

1. Определение понятия информация и информационное обеспечение

Информация (от лат. informatio – осведомление, разъяснение, изложение) – это сведения (сообщения, данные) независимо от формы их представления.

Наиболее подходящим и простым для изучаемого курса можно считать следующее определение: **информация** – это совокупность данных, зафиксированных на материальном носителе, сохранённых и распространённых во времени и пространстве.

Основные виды информации по форме представления и способам кодирования и хранения – это графическая или изобразительная; звуковая; текстовая; числовая; видеоинформация.

Существуют также виды информации, для которых до сих пор не изобретено способов кодирования и хранения – это тактильная информация, передаваемая ощущениями, органолептическая, передаваемая запахами и вкусами и др.

Информационное обеспечение в широком смысле имеет три значения:

- 1) обеспечение фактическими данными управленческих структур;
- 2) использование информационных данных для автоматизированных систем управления (АСУ);
- 3) использование информации для обеспечения деятельности различных потребителей (организаций, ученых, художников, писателей, журналистов и т. д.).

Обеспечение информацией управленческих структур (государства, корпораций, организаций) производится, прежде всего, за счет организаций, специально занимающихся сбором данных (государственные органы статистики, научные центры различного типа).

Вторым важным направлением ИО является формирование информационных данных для автоматизированных систем управления. Вводимая в БД АСУ информация является необходимым элементом работы всей системы, без которой невозможно ее математическое, техническое, организационно-правовое функционирование. Информация, вводимая в систему, ее предмашинная обработка – основа современных автоматизированных информационных систем (АИС).

Третье направление ИО связано с удовлетворением информационных запросов потребителей самого разнообразного типа: как организаций, учреждений, так и отдельных лиц. В этом случае в качестве ИО выступают не только статистические данные, данные социологических опросов, данные архивов и других официальных учреждений, но и такие типы информации, как книжные и журнальные публикации, научные отчеты, диссертации и пр. Наиболее распространенной формой этого типа ИО ранее являлись библиотеки, но в современных условиях все большее значение приобретают службы и центры анализа информации.

Информационное обеспечение – это совокупность единой системы классификации и кодирования информации, унифицированных систем

документации, схем информационных потоков, циркулирующих в организации, а также методология построения баз данных.

ИО информационных систем (ИС) является средством для решения следующих задач:

- однозначного и экономичного представления информации в системе (на основе кодирования объектов);
- организации процедур анализа и обработки информации с учетом характера связей между объектами (на основе классификации объектов);
- организации взаимодействия пользователей с системой (на основе экранных форм ввода-вывода данных);
- обеспечения эффективного использования информации в контуре управления деятельностью объекта автоматизации (на основе унифицированной системы документации).

ИО ИС включает два комплекса: немашинное информационное обеспечение (классификаторы технико-экономической информации, документы, методические инструктивные материалы) и внутримашинное информационное обеспечение (макеты/экранные формы для ввода первичных данных в ЭВМ или вывода результатной информации, структуры информационной базы: входных, выходных файлов, базы данных).

К ИО предъявляются следующие *общие требования*:

- ИО должно быть достаточным для поддержания всех автоматизируемых функций объекта;
- для кодирования информации должны использоваться принятые у заказчика классификаторы;
- для кодирования входной и выходной информации, которая используется на высшем уровне управления, должны быть использованы классификаторы этого уровня;
- должна быть обеспечена совместимость с ИО систем, взаимодействующих с разрабатываемой системой;
- формы документов должны отвечать требованиям корпоративных стандартов заказчика (или унифицированной системы документации);
- структура документов и экранных форм должна соответствовать характеристикам терминалов на рабочих местах конечных пользователей;
- графики формирования и содержание информационных сообщений, а также используемые аббревиатуры должны быть общеприняты в предметной области заказчика и согласованы с ним;
- в ИС должны быть предусмотрены средства контроля входной и выходной информации, обновления данных в информационных массивах, контроля целостности информационной базы, защиты от несанкционированного доступа.

