

В основе такой общепризнанной популярности лежит тот факт, что Delphi, как никакая другая система программирования, удовлетворяет изложенным выше требованиям. Действительно, приложения с помощью Delphi разрабатываются быстро, причем взаимодействие разработчика с интерактивной средой Delphi не вызывает внутреннего отторжения, а наоборот, оставляет ощущение комфорта. Delphi-приложения эффективны, если разработчик соблюдает определенные правила (и часто - если не соблюдает). Эти приложения надежны и при эксплуатации обладают предсказуемым поведением.[4, 22].

Пакет Delphi - продолжение линии компиляторов языка Pascal корпорации Borland. Pascal как язык очень прост, а строгий контроль типов данных способствует раннему обнаружению ошибок и позволяет быстро создавать надежные и эффективные программы. Корпорация Borland постоянно обогащала язык. Когда-то в версию 4.0 были включены средства раздельной трансляции, позже, начиная с версии 5.5, появились объекты, а в состав шестой версии пакета вошла полноценная библиотека классов Turbo Vision, реализующая оконную систему в текстовом режиме работы видеоадаптера. Это был один из первых продуктов, содержавших интегрированную среду разработки программ.

В классе инструментальных средств для начинающих программистов продуктам компании Borland пришлось конкурировать со средой Visual Basic корпорации Microsoft, где вопросы интеграции и удобства работы были решены лучше. Когда в начале 70-х годов Н. Вирт опубликовал сообщение о Pascal, это был компактный, с небольшим количеством основных понятий и зарезервированных слов язык программирования, нацеленный на обучение студентов. Язык, на котором предстоит работать пользователю Delphi, отличается от исходного не только наличием множества новых понятий и конструкций, но и идейно: в нем вместо минимизации числа понятий и использования самых простых конструкций (что, безусловно, хорошо для обучения, но не всегда оправдано в практической работе), предпочтение отдается удобству работы профессионального пользователя. Как язык Turbo Pascal естественно сравнивать с его ближайшими конкурентами - многочисленными вариациями на тему языка Basic (в первую очередь с Visual Basic корпорации Microsoft) и с C++.[4, 6]. Я считаю, что Turbo Pascal существенно превосходит Basic за счет полноценного объектного подхода, включающего в себя развитые механизмы *инкапсуляции, наследование и полиморфизм*. Последняя версия языка, применяемая в Delphi, по своим возможностям приближается к C++. Из основных механизмов, присущих C++, отсутствует только множественное наследование. (Впрочем, этим красивым и мощным механизмом порождения новых классов пользуется лишь небольшая часть программистов, пишущих на С++.) Плюсы применения языка Pascal очевидны: с одной стороны, в отличие от Visual Basic, основанного на интерпретации промежуточного кода, для него имеется компилятор, генерирующий машинный код, что позволяет получать значительно более быстрые программы. С другой - в отличие от C++ синтаксис языка Pascal способствует построению очень быстрых компиляторов. [6].

Среда программирования напоминает пакет Visual Basic. В вашем распоряжении несколько отдельных окон: меню и инструментальные панели, Object Inspector (в котором можно видеть свойства объекта и связанные с ним события), окна визуального построителя интерфейсов (Visual User Interface Builder), Object Browser (позволяющее изучать иерархию классов и просматривать списки их полей, методов и свойств), окна управления проектом (Project Manager) и редактора.

Delphi содержит полноценный текстовый редактор типа Brief, назначения клавиш в котором соответствуют принятым в Windows стандартам, а глубина иерархии операций Undo неограниченна. Как это стало уже обязательным, реализовано цветовое выделение различных лексических элементов программы. Процесс построения приложения достаточно прост. Нужно выбрать форму (в понятие формы входят обычные, диалоговые, родительские и дочерние окна MDI), задать ее свойства и включить в нее необходимые компоненты (видимые и, если понадобится, неотображаемые): меню, инструментальные панели, строку состояния и т. п., задать их свойства и далее написать (с помощью редактора исходного кода) обработчики событий. Object Browser Окна типа Object Browser стали неотъемлемой частью систем программирования на объектно-ориентированных языках. Работа с ними становится возможной сразу после того, как вы скомпилировали приложение.

Projeсt Manager - это отдельное окно, где перечисляются модули и формы, составляющие проект. При каждом модуле указывается маршрут к каталогу, в котором находится исходный текст. Жирным шрифтом выделяются измененные, но еще не сохраненные части проекта. В верхней части окна имеется набор кнопок: добавить, удалить, показать исходный текст, показать форму, задать опции и синхронизировать содержимое окна с текстом файла проекта, т. е. с головной программой на языке Pascal.

Опции, включая режимы компиляции, задаются для всего проекта в целом. В этом отношении традиционные make-файлы, используемые в компиляторах языка C, значительно более гибки.

Visual Component Library (VCL) Богатство палитры объектов для построения пользовательского интерфейса - один из ключевых факторов при выборе инструмента визуального программирования. При этом для пользователя имеет значение как число элементов, включенных непосредственно в среду, так и доступность элементов соответствующего формата на рынке. [4, 22].

## Высокопроизводительный компилятор в машинный код

Компиляторы языка Pascal компании Borland никогда не заставляли пользователя подолгу ждать результатов компиляции. Производители утверждают, что на сегодня данный компилятор - самый быстрый в мире. Компилятор, встроенный в Delphiпозволяет обрабатывать 120 тыс. строк исходного текста в минуту на машине 486/33 или 350 тыс. - при использовании процессора Pentium/90. Он предлагает легкость разработки и быстрое время проверки готового программного блока, характерного для языков четвертого поколения (4GL) и в то же время обеспечивает качество кода, характерного для компилятора 3GL. Кроме того, Delphi обеспечивает быструю разработку без необходимости писать вставки на Си или ручного написания кода (хотя это возможно).

В смысле проектирования Delphi мало чем отличается от проектирования в интерпретирующей среде, однако после выполнения компиляции мы получаем код, который исполняется в 10-20 раз быстрее, чем тоже самое, сделанное при помощи интерпретатора. Кроме того, компилятор компилятору рознь, в Delphi компиляция производится непосредственно в родной машинный код, в то время как существуют компиляторы, превращающие программу в так называемый p-код, который затем интерпретируется виртуальной p-машиной. Это не может не сказаться на фактическом быстродействии готового приложения.

По всей вероятности, такая высокая скорость объясняется в первую очередь отказом от демонстрации в процессе работы числа скомпилированных строк. Следует отметить также, что благодаря опции оптимизации сегментов удается существенно сократить размер выполняемого файла. Можно запустить компилятор в режиме проверки синтаксиса. При этом наиболее длительная операция компоновки и изготовления исполняемого файла выполняться не будет.

Вероятно, то обстоятельство, что Delphi позиционируется как средство создания приложений, взаимодействующих с базами данных, и ориентировано преимущественно на рынок инструментальных средств клиент/сервер, где до настоящего момента доминируют интерпретируемые языки, позволило его авторам не задумываться над созданием оптимизирующего компилятора, способного использовать все достоинства архитектур современных процессоров. [22].

## Мощный объектно-ориентированный язык

Совместимость с программами, созданными ранее средствами Borland Pascal, сохраняется, несмотря на то, что в язык внесены существенные изменения. Необходимость в некоторых усовершенствованиях давно ощущалась. Самое заметное из них - аппарат исключительных ситуаций, подобный тому, что имеется в C++, был первым реализован в компиляторах корпорации Borland. Не секрет, что при написании объектно-ориентированных программ, активно работающих с динамической памятью и другими ресурсами, немалую трудность представляет аккуратное освобождение этих ресурсов в случае возникновения нештатных ситуаций. Особенно это актуально для среды Windows, где число видов ресурсов довольно велико, а неряшливая работа с ними может быстро привести к зависанию всей системы. Предусмотренный в Delphi аппарат исключений максимально упрощает кодирование обработки нештатных ситуаций и освобождения ресурсов.

Объектно-ориентированный подход в новой версии языка получил значительное развитие. Перечислим основные новшества.

- введено понятие класса.

* реализованы методы классов, аналогичные статическим методам C++. Они оперируют не экземпляром класса, а самим классом.
* механизм ***инкапсуляции*** во многом усовершенствован. Введены защищенные поля и методы, которые, подобно приватным, не видны извне, но отличаются от них тем, что доступны из методов класса- наследника.
* введена обработка исключительных ситуаций. В **Delphi** это устроено в стиле С++. Исключения представлены в виде объектов, содержащих специфическую информацию о соответствующей ошибке (тип и место- нахождение ошибки). Разработчик может оставить обработку ошибки, существовавшую по умолчанию, или написать свой собственный обработчик. Обработка исключений реализована в виде *exception-handling blocks* (также еще называется *protected blocks*), которые устанавливаются ключевыми словами try и end. Существуют два типа таких блоков:*try...except* и *try...finally***.**
* появилось несколько удобных синтаксических конструкций, в числе которых преобразование типа объекта с контролем корректности (в случае неудачи инициируется исключение) и проверка объекта на принадлежность классу.
* Ссылки на классы придают дополнительный уровень гибкости, так, когда вы хотите динамически создавать объекты, чьи типы могут быть известны только во время выполнения кода. К примеру, ссылки на классы используются при формировании пользователем документа из разного типа объектов, где пользователь набирает нужные объекты из меню или палитры. Собственно, эта технология использовалась и при построении **Delphi.**
* введено средство, известное как механизм делегирования. Под ***делегированием*** понимается то, что некий объект может предоставить другому объекту отвечать на некоторые события. Он используется в Delphi для упрощения программирования событийно-ориентированных частей программ, т. е. пользовательского интерфейса и всевозможных процедур, запускаемых в ответ на манипуляции с базой данных.