Основными источниками информации для ИО служат:

- информационно-справочная информация;
- отчетные формы документов;
- информация вышестоящих органов;

- бухгалтерская информация.

2.Определение, общая структура и классификация информационных систем

Информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса.

Классическими примерами ИС являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

Другими словами ИС требует создания в памяти ЭВМ динамически обновляемой модели внешнего мира с использованием единого хранилища – базы данных.

Структура любой ИС может быть представлена совокупностью обеспечивающих подсистем

ИС можно классифицировать по целому ряду различных признаков .

По типу хранимых данных ИС делятся на фактографические и документальные. Фактографические системы предназначены для хранения и обработки структурированных данных в виде чисел и текстов. Над такими данными можно выполнять различные операции. В документальных системах информация представлена в виде документов, состоящих из наименований, описаний, рефератов и текстов. Поиск по неструктурированным данным осуществляется с использованием семантических признаков. Отобранные документы предоставляются пользователю

По степени автоматизации информационных процессов ИС делятся на ручные, автоматические и автоматизированные.

Ручные ИС характеризуются отсутствием технических средств переработки информации и выполнением всех рутинных операций человеком.

В автоматических ИС все операции по обработке информации выполняются без участия человека.

Автоматизированные ИС предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль в выполнении рутинных операций обработки данных отводится компьютеру.

В зависимости от характера обработки данных ИС делятся на информационно-поисковые и информационно-решающие.

Информационно-поисковые системы осуществляют ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных. Например, это ИС библиотечного обслуживания, резервирования и продажи билетов на транспорте, бронирования мест в гостиницах и др.

Информационно-решающие системы осуществляют, кроме того, операции переработки информации по определенному алгоритму. По характеру использования выходной информации такие системы принято делить на управляющие и советующие.

Выходная информация управляющих ИС непосредственно трансформируется в принимаемые человеком решения. Для этих систем характерны задачи расчетного характера и обработка больших объемов данных. Например, это ИС планирования производства или заказов, бухгалтерского учета и др.

Советующие ИС вырабатывают информацию, которая принимается человеком к сведению и учитывается при формировании управленческих решений, а не инициирует конкретные действия. Эти системы имитируют интеллектуальные процессы обработки знаний, а не данных. Например, экспертные системы.

В зависимости от сферы применения различают:

- ИС организационного управления, которые предназначены для автоматизации функций управленческого персонала, как промышленных предприятий, так и непромышленных объектов (гостиниц, банков, магазинов и др.).

Основными функциями подобных систем являются: оперативный контроль и регулирование, оперативный учет и анализ, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом, снабжением и другие экономические и организационные задачи.

- ИС управления технологическими процессами (АСУ ТП) – служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно предусматривается наличие развитых средств измерения параметров ТП (температуры, давления, химического состава и т.п.), процедур контроля допустимости значений параметров и регулирования ТП.

- ИС автоматизированного проектирования (САПР) – предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями подобных систем являются: инженерные расчеты, создание графической документации (чертежей, схем, планов), создание проектной документации, моделирование проектируемых объектов.

- Интегрированные (корпоративные) ИС – используются для автоматизации всех функций фирмы и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей (подсистем), работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности.

3.Определение и классификация БД

База данных – это организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

База данных – это совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей организации.

Наиболее часто для понимания этого термина используются следующие отличительные признаки:

- БД хранится и обрабатывается в вычислительной системе. Таким образом, любые некомпьютерные хранилища информации (архивы, библиотеки, картотеки и т. п.) БД не являются.
- Данные в БД логически структурированы (систематизированы) с целью обеспечения возможности их эффективного поиска и обработки в вычислительной системе. Структурированность подразумевает явное выделение составных частей (элементов), связей между ними, а также типизацию элементов и связей, при которой с типом элемента (связи) соотносится определённая семантика и допустимые операции.
- БД всегда включает метаданные, описывающие логическую структуру самой БД в формальном виде (в соответствии с некоторой моделью).

Существует огромное количество разновидностей БД, отличающихся по различным критериям.

Признаки классификации.

В зависимости от модели данных различают следующие БД:

- иерархические;
- сетевые;
- реляционные;
- объектно-реляционные;
- объектно-ориентированные (объектные).

Иногда иерархические и сетевые модели данных называют дореляционными, а объектно-реляционные и объектно-ориентированные – постреляционными.

Существует классификация БД **по технологии хранения** данных в памяти компьютера:

- БД во вторичной памяти (традиционные);
- БД в оперативной памяти (in-memory databases);
- БД в третичной памяти (tertiary databases).

Третичной памятью называют съёмные носители, которые монтируются при необходимости роботизированными механизмами после того, как с помощью каталога БД было установлено, какие именно данные нужны и где именно они хранятся.

По степени распределённости хранимых данных различают: централизованные (сосредоточенные) и распределённые БД.

По характеру содержимого БД бывают: географические, исторические, научные, мультимедийные и т.д.

4. Определение и базовые функции СУБД

СУБД – это специализированная программа (комплекс программ), предназначенная для организации и ведения БД.

Другими словами, СУБД называют прикладную информационную систему (комплекс программных средств), опирающуюся на некоторую систему управления данными, и обладающую следующим минимальным набором функций:

1. управление данными во внешней памяти;
2. управление буферами оперативной памяти;
3. управление транзакциями;
4. журнализация;
5. поддержка языков БД.

Перечисленные функции являются базовыми для любой СУБД,

Управление данными во внешней памяти. Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для ускорения доступа к данным используются индексы. Некоторые СУБД активно используют возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но в любом случае пользователи не обязаны знать, использует ли СУБД файловую систему, и если использует, то, как организованы файлы, для чего СУБД, как правило, поддерживает собственную систему именования объектов БД.

Управление буферами оперативной памяти. СУБД обычно работают с БД значительного размера и, если при обращении к любому элементу данных, будет производиться обмен с внешней памятью, то система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. Если операционная система (ОС) производит общесистемную буферизацию (например, ОС UNIX), то этого недостаточно, поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной системой замены буферов.

Отметим, что существует отдельное направление в развитии СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Оно основано на условии, что объем оперативной памяти компьютеров настолько велик, что позволяет не беспокоиться о буферизации.

Управление транзакциями. Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменения БД во внешней памяти, либо ни одно из изменений никак не отражается на состоянии БД.

Каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может ощущать себя единственным.

С управлением транзакциями в многопользовательском режиме работы связаны понятия сериализации транзакций и сериального плана выполнения смеси транзакций.

Под сериализацией параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций – это такой план, который приводит к сериализации транзакций. Понятно, что если удастся добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, присутствие других транзакций будет незаметно (кроме некоторого замедления работы по сравнению с однопользовательским режимом).

Журнализация. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти, т.е. СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризующиеся потерей информации на носителях внешней памяти.

В любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал – это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда существуют две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. Во всех случаях придерживается стратегия "упреждающей" записи в журнал (протокол Write Ahead Log – WAL). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем сам измененный объект попадет во внешнюю память основной части БД. Таким образом, если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД практически после любого сбоя.

Поддержка языков БД. Для работы с БД используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков, таких как язык определения схемы данных (DDL – Data Definition Language) и язык манипулирования данными (DML – Data Manipulation Language).

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и заканчивая обеспечением базового пользовательского интерфейса. Стандартным языком реляционных СУБД является язык SQL (Structured Query Language). SQL сочетает средства DDL и DML, т.е. позволяет определять

схему реляционной БД и манипулировать данными. Язык SQL содержит также операторы для определения ограничений целостности БД, позволяет создавать представления, фактически являющиеся хранимыми в БД запросами, индексы, домены, синонимы, последовательности и другие, хранимые в схеме пользователя, объекты. Кроме этого, позволяет решать вопросы авторизации доступа к объектам БД и выделения полномочий (привилегий) на работу с ними, управляет работой транзакций – эти специальные средства иногда определяют как третью составную часть SQL (в дополнение к DDL и DML) и называют языком управления данными (DCL - Data Control Language).