После того как Borland внесла перечисленные изменения, получился мощный объектно-ориентированный язык, сопоставимый по своим возможностям с C++. Платой за новые функции стало значительное повышение требований к профессиональной подготовке программиста.

Язык программирования **Delphi** базируется на Borland Object Pascal.

Кроме того, **Delphi** поддерживает такие низкоуровневые особенности, как подклассы элементов управления Windows, перекрытие цикла обработки сообщений Windows, использование встроенного ассемблера.[22].

## Объектно-ориентированная модель программных компонент

Основной упор этой модели в **Delphi** делается на максимальном повторном использовании кода. Это позволяет разработчикам строить приложения весьма быстро из заранее подготовленных объектов, а также дает им возможность создавать свои собственные объекты для среды **Delphi**. Никаких ограничений по типам объектов, которые могут создавать разработчики, не существует. Действительно, все в **Delphi** написано на нем же, поэтому разработчики имеют доступ к тем же объектам и инструментам, которые использовались для создания среды разработки. В результате нет никакой разницы между объектами, поставляемыми Borland или третьими фирмами, и объектами, которые вы можете создать.

В стандартную поставку **Delphi** входят основные объекты, которые образуют удачно подобранную иерархию из 270 базовых классов. На **Delphi** можно одинаково хорошо писать как приложения к корпоративным базам данных, так и, к примеру, игровые программы. Во многом это объясняется тем, что традиционно в среде Windows было достаточно сложно реализовывать пользовательский интерфейс. Событийная модель в Windows всегда была сложна для понимания и отладки. Но именно разработка интерфейса в **Delphi** является самой простой задачей для программиста.

Благодаря такой возможности приложения, изготовленные при помощи **Delphi**, работают надежно и устойчиво. **Delphi** поддерживает использование уже существующих объектов, включая DLL, написанные на С и С++, OLE сервера, VBX, объекты, созданные при помощи **Delphi**. Из готовых компонент работающие приложения собираются очень быстро. Кроме того, поскольку **Delphi** имеет полностью объектную ориентацию, разработчики могут создавать свои повторно используемые объекты для того, чтобы уменьшить затараты на разработку.

**Delphi** предлагает разработчикам - как в составе команды, так и индивидуальным - открытую архитектуру, позволяющую добавлять компоненты, где бы они ни были изготовлены, и оперировать этими вновь введенными компонентами в визуальном построителе. Разработчики могут добавлять CASE-инструменты, кодовые генераторы, а также авторские help’ы, доступные через меню **Delphi.** [22].

## Библиотека визуальных компонент

Компоненты, используемые при разработке в **Delphi**, встроены в среду разработки приложений и представляют из себя набор типов объектов, используемых в качестве фундамента при строительстве приложения.

Этот костяк называется **Visual Component Library** (VCL). В VCL есть такие стандартные элементы управления, как строки редактирования, статические элементы управления, строки редактирования со списками, списки объектов. Еще имеются такие компоненты, которые ранее были доступны только в библиотеках третьих фирм: табличные элементы управления, закладки, многостраничные записные книжки. Все объекты разбиты на страницы по своей функциональности и представленны в *палитре компонент*.

VCL содержит специальный объект, предоставлющий интерфейс графических устройств Windows, и позволяющий разработчикам рисовать, не заботясь об обычных для программирования в среде Windows деталях.

Ключевой особенностью **Delphi** является возможность не только использовать визуальные компоненты для строительства приложений, но и создание новых компонент. Такая возможность позволяет разработчикам не переходить в другую среду разработки, а наоборот, встраивать новые инструменты в существующую среду. Кроме того, можно улучшить или полностью заменить существующие по умолчанию в **Delphi** компоненты.

Здесь следует отметить, что обычных ограничений, присущих средам визуальной разработки, в **Delphi** нет. Сам **Delphi** написан при помощи **Delphi**, что говорит об отсутствии таких ограничений.

Классы объектов построены в виде иерархии, состоящей из абстрактных, промежуточных, и готовых компонент. Разработчик может пользоваться готовыми компонентами, создавать собственные на основе абстрактных или промежуточных, а также создавать собственные объекты. Рассмотрим некоторые из них.



**TMainMenu** позволяет поместить главное меню в программу. При помещении TMainMenu на форму это выглядит, как просто иконка. Иконки данного типа называют *невизуальным компонентом*, поскольку они невидимы во время выполнения программы.

**TPopupMenu** позволяет создавать всплывающие меню. Этот тип меню появляется по щелчку правой кнопки мыши на объекте, к которому привязано данное меню. У всех видимых объектов имеется свойство PopupMenu, где и указывается нужное меню. Создается PopupMenu аналогично главному меню.



**TLabel** служит для отображения текста на экране. Можно изменить шрифт и цвет метки, если дважды щелкнуть на свойство Font в Инспекторе Объектов. Это легко сделать и во время выполнения программы, написав всего одну строчку кода.



**TEdit** - стандартный управляющий элемент Windows для ввода. Он может быть использован для отображения короткого фрагмента текста и позволяет пользователю вводить текст во время выполнения программы.



**TMemo -** иная форма TEdit. Подразумевает работу с большими текстами. TMemo может переносить слова, сохранять в ClipBoard фрагменты текста и восстанавливать их, и другие основные функции редактора. TMemo имеет ограничения на объем текста в 32Кб, это составляет 10-20 страниц (есть подобные компоненты, где этот предел снят).



**TButton** позволяет выполнить какие-либо действия при нажатии кнопки во время выполнения программы. В Delphi все делается очень просто. Поместив TButton на форму, по двойному щелчку можно создать заготовку обработчика события нажатия кнопки.



**TCheckBox** отображает строку текста с маленьким окошком рядом. В окошке можно поставить отметку, которая означает, что что-то выбрано.



**TRadioButton** позволяет выбрать только одну опцию из нескольких.



**TListBox** нужен для показа прокручиваемого списка. Классический пример ListBox’а в среде Windows - выбор файла из списка в пункте меню File | Open многих приложений. Названия файлов или директорий и находятся в ListBox’е.



**TComboBox** во многом напоминает ListBox, за исключением того, что позволяет водить информацию в маленьком поле ввода сверху ListBox. Есть несколько типов ComboBox, но наиболее популярен спадающий вниз (drop-down combo box), который можно видеть внизу окна диалога выбора файла.



**TScrollbar -** полоса прокрутки, появляется автоматически в объектах редактирования, ListBox’ах при необходимости прокрутки текста для просмотра.



**TGroupBox** используется для визуальных целей и для указания Windows, каков порядок перемещения по компонентам на форме (при нажатии клавиши TAB).



**TRadioGroup** используется аналогично TGroupBox, для группировки объектов TRadioButton.



**TPanel** - управляющий элемент, похожий на TGroupBox, используется в декоративных целях. Чтобы использовать TPanel, можно просто поместить его на форму и затем положите другие компоненты на него. Теперь при перемещении TPanel будут передвигаться и эти компоненты. TPanel используется также для создания линейки инструментов и окна статуса.



**TBitBtn -** кнопка вроде TButton, однако на ней можно разместить картинку (glyph). TBitBtn имеет несколько предопределенных типов (bkClose, bkOK и др), при выборе которых кнопка принимает соответствующий вид. Кроме того, нажатие кнопки на модальном окне приводит к закрытию окна с соответствующим модальным результатом.



**TSpeedButton -** кнопка для создания панели быстрого доступа к командам (SpeedBar). Пример - SpeedBar слева от Палитры Компонент в среде Delphi. Обычно на данную кнопку помещается только картинка (glyph).



**TTabSet** - горизонтальные закладки. Обычно используется вместе с TNoteBook для создания многостраничных окон. Название страниц можно задать в свойстве Tabs.



**TNoteBook** - используется для создания многостраничного диалога, на каждой странице располагается свой набор объектов. Используется совместно с TTabSet.



**TTabbedNotebook** - многостраничный диалог со встроенными закладками, в данном случае - закладки сверху.



**TMaskEdit -** аналог TEdit, но с возможностью форматированного ввода. Формат определяется в свойстве EditMask. В редакторе свойств для EditMask есть заготовки некоторых форматов: даты, валюты и т.п.