5. Структурная схема типовой СУБД



Основной компонент – **контроллер БД** взаимодействует с прикладными программами пользователей и их запросами. Он принимает запрос и проверяет внешние и концептуальные схемы для определения записей, которые необходимы для удовлетворения требований запроса, затем вызывает контроллер файлов для выполнения поступившего запроса, т.е. извлечения необходимых данных.

Перечислим основные программные компоненты, входящие в состав контроллера БД.

1. Модуль прав доступа проверяет наличие у пользователя полномочий для выполнения затребованной операции.
2. Процессор команд получает после проверки полномочий пользователя управление для непосредственного выполнения операции.
3. Средства контроля целостности в случае выполнения операций, которые изменяют содержимое БД, осуществляют проверку того, удовлетворяет ли затребованная операция всем установленным ограничениям поддержания целостности данных.
4. Оптимизатор запросов определяет оптимальную по скорости или затрате других ресурсов стратегию выполнения запроса.
5. Контроллер транзакций осуществляет требуемую обработку операций, поступающих в процессе выполнения транзакций.
6. Планировщик отвечает за бесконфликтное выполнение параллельных операций с БД. Он управляет относительным порядком выполнения операций, затребованных в отдельных транзакциях.

7. Контроллер восстановления гарантирует восстановление БД до непротиворечивого состояния при возникновении сбоев. В частности, отвечает за фиксацию и отмену результатов выполнения транзакций.

8. Контроллер буферов осуществляет перенос данных между оперативной памятью и вторичным запоминающим устройством - например, жестким диском или магнитной лентой. Контроллер восстановления и контроллер буферов иногда в совокупности называют контроллером данных.

Процессор запросов преобразует SQL запросы в последовательность низкоуровневых инструкций для контроллера БД.

Контроллер файлов манипулирует предназначенными для хранения данных файлами и отвечает за распределение доступного дискового пространства. Он создает и поддерживает список структур и индексов, определенных во внутренней схеме. Если используются хешированные файлы, то в его обязанности входит и вызов функций хеширования для генерации адресов записей. Однако контроллер файлов не управляет физическим вводом и выводом данных непосредственно, а лишь передает запросы соответствующим методам доступа, которые считывают данные в системные буферы или записывают их оттуда на диск.

Препроцессор DML преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка. Для генерации соответствующего кода препроцессор языка DML должен взаимодействовать с процессором запросов.

Компилятор DDL преобразует DDL-команды в набор таблиц, содержащих метаданные. Затем эти таблицы сохраняются в системном каталоге, а управляющая информация - в заголовках файлов с данными.

Контроллер системного каталога (словаря данных) управляет доступом к системному каталогу и обеспечивает работу с ним.

6. Типовые функции СУБД

1. Хранение, извлечение и обновление данных. СУБД должна предоставлять пользователям возможность сохранять, извлекать и обновлять данные - это фундаментальная функция любой СУБД. Способ реализации этой функции должен позволять скрывать от конечного пользователя внутренние детали физической реализации системы (например, файловую организацию или используемые структуры хранения).

2. Ведение системного каталога. Системный каталог, или словарь данных, является централизованным хранилищем информации, описывающей данные в БД (метаданные). Предполагается, что каталог доступен как пользователям, так и системным службам СУБД. В зависимости от типа используемой СУБД количество информации и способы ее применения могут варьироваться. Обычно в системном каталоге хранятся следующие сведения: имена, типы и размеры элементов данных; имена связей; накладываемые на данные ограничения поддержания целостности; имена санкционированных пользователей; внешняя, концептуальная и внутренняя схемы и связи между ними; статистические данные, например частота транзакций и счетчики обращений к объектам БД.

Основными преимуществами ведения системного каталога являются:

- контроль доступа к его данным, как к любому другому ресурсу БД;
- определение смысла хранимых данных, что помогает другим пользователям понять их назначение;
- легкое обнаружение избыточности и противоречивости описания отдельных элементов данных;
- протоколирование внесенных в БД изменений;
- дополнительное усиление мер обеспечения безопасности;
- новые возможности организации поддержки целостности данных;
- аудит сохраняемой информации.