**TOutline** - используется для представления иерархических отношений связанных данных. Например - дерево директорий.



**TStringGrid -** служит для представления текстовых данных в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство Cell.



**TDrawGrid** - служит для представления данных любого типа в виде таблицы. Доступ к каждому элементу таблицы происходит через свойство CellRect.



**TImage -** отображает графическое изображение на форме. Воспринимает форматы BMP, ICO, WMF. Если картинку подключить во время дизайна программы, то она прикомпилируется к EXE файлу.



**TShape -** служит для отображения простейших графических объектов на форме: окружность, квадрат и т.п.



**TBevel** - элемент для рельефного оформления интерфейса.



**THeader** - элемент оформления для создания заголовков с изменяемыми размерами для таблиц.



**TScrollBox** - позволяет создать на форме прокручиваемую область с размерами большими, нежели экран. На этой области можно разместить свои объекты.



**TTimer -** таймер, событие OnTimer периодически вызывается через промежуток времени, указанный в свойстве Interval. Период времени может составлять от 1 до 65535 мс.



**TPaintBox -** место для рисования. В обработчики событий, связанных с мышкой передаются относительные координаты мышки в TPaintBox, а не абсолютные в форме.



**TFileListBox -** специализированный ListBox, в котором отображаются файлы из указанной директории (св-во Directory). На названия файлов можно наложить маску, для этого служит св-во Mask. Кроме того, в св-ве FileEdit можно указать объект TEdit для редактирования маски.



**TDirectoryListBox -** специализированный ListBox, в котором отображается структура директорий текущего диска. В св-ве FileList можно указать TFileListBox, который будет автоматически отслеживать переход в другую директорию.



**TDriveComboBox -** специализированный ComboBox для выбора текущего диска. Имеет свойство DirList, в котором можно указатьTDirectoryListBox, который будет отслеживать переход на другой диск.



**TFilterComboBox** - специализированный ComboBox для выбора маски имени файлов. Список масок определяется в свойстве Filter. В свойстве FileList указывается TFileListBox, на который устанавливается маска.



С помощью последних четырех компонент (TFileListBox, TDirectoryListBox, TDriveComboBox, TFilterComboBox) можно построить свой собственный диалог выбора файла, причем для этого не потребуется написать ни одной строчки кода.

**TMediaPlayer -** служит для управления мултимедйными устройствами (типа CD-ROM, MIDI и т.п.). Выполнен в виде панели управления с кнопками Play, Stop, Record и др. Для воспроизведения может понадобиться как соответствующее оборудование, так и программное обеспечение. Подключение устройств и установка ПО производится в среде Windows. Например, для воспроизведения видео, записанного в формате AVI, в потребуется установить ПО MicroSoft Video (в Windows 3.0, 3.1, WFW 3.11).



**TOLEContainer** - контейнер, содержащий OLE объекты. Поддерживается OLE 2.02

**TDDEClientConv,TDDEClientItem, TDDEServerConv, TDDEServerItem -** 4 объекта для организации DDE. С помощью этих объектов можно построить приложение как DDE-сервер, так и DDE-клиент.



**TChartFX -** деловая графика. Компонент позволяет строить всевозможные графики и гистограммы.



## Формы, модули и метод разработки “Two-Way Tools”

***Формы*** - *это объекты, в которые помещаются другие объекты для создания пользовательского интерфейса любого приложения*. ***Модули*** *состоят из кода, который реализует функционирование приложения, обработчики событий для форм и их компонент*.

Информация о формах хранится в двух типах файлов - *.dfm* и *.pas*, причем первый тип файла - двоичный - хранит образ формы и ее свойства, второй тип описывает функционирование **обработчиков событий** и поведение компонент. Оба файла автоматически синхронизируются **Delphi**, так что если добавить новую форму проект, связанный с ним файл *.pas* автоматически будет создан, и его имя будет добавлено в проект.

Такая синхронизация и делает Delphi ***two-way-инструментом***, обеспечивая полное соответствие между кодом и визуальным представлением. Как только добавляется новый объект или код, **Delphi** устанавливает т.н**. *“кодовую синхронизацию”*** между визуальными элементами и соответствующими им кодовыми представлениями.

***Two-way tools*** - *однозначное соответствие между визуальным проектированием и классическим написанием текста программы* Это означает, что разработчик всегда может видеть код, соответствующий тому, что он построил при помощи визуальных инструментов и наоборот.

Визуальный построитель интерфейсов (Visual User-interface builder) дает возможность быстро создавать клиент-серверные приложения визуально, просто выбирая компоненты из соответствующей палитры. В процессе построения приложения разработчик выбирает из палитры компонент готовые компоненты как художник, делающий крупные мазки кистью. Еще до компиляции он видит результаты своей работы - после подключения к источнику данных их можно видеть отображенными на форме, можно перемещаться по данным, представлять их в том или ином виде.[4, 22].

## Масштабируемые средства для построения баз данных

Мощность и гибкость Delphi при работе с базами данных основана на низкоуровневом ***ядре*** - процессоре баз данных ***Borland Database Engine*** (BDE). Его интерфейс с прикладными программами называется *Integrated Database Application Programming Interface* (IDAPI). В принципе, сейчас не различают эти два названия (BDE и IDAPI) и считают их синонимами. BDE позволяет осуществлять доступ к данным как с использованием традиционного record-ориентированного (навигационного) подхода, так и с использованием set-ориентированного подхода, используемого в SQL-серверах баз данных. Кроме BDE, Delphi позволяет осуществлять доступ к базам данных, используя технологию (и, соответственно, драйверы) Open DataBase Connectivity (ODBC) фирмы Microsoft. Но, как показывает практика, производительность систем с использованием BDE гораздо выше, чем оных при использовании ODBC. ODBC драйвера работают через специальный “ODBC socket”, который позволяет встраивать их в BDE.

Все инструментальные средства баз данных Borland - Paradox, dBase, Database Desktop - используют BDE. Все особенности, имеющиеся в Paradox или dBase, “наследуются” BDE, и поэтому этими же особенностями обладает и Delphi.

Библиотека объектов содержит набор визуальных компонент, значительно упрощающих разработку приложений для СУБД с архитектурой клиент-сервер. Объекты инкапсулируют в себя нижний уровень - Borland Database Engine.

Предусмотрены специальные наборы компонент, отвечающих за доступ к данным, и компонент, отображающих данные. Компоненты доступа к данным позволяют осуществлять соединения с БД, производить выборку, копирование данных, и т.п.

Компоненты визуализации данных позволяют отображать данные виде таблиц, полей, списков. Отображаемые данные могут быть текстового, графического или произвольного формата.

Таблицы сохраняются в базе данных. Некоторые СУБД сохраняют базу данных в виде нескольких отдельных файлов, представляющих собой таблицы (в основном, все локальные СУБД), в то время как другие состоят из одного файла, который содержит в себе все таблицы и индексы (InterBase). Например, таблицы dBase и Paradox всегда сохраняются в отдельных файлах на диске. Директорий, содержащий dBase *.DBF* файлы или Paradox *.DB* файлы, рассматривается как база данных. Другими словами, любой директорий, содержащий файлы в формате Paradox или dBase, рассматривается Delphi как единая база данных. Для переключения на другую базу данных нужно просто переключиться на другой директорий. InterBase сохраняет все таблицы в одном файле, имеющем расширение *.GDB*, поэтому этот файл и есть база данных InterBase.

Объекты БД в **Delphi** основаны на SQL и включают в себя полную мощь Borland Database Engine. В состав **Delphi** также включен Borland SQL Link, поэтому доступ к СУБД Oracle, Sybase, Informix и InterBase происходит с высокой эффективностью. Кроме того, **Delphi** включает в себя локальный сервер Interbase для того, чтобы можно было разработать расширяемые на любые внешние SQL-сервера приложения в офлайновом режиме. Разработчик в среде **Delphi**, проектирующий информационную систему для локальной машины (к примеру, небольшую систему учета медицинских карточек для одного компьютера), может использовать для хранения информации файлы формата **.dbf** (как в dBase или Clipper) или **.db** (Paradox). Если же он будет использовать локальный InterBase for Windows 4.0 (это локальный SQL-сервер, входящий в поставку), то его приложение безо всяких изменений будет работать и в составе большой системы с архитектурой клиент-сервер.

***Масштабируемость*** на практике - одно и то же приложение можно использовать как для локального, так и для более серьезного клиент-серверного вариантов.[4, 22].

В состав пакета Delphi также входит множество утилит для работы и управления базами данных. Вот некоторые из них.

***Database Desktop*** - это утилита, во многом похожая на Paradox, которая поставляется вместе с Delphi для интерактивной работы с таблицами различных форматов локальных баз данных - Paradox и dBase, а также SQL-серверных баз данных InterBase, Oracle, Informix, Sybase (с использованием SQL Links). Она позволяет создавать как структуру реляционных таблиц, так и всевозможные ограничения целостности таблиц, индексы, первичные ключи и внешние ключи.