3. Поддержка транзакций. СУБД должна иметь механизм, который гарантирует выполнение либо всех операций данной транзакции, либо ни одной из них. Примерами простых транзакций могут служить операции добавления, удаления или обновления строк в БД. Если во время выполнения транзакции произойдет сбой, БД попадает в противоречивое состояние, поскольку некоторые изменения уже будут внесены, а остальные - еще нет. Поэтому все частичные изменения должны быть отменены для возвращения БД в прежнее, непротиворечивое состояние.

4. Сервисы управления параллельностью. СУБД должна иметь механизм, который гарантирует корректное обновление БД при параллельном выполнении операций изменения данных многими пользователями. При этом параллельный доступ сравнительно просто организовать, если все пользователи выполняют только чтение данных, поскольку в этом случае они не могут помешать друг другу. Однако когда несколько пользователей одновременно получают доступ к БД, конфликт с нежелательными последствиями легко может возникнуть, например, если хотя бы один из них

попытается изменить данные. СУБД должна гарантировать, что при одновременном доступе к БД многих пользователей подобных конфликтов при изменении не произойдет.

5. Сервисы восстановления. При обсуждении поддержки транзакций упоминалось, что при сбое транзакции БД должна быть возвращена в непротиворечивое состояние, что должно гарантироваться возможностями СУБД, в частности, ведением журнала транзакций.

6. Сервисы контроля доступа к данным. СУБД должна иметь механизм, гарантирующий возможность доступа к БД только санкционированных пользователей. Термин "безопасность" относится к защите БД от преднамеренного или случайного несанкционированного доступа.

7. Поддержка обмена данными. СУБД должна обладать способностью к интеграции с коммуникационным программным обеспечением с целью организации доступа удаленных пользователей к централизованной БД (в рамках системы распределенной обработки).

8. Службы поддержки целостности данных. СУБД должна обладать инструментами контроля того, чтобы данные и их изменения соответствовали заданным правилам. Целостность базы данных означает корректность и непротиворечивость хранимых данных и выражается в виде ограничений или правил сохранения непротиворечивости данных, которые не должны нарушаться в базе.

9. Службы поддержки независимости от данных. СУБД должна обладать инструментами поддержки независимости программ от структуры БД. Обычно это достигается за счет реализации механизма поддержки представлений или подсем. Физическая независимость от данных достигается довольно просто, так как обычно имеется несколько типов допустимых изменений физических характеристик БД, которые никак не влияют на представления. Как правило, система легко адаптируется к добавлению нового объекта, атрибута или связи, но не к их удалению. В некоторых системах вообще запрещается вносить любые изменения в уже существующие компоненты логической схемы.

10. Вспомогательные службы. СУБД должна предоставлять некоторый набор вспомогательных служб (утилит) которые обычно предназначены для эффективного администрирования БД. Одни утилиты работают на внешнем уровне и могут быть созданы администратором БД, тогда как другие функционируют на внутреннем уровне системы и потому должны быть предоставлены самим разработчиком СУБД. Примерами подобных утилит являются:

- утилиты импортирования и экспортирования, предназначенные для загрузки и выгрузки БД в плоские файлы;
- средства мониторинга, предназначенные для отслеживания характеристик функционирования и использования БД;
- программы статистического анализа, позволяющие оценить производительность или степень использования БД;

- инструменты реорганизации индексов, предназначенные для перестройки индексов и обработки случаев их переполнения;
- инструменты сбора мусора и перераспределения памяти для физического устранения удаленных записей с запоминающих устройств, объединения освобожденного пространства и перераспределения памяти в случае необходимости.

7.Преимущества и недостатки СУБД(по сравнению с файловыми системами)

Рассмотрим основные преимущества:

1. Контроль избыточности данных. Традиционные файловые системы неэкономно расходуют внешнюю память, сохраняя одни и те же данные в нескольких файлах. При использовании СУБД, наоборот, предпринимается попытка исключить избыточность данных за счет интеграции файлов, чтобы избежать хранения нескольких копий одного и того же элемента информации. Однако полностью избыточность информации в БД не исключается, а лишь контролируется ее степень. В одних случаях ключевые элементы данных необходимо дублировать для моделирования связей, а в других случаях некоторые данные потребуются дублировать из соображений повышения производительности системы.

2. Непротиворечивость данных. Устранение избыточности данных или контроль над ней позволяет сократить риск возникновения противоречивых состояний. Если элемент данных хранится в базе только в одном экземпляре, то для изменения его значения потребуется выполнить только одну операцию обновления, причем новое значение станет доступным сразу всем пользователям БД. Если элемент данных хранится в нескольких экземплярах, то можно следить за тем, чтобы копии не противоречили друг другу.

3. Больше полезной информации при одинаковом объеме хранимых данных. Благодаря интеграции, на основе тех же данных можно получать дополнительную информацию, т.е. выполняется основной принцип холизма - целое всегда есть нечто большее, чем простая сумма его частей.

4. Совместное использование данных. Файлы обычно принадлежат отдельным лицам, которые используют их в своей работе. БД принадлежит группам пользователей или всей организации в целом и может совместно использоваться всеми зарегистрированными пользователями. При такой организации работы большее количество пользователей может работать с большим объемом данных и создавать новые приложения на основе уже существующей в БД информации.

5. Поддержка целостности данных. Целостность БД означает корректность и непротиворечивость хранимых в ней данных. Целостность обычно описывается с помощью ограничений, т.е. правил поддержки непротиворечивости, которые не должны нарушаться в БД. Ограничения можно применять к элементам данных внутри одной записи или к связям между записями.

6. Повышенная безопасность. Безопасность БД заключается в защите данных от несанкционированного доступа со стороны пользователей. Без привлечения соответствующих мер безопасности интегрированные данные становятся более уязвимыми, чем данные в файловой системе. Система обеспечения безопасности может быть выражена в форме учетных имен и паролей для идентификации пользователей, которые зарегистрированы в этой БД. Доступ к данным со стороны зарегистрированного пользователя может

быть ограничен только некоторыми операциями (извлечением, вставкой, обновлением и удалением).

7. Применение стандартов. Интеграция позволяет определять и применять необходимые стандарты. Например, стандарты предприятия, государственные и международные стандарты могут регламентировать формат данных при обмене ими между системами, соглашения об именах, форму представления документации, процедуры обновления и правила доступа.

8. Повышение экономической эффективности с ростом масштабов системы. Комбинируя все рабочие данные в одной БД и создавая набор приложений, которые работают с одним источником данных, можно добиться существенной экономии средств.

9. Повышение доступности данных и их готовности к работе. Данные в результате интеграции становятся непосредственно доступными конечным пользователям. Потенциально это повышает функциональность системы, что, например, может быть использовано для более качественного обслуживания конечных пользователей. Во многих СУБД предусмотрены языки запросов или инструменты для создания отчетов, которые позволяют пользователям формулировать непредусмотренные заранее запросы и почти немедленно получать требуемую информацию на своих терминалах, не прибегая к помощи программиста.

10. Улучшение показателей производительности. На базовом уровне СУБД обеспечивает все низкоуровневые процедуры работы с файлами, которые обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, необходимых пользователям функций, не заботясь о подробностях их выполнения на более низком уровне.

11. Упрощение сопровождения системы за счет независимости от данных. В файловых системах описания данных и логика доступа к данным встроены в каждое приложение, что делает программы зависимыми от данных. В СУБД описания данных отделены от приложений, а потому приложения защищены от изменений в описаниях данных.

12. Улучшенное управление параллельностью. В файловых системах при одновременном доступе к одному и тому же файлу двух пользователей может возникнуть конфликт двух запросов, результатом которого будет потеря информации или утрата ее целостности. В СУБД предусмотрена возможность параллельного доступа к БД и гарантируется отсутствие подобных проблем.

13. Развитые службы резервного копирования и восстановления. Ответственность за обеспечение защиты данных от сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. В СУБД предусмотрены средства защиты или хотя бы сокращения объема потерь информации при возникновении различных сбоев.