***WISQL*** (Windows Interactive SQL) - интерактивное средство посылки SQL-запросов к InterBase (в том числе и локальному InterBase), входящее в поставку Delphi, позволяет создавать таблицы - через посылку SQL-запросов. Database Desktop не обладает всеми возможностями по управлению SQL-серверными базами данных. Поэтому с помощью Database Desktop удобно создавать или локальные базы данных или только простейшие SQL-серверные базы данных, состоящие из небольшого числа таблиц, не очень сильно связанных друг с другом. Если же необходимо создать базу данных, состоящую из большого числа таблиц, имеющих сложные взаимосвязи, можно воспользоваться языком SQL. Можно записать всю последовательность SQL-предложений в один так называемый скрипт и послать его на выполнение. Конкретные реализации языка SQL незначительно отличаются в различных SQL-серверах, однако базовые предложения остаются одинаковыми для всех реализаций. Практика показывает, что если нет необходимости создавать таблицы во время выполнения программы, то лучше воспользоваться WISQL.



***InterBase*** - это система управления реляционными базами данных, поставляемая корпорацией BORLAND для построения приложений с архитектурой клиент-сервер произвольного масштаба: от сетевой среды небольшой рабочей группы с сервером под управлением Novell NetWare или Windows NT на базе IBM PC до информационных систем крупного предприятия на базе серверов IBM, Hewlett-Packard, SUN и т.п.

В пакет Delphi входит однопользовательская версия InterBase для Windows - Local InterBase. Используя Local InterBase можно создавать и отлаживать приложения, работающие с данными по схеме клиент-сервер, без подключения к настоящему серверу. В дальнейшем потребуется только перенастроить используемый псевдоним базы данных и программа будет работать с реальной базой без перекомпиляции. Кроме того, Local InterBase можно использовать в приложениях для работы с данными вместо таблиц Paradox.

Важной составной частью приложения является вывод данных на печать - получение отчета. В пакет Delphi входит средство для генерации и печати отчетов - ***ReportSmith***. Вы можете объединить отчет с приложениями Delphi. Также, библиотека визуальных компонент Delphi включает специальный компонент TReport. В данном уроке показано, как использовать компоненту TRepor и рассмотрены основные принципы проектирования отчетов в ReportSmith.

Borland ReportSmith является инструментом для получения отчетов и интегрирован в среду Delphi. Отчет может быть добавлен к приложениям Delphi. Отчеты могут быть созданы для SQL БД или локальных БД и не требуют знания сложных команд БД. Интерфейс ReportSmith использует стандартные инструменты Windows типа tool bar, formatting ribbon, и “drag and drop”. Если пользователь уже знаком с интерфейсом стандартных Windows-программ, типа Word for Windows или Quattro Pro for Windows, ему будет “знаком” и интерфейс ReportSmith. ReportSmith предлагает 4 типа отчетов: Табличный, Кросс-таблица(CrossTab), Форма(Form) и Наклейка(Label).

ReportSmith использует концепцию “живых данных”, т.е. работа происходит с настоящими данными все время, а не только тогда, когда запускается просмотр (preview). Кроме этого, ReportSmith легко работает с чрезвычайно большими БД при помощи адаптивной технологии управления памятью. В ReportSmith можно управлять тем, где сохраняется результат выборки данных из БД: в локальный памяти клиентской PC, на жестком диске клиентской PC, или на сервере.

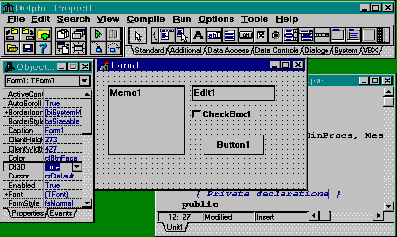
## Настраиваемая среда разработчика

После запуска **Delphi** в верхнем окне горизонтально располагаются иконки палитры компонент. Если курсор задерживается на одной из иконок, под ней в желтом прямоугольнике появляется подсказка

Из этой палитры компонент можно выбирать компоненты, из которых можно строить приложения. Компоненты включают в себя как визуальные, так и логические компоненты. Такие вещи, как кнопки, поля редактирования - это визуальные компоненты; а таблицы, отчеты - это логические.

Поскольку в **Delphi** программа строится визуальным образом, все эти компоненты имеют свое графическое представление в поле форм для того, чтобы можно было бы ими соответствующим образом оперировать. Но для работающей программы видимыми остаются только визуальные компоненты. Компоненты сгруппированы на страницах палитры по своим функциям. К примеру, компоненты, представляющие Windows “common dialogs” все размещены на странице палитры с названием “Dialogs”.

**Delphi** позволяет разработчикам настроить среду для максимального удобства. Можно легко изменить палитру компонент, инструментальную линейку, а также настраивать выделение синтаксиса цветом.



В **Delphi** можно определить свою группу компонент и разместить ее на странице палитры, а если возникнет необходимость, перегруппировать компоненты или удалить неиспользуемые.

* **Интеллектуальный редактор.** Редактирование программ можно осуществлять, используя запись и исполнение макросов, работу с текстовыми блоками, настраиваемые комбинации клавиш и цветовое выделение строк.
* ***Графический отладчик***. **Delphi** обладает мощнейшим, встроенным в редактор графическим отладчиком, позволяющим находить и устранять ошибки в коде. Можно установить точки останова, проверить и изменить переменные, при помощи пошагового выполнения в точности понять поведение программы. Если же требуются возможности более тонкой отладки, можно использовать отдельно доступный Turbo Debugger, проверив ассемблерные инструкции и регистры процессора.
* ***Инспектор объектов***. Этот инструмент представляет из себя отдельное окно, где вы можете в период проектирования программы устанавливать значения свойств и событий объектов (Properties & Events).
* ***Менеджер проектов.*** Дает возможность разработчику просмотреть все модули в соответствующем проекте и снабжает удобным механизмом для управления проектами. Менеджер проектов показывает имена файлов, время/дату выбранных форм и пр. Можно немедленно попась в текст или форму, просто щелкнув мышкой на соответствующее имя.
* ***Навигатор объектов.*** Показывает библиотеку доступных объектов и осуществляет навигацию по приложению. Можно посмотреть иерархию объектов, прекомпилированные модули в библиотеке, список глобальных имен вашего кода.
* ***Дизайнер меню.*** Можно создавать меню, сохранить созданные в виде шаблонов и затем использовать в их в любом приложении.
* ***Эксперты.*** Это набор инструментальных программ, облегчающих проектирование и настройку Ваших приложений. Есть возможность подключать самостоятельно разработанные эксперты. Потенциально это та возможность, при помощи которой третьи фирмы могут расширять **Delphi** CASE-инструментами, разработанными специально для **Delphi.** Включает в себя:

Эксперт форм, работающих с базами данных

Эксперт стилей и шаблонов приложений

Эксперт шаблонов форм

В состав RAD Pack входит эксперт для преобразования ресурсов, изготовленных в Borland Pascal 7.0, в формы **Delphi.** Уже появились эксперты, облегчающие построение DLL и даже написание собственных экспертов

* ***Интерактивная обучающая система.*** Позволяет более полно освоить **Delphi**. Она являются не просто системой подсказок, а показывает возможности **Delphi** на самой среде разработчика.

## Незначительные требования к аппаратным и программным средствам

Delphi это высокопроизводительный инструмент создания приложений. Для запуска Delphi требуется как минимум 386 компьютер с 4MB памяти. Более подходящей машиной будет 486DX 66MHz с 8MB ОЗУ.

Небольшие программы, созданные на Delphi будут работать на любом компьютере. Другими словами, они не требуют того ОЗУ или скорости процессора, что необходимо для среды Delphi. [4].

# Проектирование базы данных

Проектирование БД связанно с разрешением проблем представления данных между конечными пользователями. Они продиктованы различными потребностями и задачами лиц, которые используют эти данные. Пользователи могут быть выделены в отдельные группы. Каждая из групп воздействует на результаты проектирования в разных направлениях. Необходимо собрать информацию о реальных и потенциальных приложениях, а также о пользователях базы данных, чтобы устранить все противоречия ещё на начальном этапе, так как многолетний мировой опыт использования информационных систем, построенных на основе баз данных, показывает, что недостатки проекта допущенные на этапе проектирования невозможно устранить любыми ухищрениями в программах приложений.

Проектирование обычно поручается человеку (группе лиц) – ***администратору базы данных* (АБД)**. Им может быть как специально выделенный сотрудник, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

В основу проектирования БД должны быть положены представления ***конечных пользователей***конкретной организации — концептуальные требования к систе­ме. Именно конечный пользователь в своей работе принимает решения с учетом получаемой в результате доступа к базе данных информации. От оперативности и качества этой информации будет зависеть эффективность работы организации. Данные, помещаемые в базу данных, также предоставляет конечный пользова­тель. Кроме того, БД должна предоставлять доступ к данным пользователям, которые практически не имеют или не хотят иметь представления о физическом размещении в памяти данных и их описаний, о механизмах поиска запрашиваемых данных или о поддержании баз данных в актуальном состоянии.[15].

***Прикладные программисты*** хотели бы иметь в одном месте (например, в одной таблице) все данные, необходимые им для реализации запроса из прикладной программы или с терминала.

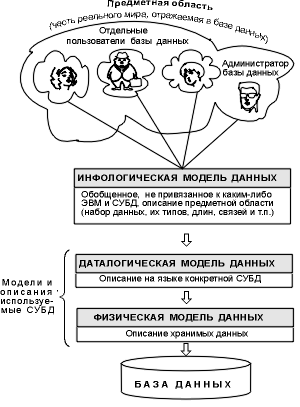
АБД же заботятся об исключении возможных искажений хранимых данных при вводе в базу данных новой информации и обновлении или удалении существующей. Для этого они удаляют из базы данных дубликаты и нежелательные функциональные связи между таблицами, разбивая базу данных на множество маленьких таблиц.

Чтобы различать представления данных конечными пользователями, программистами и АБД создаются разные уровни моделей данных. Их общая структура представлена на рис. 14.

Основное различие между указанными выше тремя типами моделей данных (концептуальной, логической и физической) состоит в способах представления взаимосвязей между объектами. При проектировании БД требуется различать взаимосвязи между объектами, между свойствами одного объекта и между свойствами различных объектов. Рассмотрим каждую из них более подробно.

## Инфологическая модель данных

**Рис. 14. Уровни моделей данных.**



***Предметная область*** *– часть реального мира отражённая в базу данных.*

Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, АБД сначала создает обобщенное неформальное описание создаваемой базы данных. Это *описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют* ***инфологической моделью данны****х*.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком. Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

***Сущность*** – *любой реальный или не реальный различимый объект (объект, который мы можем отличить от другого), информацию о котором необходимо хранить в базе данных.* Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет и т.д. Необходимо различать такие понятия, как *тип сущности* и *экземпляр сущности*. Понятие ***тип сущности*** относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. ***Экземпляр сущности*** относится к конкретной вещи в наборе. Например, типом сущности может быть ГОРОД, а экземпляром – Москва, Киев и т.д.

***Атрибут*** – *поименованная характеристика сущности.* Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей (например, ЦВЕТ может быть определен для многих сущностей: СОБАКА, АВТОМОБИЛЬ, ДЫМ и т.д.). Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности АВТОМОБИЛЬ являются ТИП, МАРКА, НОМЕРНОЙ ЗНАК, ЦВЕТ и т.д. Здесь также существует различие между типом и экземпляром. Тип атрибута ЦВЕТ имеет много экземпляров или значений: Красный, Синий, Банановый, Белая ночь и т.д., однако каждому экземпляру сущности присваивается только одно значение атрибута.

Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для автомобильного завода цвет – это только атрибут продукта производства, а для лакокрасочной фабрики цвет – тип сущности.

***Связь*** – *ассоциирование двух или более сущностей.* Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (***рекурсивная связь***). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность инфологических моделей.

При определении инфологической модели необходимо принимать во внимание следующее:

* База данных должна удовлетворять актуальным информационным пот­ребностям организации. Получаемая информация должна по структуре и содержанию соответствовать решаемым задачам.
* База данных должна обеспечивать получение требуемых данных за приемлемое время, то есть отвечать заданным требованиям производи­тельности.
* База данных должна удовлетворять выявленным и вновь возникающим требованиям всех пользователей.
* База данных должна легко расширяться при реорганизации и расшире­нии предметной области.
* База данных должна легко изменяться при изменении программной и аппаратной среды.

## Инфологическая модель данных "сущность-связь"

Инфологическая модель отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных. Существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель "сущность-связь" и т.д. Наиболее популярной из них оказалась модель ***"сущность-связь"*** или называемая ещё ***ER-моделью*** (от англ. Entity-Relationship, т.е. сущность-связь).

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных.

В них сущности изображаются помеченными прямоугольниками, ассоциации (связи)– помеченными ромбами или шестиугольниками, атрибуты – помеченными овалами, а связи между ними – ненаправленными ребрами, над которыми может проставляться степень связи (1 или буква, заменяющая слово "много") и необходимое пояснение.

Между двумя сущностям, например, А и В возможны четыре вида связей.

Первый тип – ***связь ОДИН-К-ОДНОМУ*** (1:1): в каждый момент времени каждому представителю (экземпляру) сущности А соответствует 1 или 0 представителей сущности В:



Студент может не "заработать" стипендию, получить обычную или одну из повышенных стипендий. Или, допус­тим, в определенный момент времени один клиент может сделать только один заказ. В этом случае между объектами КЛИЕНТ и ЗАКАЗ устанавливается взаимосвязь “один к одному”, обозначаемая одинарными стрелками.

Между данными, хранящимися в объектах КЛИЕНТ и ЗАКАЗ, будет сущест­вовать взаимосвязь, в которой каждая запись в одном объекте будет однозначно указывать на запись в другом объекте. Ни в одном, ни в другом объекте не может существовать записи, не связанной с какой-либо записью в другом объекте.

Второй тип – ***связь ОДИН-КО-МНОГИМ*** (1:М): одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности В.



Квартира может пустовать, в ней может жить один или несколько жильцов. Или, например, в определенный момент времени один клиент может стать обладателем несколь­ких моделей автомобилей, при этом несколько клиентов не могут являться обладателями одного автомобиля. Взаимосвязь “один ко многим” можно обоз­начить с помощью одинарной стрелки в направлении к “одному” и двойной стрелки в направлении ко “многим” .В этом случае одной записи данных первого объекта (его часто называют родительским или основным) будет соответствовать несколько записей второго объекта (дочернего или подчиненного). Взаимосвязь “один ко многим” очень распространена при разработке реляционных баз данных.

Третий тип – ***связь МНОГИЕ-К-ОДНОМУ*** (М:1): одному представителю сущности B соответствуют 0, 1 или несколько представителей сущности А.

В принципе нет никакой разницы между связью ОДИН-КО-МНОГИМ и МНОГИЕ-К-ОДНОМ, т.к. между двумя сущностями возможны связи в обоих направлениях и всё зависит от того с какими сущностями связаны данные.

Четвёртый тип – ***связь МНОГИЕ-КО-МНОГИМ*** (N:М): одному представителю сущности B соответствуют 0, 1 или несколько представителей сущности А и одновременно одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности В.

Это также обусловлено тем, что между двумя сущностями возможны связи в обоих направлениях.

Если связь между сущностями МУЖЧИНЫ и ЖЕНЩИНЫ называется БРАК, то существует четыре возможных представления такой связи:



Или например каждый продавец может обслуживать нескольких клиентов. С другой стороны, приобретая автомобили в различное время, каждый клиент вполне может быть обслужен различными продавцами. Между объектами КЛИЕНТ и ПРОДАВЕЦ существует взаимосвязь “многие ко многим”. Такая взаимосвязь обозначается двойными стрелками.

Характер связей между сущностями не ограничивается перечисленными. Существуют и более сложные связи:



* множество связей между одними и теми же сущностями

(пациент, имея одного лечащего врача, может иметь также несколько врачей-консультантов; врач может быть лечащим врачом нескольких пациентов и может одновременно консультировать несколько других пациентов);



* тренарные связи:
* связи более высоких порядков, семантика (смысл) которых иногда очень сложна.

В приведенных примерах связей не показаны атрибуты сущностей и ассоциаций во всех ER-диаграммах. Так, ввод лишь нескольких основных атрибутов в описание брачных связей значительно усложнит ER-диаграмму (рис. 2.1,а). В связи с этим язык ER-диаграмм используется для построении небольших моделей и иллюстрации отдельных фрагментов больших. Чаще же применяется менее наглядный, но более содержательный ***язык инфологического моделирования*** (ЯИМ), в котором сущности и ассоциации представляются предложениями вида:

СУЩНОСТЬ (атрибут 1, атрибут 2 , ..., атрибут n)

АССОЦИАЦИЯ [СУЩНОСТЬ S1, СУЩНОСТЬ S2, ...]

(атрибут 1, атрибут 2, ..., атрибут n)

где S – степень связи, а атрибуты, входящие в ключ, должны быть отмечены с помощью подчеркивания.

Так, рассмотренный выше пример множества связей между сущностями, может быть описан на ЯИМ следующим образом:

Врач (Номер\_врача, Фамилия, Имя, Отчество, Специальность)

Пациент (Регистрационный\_номер, Номер койки, Фамилия,

Имя, Отчество, Адрес, Дата рождения, Пол)

Лечащий\_врач [Врач 1, Пациент M]

(Номер\_врача, Регистрационный\_номер)

Консультант [Врач M,Пациент N]

(Номер\_врача, Регистрационный\_номер).