К недостаткам относятся:

1. Сложность. Обеспечение функциональности, которой должна обладать каждая хорошая СУБД, сопровождается ее значительным

усложнением. Чтобы воспользоваться всеми преимуществами СУБД, проектировщики и разработчики БД, администраторы, а также конечные пользователи должны хорошо понимать ее функциональные возможности. Непонимание принципов работы системы может привести к неудачным результатам работы.

2. Размер программного обеспечения. Сложность и широта функциональных возможностей приводит к тому, что СУБД становится программным продуктом, который может занимать много места на диске и требовать большого объема оперативной памяти для эффективной работы.

3. Стоимость. В зависимости от имеющейся вычислительной среды и требуемых функциональных возможностей, стоимость СУБД может варьироваться в очень широких пределах. Кроме того, следует учесть ежегодные расходы на сопровождение системы, которые составляют некоторый процент от ее общей стоимости.

4. Дополнительные затраты на аппаратное обеспечение. Для удовлетворения требований, предъявляемых к дисковым накопителям со стороны СУБД и БД, может понадобиться приобрести дополнительные устройства хранения информации. Более того, для достижения требуемой производительности может понадобиться более мощный компьютер, который, возможно, будет работать только с конкретной СУБД.

5. Затраты на преобразование приложений. В некоторых ситуациях стоимость СУБД и дополнительного аппаратного обеспечения может оказаться несущественной по сравнению со стоимостью преобразования существующих приложений для работы с новой СУБД и новым аппаратным обеспечением. Эти затраты также включают стоимость подготовки персонала для работы с новой системой и оплату услуг специалистов, которые будут оказывать помощь в преобразовании и запуске новой системы.

6. Производительность. Обычно файловая система создается для конкретных специализированных приложений, поэтому ее производительность может быть весьма высока. СУБД предназначена для решения более общих задач и обслуживания сразу нескольких приложений, а не какого-то одного. В результате многие приложения в новой среде будут работать не так быстро, как прежде.

7. Более серьезные последствия при выходе системы из строя. Централизация ресурсов повышает уязвимость системы. Поскольку работа всех пользователей и приложений зависит от готовности к работе СУБД, выход из строя одного из ее компонентов может привести к полному прекращению всей работы предприятия и его филиалов.

8. Понятие структуры данных, классификация типов

Данные, хранящиеся непосредственно в памяти ЭВМ, представляют собой совокупность нулей и единиц (битов). Биты объединяются в последовательности. Каждому участку оперативной памяти, который может вместить один байт или слово, присваивается порядковый номер (адрес).

Под **структурой данных** в общем случае понимают множество элементов данных и множество связей между ними.

Понятие **физическая структура данных** отражает способ физического представления данных в памяти машины и называется еще структурой хранения, внутренней структурой или структурой памяти.

Любые данные могут быть отнесены к одному из двух типов: основному (простому, базовому, примитивному), форма представления которого определяется архитектурой ЭВМ, или сложному (структурированному, составному), который создается пользователем для решения конкретных задач.