Для примера ER-диаграмма базы данных "Питание"показана на рис.16, а модель на языке ЯИМ имеет следующий вид:

Блюда (БЛ, Блюдо, Вид)

Продукты (ПР, Продукт, Калорийность)

Поставщики (ПОС, Город, Поставщик) [Город]

Состав [Блюда M, Продукты N] (БЛ, ПР, Вес (г))

Поставки [Поставщики M, Продукты N] (ПОС, ПР, Дата\_П, Цена, Вес (кг))

Города (Город, Страна)

Рецепты (БЛ, Рецепт) {Блюда}

Расход (БЛ, Дата\_Р, Порций) {Блюда}

В этих моделях Блюдо, Продукт и Поставщик – наименования, а БЛ, ПР и ПОС – цифровые коды блюд, продуктов и организаций, поставляющих эти продукты.

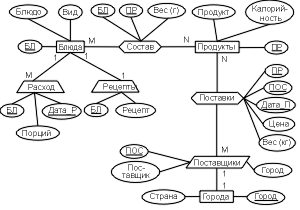


Рис. 15. ER-диаграмма базы данных ‘Питание’.



Рис. 16. Диаграмма “Таблицы-связи”

Существует ещё наиболее распространенная модификация ER-диаграмм для представления инфологической модели баз данных - "Таблица-связь", пример использования которого приведен на рис. 16. В нем все сущности изображаются одностолбцовыми таблицами с заголовками, состоящими из имени и типа сущности. Строки таблицы – это перечень атрибутов сущности, а те из них, которые составляют первичный ключ, располагаются рядом и обводятся рамкой. Связи между сущностями указываются стрелками, направленными от первичных ключей или их составляющих. Именно этот тип диаграмм будет использоваться при построении инфологической модели базы данных, разрабатываемой в данной дипломной работе.

## Даталогическая модель данных

*Описание, создаваемое по инфологической модели данных, называют* ***даталогической моделью данных***. Даталогическая модель отражает логические связи между элементами данных вне зависимости от их содержания и среды хранения. Пользователям выделяются подмножества этой логической модели, назы­ваемые ***внешними моделями***, отражающие их представления о предметной области. Внешняя модель соответствует представлениям, которые пользователи получают на осно­ве логической модели, в то время как концептуальные требования отражают представления, которые пользователи первоначально желали иметь и которые легли в основу разработки инфологической модели. Даталогическая модель отобра­жается в физическую память, такую, как диск, лента или какой-либо другой носитель информации. Даталогическая модель в основном используется прикладными программистами для реализации требований, которые выдвинули конечные пользователи, отражённых в инфологической концептуальной модели.

Типы даталогических моделей уже обсуждались нами ранее. Это есть не что иное, как Модели представления данных, т.о. даталогическая модель данных может быть *реляционной, иерархической* или *сете­вой*.

При разработке даталогической модели, кроме требований предъявляемых для построения инфологической модели, предъявляются дополнительные требования:

* Загруженные в базу данных корректные данные должны оставаться корректными.
* Данные до включения в базу данных должны проверяться на достовер­ность.
* Доступ к данным, размещаемым в базе данных, должны иметь только лица с соответствующими полномочиями.
* Разрешение проблем, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
* Способы обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;

Если инфологическая модель данных предназначена для наглядного отражения представления пользователей, т.е. является человеко-ориентированной, то даталогическая модель уже является компьютеро-ориентированной. С её помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных.

## Переход от ER – модели к реляционной.

Переход от инфологической модели “сущность-связь”- это сравнительно простая задача, поскольку в терминологии и принципах ER-модели и реляционного подхода имеется взаимно однозначное соответствие. Существует ряд хорошо зарекомендовавших себя правил с пощью которых из ER-диаграмм отроются реляционные таблицы.

1. Каждая простая сущность превращается в таблицу. ***Простая сущность*** - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.
2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.
3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.
4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.
5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.
6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа: *все подтипы в одной таблице* (а) или *для каждого подтипа - отдельная таблица* (б). При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА; он становится частью первичного ключа. При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).
7. Имеется два способа работы при наличии исключающих связей: *общий столбец* и *явные внешние ключи* (б). Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи. Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

## Физическая модель данных

***Физическая модель данных –*** *модель, определяющая размещение данных на внешних носителях, методы доступа и технику индексирования*. Она так же называется ***внутренней моделью*** системы.

Внешние модели (даталогические модели) никак не связаны с типом физической памяти, в которой будут храниться данные, и с методами доступа к этим данным. Внутренние модели (физические модели) наоборот определяют и оперируют размещением данных и их взаимосвязях на запоминающих устройствах.

Физическая организация данных оказывает основное влияние на эксплуатационные характеристики БД. Разработчики СУБД пытаются создать наиболее производительные физические модели данных, предлагая пользователям тот или иной инструментарий для поднастройки модели под конкретную БД. Существует большое разнообразие способов реализации и корректировки физических моделей современных промышленных БД, что не позволяет рассмотреть их подробно.

Физическая модель данных является полностью компьютерно-ориентированной и конечные пользователи, а порой и прикладные программисты, не имеют никакого представления о том, каким образом данные запоминаются и извлекаются или каким способом организуются индексы в таблицах для быстрого поиска или ссылочная целостность. Эти и множество других функций по методам доступа и поддержании баз данных на внешних носителях, а также способов поиска и доступа к данным в современных СУБД обеспечивается в основном ***ядром базы данных***, что значительно облегчает задачу создания БД и их ведение.

Трехуровневая архитектура (инфологический, даталогический и физический уровни) позволяет обеспечить ***независимость хранимых данных*** от использующих их программ. АБД может при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь физическую модель данных. АБД может подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, даталогическую модель. Указанные изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся "прозрачными" для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

## Этапы проектирования базы данных

Этапы проектирования базы данных с учетом рассмотренных выше аспектов:

1. Проектирование инфологической концептуальной модели баз данных:

а) Исследование предметной области применения и выявление требований конечных пользователей и решаемых задач.

в) Анализ данных: сбор основных данных (объекты, связи между объектами).

с) Построение ER-диаграммы базы данных.

1. Проектирование даталогической модели базы данных (учитывать требования СУБД ).

a) Потенциально возможные прикладные программы: сбор информации о потенциальных возможностях использования данных.

1. Проектирование физической модели базы данных (оценка эксплуатационных характеристик прикладных программ).
2. Реализациябазы данных (оценка при неудовлетворительных эксплуатационных характеристиках).

# Практическая часть

## Предметная область и задачи, возложенные на базу данных

Для демонстрации практического примера организации базы данных с помощью описанных в дипломной работе средств, спроектируем базу данных для хранения информации о качественных характеристиках и количестве зерна пшеницы. Объект - зерноперерабатывающее предприятие мукомольной промышленности. Технология приемки зерна следующая: клиенты привозят на предприятие зерно, работники производственной лаборатории берут пробы зерна с каждой машины и проводят лабораторные анализы, в результате которых определяются такие характеристики зерна – влажность, зерновая примесь, сорная примесь, проход (сито), клейковина, натура, стекловидность. Все это качественные характеристики, от которых зависит количество муки, которое заберет клиент в обмен на зерно. Наша цель – спроектировать базу данных, в которой будет храниться информация о зерне, принятом от клиентов. Подразумевается, что информация накапливается постоянно с каждым днем, она может изменяться; данная база данных является частью большого комплекса автоматизированной системы управления предприятием (АСУП).

База данных несомненно носит характер фактографической информационной системы и должна выдавать однозначные сведения на поставленные запросы. Конечными пользователями базы данных являются работники производственной лаборатории, зерноперерабатывающего цеха, бухгалтерии (отдел сбыта готовой продукции), которые относятся к категории пользователей не искушенных в вопросах ведения, администрирования баз данных и поддержании их в актуальном состоянии. Это накладывает определенные требования на разработку системы управления базой данных, при которой все методы доступа, поиска и большинство функций администрирования скрыты внутри программы и прозрачны при работе что, несомненно, скажется на разработке программного интерфейса. Более подробно все требования перечислены ниже:

1. ***Предоставление общей информации о количестве принятого зерна и его качественных характеристиках.*** Это совокупность сведений о количестве зерна, привезенном каждым клиентом, включает в себя *общую* информацию такую как название фирмы клиента или фамилия (если это физическое лицо), дата приемки зерна, номер автомашины, класс пшеницы, масса брутто (с автомашиной), масса тары (автомашины), масса нетто (чисто зерно), номер склада (куда поступило зерно). Подразумевается, что информация будет изменятся и пополнятся постоянно.
2. ***Ведение справочника клиентов****.* Клиенты бывают постоянные и одноразовые, но, несмотря на это, информация о них остается в базе данных. Справочник постоянно пополняется, редактируется. Как правило, удалением информации о клиентах и пополнением справочника занимается один человек.
3. ***Ведение справочника сельськохозяйственных культур****.*По каждой культуре по каждому классу необходимо вести данные о базисных качественных характеристиках, на основе которых в дальнейшем будет определяться отклонение фактических характеристик от базисных. Как правило, удалением информации о культурах и пополнением справочника занимается тоже один человек. Нужно учитывать и тот факт, что в будущем предприятие может работать не только с пшеницей, а и с другими культурами, например, с кукурудзой.
4. ***Распределение клиентов****.* С этой задачей сталкиваются в процессе формирования квитанций на выдачу муки. Идет распределение по плательщикам и неплательщикам НДС, по юридическим и физическим лицам.
5. ***Предоставление информации о конкретном клиенте.*** Все информация о поступлении зерна ведется в разрезе каждого клиента и может быть выдана по требованию самого клиента или, например, работникам налоговой инспекции.
6. ***Получение статистической информации о качестве и количестве зерна****.* По результатам работы за период (как правило, год) формируются статистические отчеты для статистического управления о средних качественных показателях зерна.

## Определение объектов базы данных

Анализ определенных выше задач позволяет выделить сущности (объекты) проектируемой базы данных и, построить ее инфологическую модель на языке "Таблицы-связи". В результате анализа были определены следующие объекты базы данных:

1. **Клиенты** *(Код клиента, Название клиента, Тип клиента, Банковские реквизиты, Юридический адрес, Телефон)*

Эта сущность отводится для хранения основных сведений о клиентах. Реквизит “Тип клиента” введен для указания юридического или физического лица, а также – плательщик или неплательщик он НДС. Реквизит “Код клиента” введен для однозначной идентификации клиента в рамках предприятия.

1. **Культуры** (*Код культуры, Название культуры, Класс, Базисная влажность, Базисная зерновая примесь, Базисная сорная примесь, Базисная стекловидность, Базисная натура, Базисная клейковина*)
2. **Вид поступления** (*Код вида поступления, Название вида поступления*).

Данная сущность представляет собой вид поступления зерна на предприятие. Зерно могут привозить клиенты для переработки на муку (давальческое зерно), а также предприятие может покупать зерно для собственных нужд (собственное зерно). Кроме этого, предприятие может брать участие в государственной программе по заготовке зерна (зерно госрезервов).

1. **Накладные** (*Номер\_накладной, Дата накладной, Код клиента, Номер\_автомашины, Код культуры, Код вида поступления, Масса брутто, Масса тары, Масса нетто, Номер склада, Номер лабораторного анализа*).

Сущность отражает поступление зерна на предприятие по накладных клиентов.

## Инфологическая и даталогическая модели базы данных

Инфологическая модель базы данных изображена на рис.17.

\*Номер накдладной

\*Дата накладной,

Код клиента,

Номер\_автомашины,

Код культуры,

Код вида поступления,

Масса брутто,

Масса тары,

Масса нетто,

Номер склада,

Номер лабораторного анализа

##### НАКЛАДНЫЕ

\*Код клиента,

Название клиента,

Тип клиента,

Банковские реквизиты, Юридический адрес,

Телефон

##### КЛИЕНТЫ

\*Код культуры,

Название культуры,

Класс культуры,

Базисная влажность,

Базисная зерновая примесь,

Базисная сорная примесь,

Базисная стекловидность, Базисная натура,

Базисная клейковина

##### КУЛЬТУРЫ

\*Код вида поступления

Название вида поступления

##### ПОСТУПЛЕНИЯ

Рис. 17. Инфологическая модель базы данных.

Звёздочками на инфологической модели показаны ключевые атрибуты.

В качестве даталогической модели базы данных была выбрана реляционная модель, поскольку именно реляционная модель является результатом более развитых представлений о формировании и ведении баз данных, на которые наложен строгий математический аппарат. Реляционные модели наиболее логично и наглядно отражают структуру хранимой информации и внутренних связей, что позволяет более полно анализировать структуру базы данных при разработке. Это привело к тому, что именно реляционные модели баз данных наиболее распространены в настоящее время и являются стандартом, на который переводятся все существовавшие ранее базы данных с иерархической и сетевой моделью. Ещё одним веским доводом в пользу выбора реляционной модели является тот факт, что подавляющее большинство предоставляемых средств для разработки баз данных ориентированны исключительно на реляционную модель. Кроме того, реляционные базы данных в последствии легче расширять и интегрировать, что является неотъемлемой частью дальнейшего развития баз данных, с увеличением возлагаемых на них задач.

Инфологическая модель базы данных легко отображается в реляционную даталогическую модель, используя описанные ранее правила по переводу. В результате получается семь таблиц реляционной базы данных, где каждая сущность напрямую отражается в отдельную таблицу, атрибуты каждой сущности становятся полями этой таблицы, а первичные ключи сущности становятся первичными ключами таблицы. На данном этапе необходимо также провести нормализацию полученных таблиц с целью устранения избыточности данных. Эта процедура в дальнейшем значительно облегчит усилия, которые будут затрачиваться на поддержании таблиц базы данных в целостном состоянии.

Правила нормализации, описанные ранее, предписывают для таких случаев заводить отдельную таблицу для каждого поля и хранить в ней все значения характерные только для одного поля. Я же принял решение свести значения для всех полей в одну таблицу и различать их не только по уникальному номеру-идентификатору, но и указывать таблицу, к которой это поле принадлежит, а также название самого поля. Выбор такого варианта оправдан тем, что таких полей в таблицах базы данных более десяти. Организация отдельных таблиц для каждого такого поля существенно усложнит структуру базы данных. Кроме того добавление новых атрибутов с подобным ограниченным числом значений потребует организовывать новую таблицу и затрату больших усилий по изменению интерфейса взаимодействия с конечным пользователем и исправлению программного кода, нежели от добавления этих значений в одну универсальную таблицу.

Конечно, в этом случае таблицы базы данных не будут до конца нормализованы, что накладывает некоторые требования на процедуры поддержания базы данных в целостном состоянии, но даёт возможность “безболезненных изменений” в программном коде, что может существенно сократить время разработки в дальнейшем. Процедуры по поддержанию целостности можно реализовать в программном коде, сделав их таким образом прозрачными для конечного пользователя.

Таким образом, в результате нормализации таблиц базы данных и проведённого анализа состава полей этих таблиц получилась ещё одна таблица, содержащая значения тех полей, которые принимают ограниченное число значений. Состав полей, первичный ключ (выделен жирным шрифтом с курсивом) и описание полей представлены в таблице 1.

Таблица 1.

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| ***Имя\_таблицы*** | Название таблицы к которой принадлежит поле |
| ***Имя\_поля*** | Название поля в этой таблице |
| ***Код*** | Уникальный номер (код) для значения данного поля |
| Значение | Значение для поля с данным кодом |

Такой состав таблиц позволяет выполнять все возложенные задачи, поскольку он выведен из инфологической модели, проектируемой исходя из требований конечных пользователей.

## Физическое описание модели

База данных организованна в популярном формате локальных баз данных dBase. Этот формат для организации реляционных баз данных довольно распространен, поскольку обладает очень развитой системой хранимых типов данных, возможностями индексирования полей, что позволяет получать доступ к данным за минимальное время, а также функциями по обеспечению ссылочной целостности между реляционными таблицами, что позволяет разработчику минимизировать временные затраты на создание базы данных, а конечному пользователю затраты на поддержание целостности хранимых данных и получения из базы данных самих хранимых данных. Поскольку базы данных dBase - реляционные базы данных, то запросы к данным осуществляются с помощью реляционного языка запросов SQL. Благодаря развитой системе определения ключевых полей и индексов при создании таблиц запросы будут выполняться с минимальными временными затратами. Хотя этот фактор для локальных баз данных не является ключевым, однако, при росте объема хранимых данных именно скорость выполнения запросов становиться решающим фактором при выборе формата баз данных.

Типы данных, хранимые в таблицах dBase, очень разнообразны. Это и символьные значения и разнообразные типы числовых значений, включающие целочисленные, вещественные, вещественные с плавающей запятой, числа в двоичном и двоично-десятичном формате, логические типы, специальные форматы для хранения значений даты, времени и денежных сумм, графические типы для хранения графических изображений в самых популярных форматах, а также строковые значения неограниченной длины (в том числе и форматированные в формате rtf) и даже типы поддерживаемые технологией OLE (Object Linking and Embedding) фирмы Micrоsoft. Такое разнообразие типов данных может отвечать даже самым изысканным задачам, которым призвана служить создаваемая база данных и без сомнения подходит для решения круга задач возложенного на базу данных о зерна зерноперерабатывающего предприятия.

База данных ZERNO представлена 4-мя таблицами (или по терминологии реляционных баз данных - 4-мя реляционными отношениями): Dorg, Dkul, Dtyp, Fnakl. Рассмотрим структуру каждой более подробно.

В таблице **Dorg** представлена информация о клиентах. Поля, их типы, и назначение представлены в таблице 2.

***Таблица 2.***

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Описание** |
| O\_Kod | Numeric[6] | Код клиента |
| *O\_Name* | Character[30] | Название |
| *O\_Typ* | Numeric[1] | Тип |
| *O\_Bank* | Character[45] | Банковские реквизиты |
| *O\_Address* | Character[45] | Адрес |
| *O\_Phone* | Numeric[10] | Номер телефона |

Первичным ключем таблицы является поле O\_Kod, которое однозначно определяет каждую запись в таблице.

В таблице **Dkul** представлена информация о культурах. Поля, их типы, и назначение представлены в таблице 3.

***Таблица 3.***

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Назначение** |
| ***K\_Kod*** | Numeric[3] | Код культуры |
| *K\_Nazva* | String[30] | Название культуры |
| *K\_Class* | Numeric[1] | Класс культуры |
| *K\_Volo* | Numeric[4.2] | Базисная влажность |
| *K\_Domi* | Numeric[4.2] | Базисная зерновая примесь |
| *K\_Soro* | Numeric[4.2] | Базисная сорная примесь |
| *K\_Natu* | Numeric[4.2] | Базисная натура |
| *K\_Sklo* | Numeric[4.2] | Базисная стекловидность |
| *K\_Kley* | Numeric[4.2] | Базисная клейковина |

Первичным ключем является поле K\_Kod, однозначно определяющее любую запись в таблице.

В таблице **Dtyp** представлена информация о видах поступления зерна.. Поля, их типы, и назначение представленны в таблице 4.

***Таблица 4.***

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Назначение** |
| ***T\_Kod*** | Numeric[2] | Код вида поступления |
| ***T\_Nazva*** | Character[25] | Название вида поступления |

В таблице **Fnakl** представлена информация о накладных, по которым поступает зерно. Поля, их типы, и назначение представлены в таблице 5.

***Таблица 5.***

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип поля** | **Назначение** |
| ***N\_Nomer*** | Character[6] | Номер накладной |
| ***N\_Date*** | Date | Дата накладной |
| ***N\_Avto*** | Character[7] | Номер автомашины |
| ***N\_Kodo*** | Numeric[6] | Код клиента |
| ***N\_Kodk*** | Numeric[3] | Код культуры |
| ***N\_Kodt*** | Numeric[2] | Код вида поступления |
| ***N\_NSkl*** | Numeric[1] | Номер склада |
| ***N\_Brutto*** | Numeric[9] | Масса брутто |
| ***N\_Tara*** | Numeric[9] | Масса тары |
| ***N\_Netto*** | Numeric[9] | Масса нетто |
| ***N\_Nana*** | Character[5] | Номер лабораторного анализа |

Ключевые поля данной таблицы – N\_Nomer и N\_Date. По полю N\_Nana данная таблица связана с базой данных лаборатории, где содержится информация о фактических качественных показателях принятого зерна. Структуру таблиц базы данных лаборатории я здесь приводить не буду, это отдельная задача. Кроме того, на базе данной таблицы (Fnakl) в задаче реализации готовой продукции формируются квитанции и накладные на выдачу муки клиентам.

## Програмная реализация

Все описанные таблицы, составляющие основу базы данных, функционируют в рамках созданной системы управления базой данных ”Zernot”. СУБД ”Zernot” создана средствами среды программирования Delphi 5.0.

В основу создания данной СУБД положен принцип экономии времени и усилий конечного пользователя, т.е. работников зерноперерабатывающего предприятия, предполагая, что машина берет на себя все рутинные функции управления и доступа к хранимым данным. Этот принцип прослеживался во всех моментах реализации данной СУБД, включая создание удобного интерфейса для работы конечных пользователей с этим программным продуктом, продуманной структурой реляционных таблиц, выбранным форматом баз данных выполняющие SQL-запросы за наиболее короткое время. Даже функции администрирования базы данных не требуют знакомства с теорией реляционной баз данных, СУБД самостоятельно тестирует находящиеся в базе данных записи и производит приведение базы данных к целостному состоянию. Пользователю остается согласиться со всей проделанной работой (или ее частью) или провести все самостоятельно. За сохранность введенных данных можно не беспокоиться, поскольку никакая информация, внесенная в базу данных не может быть удалена без подтверждения пользователя.

**Краткое описание программного проекта**

Проект Azagot, реализующий доступ к БД Zerno выполнен в виде библиотеки DLL. Состоит из двух форм MainForm (главная форма) и NaklForm (форма для ввода накладных). Из библиотеки Global.DLL в проект импортируются функции работы со справочниками клиентов, культур и видов поступления соответственно ShowDovOrg, ShowDovKul, ShowDovTyp.

Доступ к таблицам БД реализован с помощью компоненты DataModule. Тексты программ приведены в дополнении к дипломной работе. Поскольку база данных о зерне тесно связана с задачами реализации муки и лабораторией, в программном интерфейсе зарезервованы визуальные компоненты для взаимосвязи с этими задачами.

Чтобы попасть в окно ввода накладных, нужно на главной форме нажать кнопку с надписью “ЗЕРНО”. После этого откроется окно, через которое непосредственно реализуется работа с базой данный “ЗЕРНО”.

Для подсчета общего количества зерна за день в программе используется компонент TQuery, с помощью свойства SQL которого процедура подсчитывания очень быстро и эффективно выводит результат.

# Заключение

В данной дипломной работе детально рассмотрены фундаментальные понятия теории экономических информационных систем, дан обзор новейшим технологиям организации доступа к данным, построениям моделей и систем управления базами данных. На основе общепринятых критериев аргументирован выбор реляционной модели базы данных и среды Borland Delphi 5.0 для практической реализации программы для управления работой базы данных.

Процесс проектирования баз данных детально описан. Кроме этого, автором разработана база данных для зерноперерабатывающего предприятия, в которой хранится информация о качестве и количестве зерна, поступившего от клиентов на переработку. Данная система может быть успешно использована на предприятиях мукомельной промышленности.

Таким образом, автором, на разработанном конкретном примере, показан процесс оптимальной организации базы данных для наращивания экономической и статистической информации. Цель дипломной работы достигнута.

Материалы дипломной работы будут полезными для студентов, изучающих теорию экономических информационных систем, проектирование баз данных, а также программистам и аналитикам предприятий зерноперерабатывающей промышленности, которые занимаются разработкой информационных систем по управлению процессами на предприятии.

# Список литературы

1. Дж. Тельман, "Основы систем баз данных", Москва, Финансы и статистика', 1983г.
2. Дейт К., "Введение в системы баз данных", Москва, 'Hаука', 1980 г.
3. Когловский М.Р., "Технология баз данных на персональных ЭВМ", Москва, 'Финансы и статистика', 1992 г.
4. Шумаков П. В. “Delphi 3.0 и создание баз данных”. Москва 1997г.
5. Джон Матчо, Дэвид Р.Фолкнер. “Delphi” — пер. с англ. — М.:Бином, 1995г.
6. A.M.Епанешников., "Программирование в среде Delphi 2.0"
7. Дж. Мартин., "Организация баз данных в вычислительных системах" М: Мир 1978г.
8. С.М.Диго "Проектирование и использования баз данных". Москва: Финансы и статистика 1995.
9. Горев А., Ахаян Р., Макашарипов С. “Эффективная работа с СУБД”.СПб.:Питер, 1997.— 704 с.,ил.
10. Атре Ш. Структурный подход к организации баз данных. – М.: Финансы и статистика, 1983. – 320 с.
11. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. – М.: Финансы и статистика, 1989. – 351 с.
12. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. -М.: Мир, 2001. – 252 с.
13. Кириллов В.В. Структуризованный язык запросов (SQL). – СПб.: ИТМО, 1994. – 80 с.
14. Мейер М. Теория реляционных баз данных. – М.: Мир, 1998. – 608 с.
15. Тиори Т., Фрай Дж. Проектирование структур баз данных. В 2 кн., – М.: Мир, 1999. Кн. 1. – 287 с.: Кн. 2. – 320 с.
16. Цикритизис Д., Лоховски Ф. Модели данных. – М.: Финансы и статистика, 1985. – 344 с.
17. Paradox for Windows: Практическое руководство. Под редакцией Оспищева Д. А. Издательство АОЗ’ "Алевар", 1993.
18. Брябрин В.М., "Программное обеспечение персональных ЭВМ", Москва, 'Hаука', 1989 г.
19. Шафрин Ю.А. “Основы компьютерной технологии”. М., 1998
20. “Кибернетические диалоговые системы”, И.П.Кузнецов.
21. “Рекоммендации по общепользовательскому интерфейсу”, Microsoft, редакция 2000г.

22. Тейксейра С., Пачеко К. Delphi 5. Руководство разоработчика.Москва-Санкт-Петербург-Киев, 2000г.