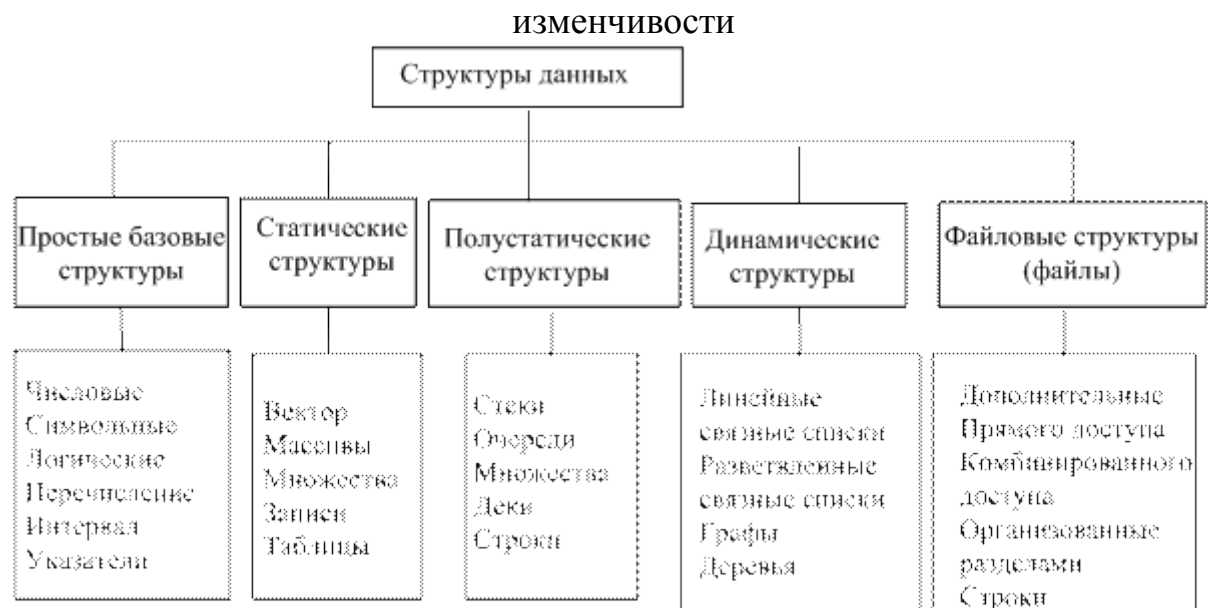
Данные простого типа – это символы, числа и т.п. элементы, дальнейшее дробление которых на составные части, большие, чем биты не имеет смысла.

С точки зрения физической структуры важным является то обстоятельство, что в данной системе программирования всегда можно заранее сказать, каков будет размер данного простого типа и какова структура его размещения в памяти. С логической точки зрения простые данные являются неделимыми единицами.

Сложными называются такие структуры данных, составными частями которых являются другие структуры данных – простые или в свою очередь интегрированные.

В зависимости от отсутствия или наличия явно заданных связей между элементами данных следует различать несвязные структуры (векторы, массивы, строки, стеки, очереди) и связные структуры (связные списки).

Важный признак структуры данных – ее изменчивость – изменение числа элементов и (или) связей между элементами структуры. По признаку



При описании любых используемых констант, переменных, полей таблиц и функций явно или неявно указывается их тип. Выделим следующие категории типов:

1. Встроенные типы данных, т.е. типы, предопределенные в языке программирования или языке БД (SQL). Обычно в состав встроенных типов данных включаются такие типы, операции над значениями которых поддерживаются командами компьютеров. В традиционный набор встроенных типов обычно входят следующие:

- символьный тип CHARACTER (или CHAR) – это набор печатных символов из алфавита, зафиксированного в описании языка.
- логический тип BOOLEAN обычно содержит два значения - TRUE (истина) и FALSE (ложь). Несмотря на то, что для хранения значений этого типа теоретически достаточно одного бита, обычно переменные занимают один байт памяти
- тип целых чисел INTEGER в общем случае включает подмножество целых чисел, определяемое числом разрядов, которое используется для внутреннего представления значений. При определении типа целых чисел обычно стремятся к тому, чтобы множество его значений было симметрично относительно нуля

Наряду со знаковыми целыми типами в языках часто поддерживаются беззнаковые целые. Для поддержки численных вычислений в языках обычно специфицируется встроенный тип чисел с плавающей точкой с базовым названием REAL или FLOAT.

2. Уточняемые типы данных – это типы, которые определены на основе встроенного типа данных, значения которого каким-либо образом упорядочены или ограничены.

Основной проблемой уточняемых типов является потребность в динамическом контроле значений, формируемых при вычислении выражений и возвращаемых функциями.

3. Перечисляемые типы данных представляют явно определяемые целые типы с конечным числом именованных значений. Это очень простой и легко реализуемый механизм, часто являющийся очень полезным.

Обычно для любого перечисляемого типа предопределяются операции получения значения по его номеру и получения номера по значению. Кроме того, для перечисляемого типа предопределяются операции сравнения и получения следующего и предыдущего значения.

4. Конструируемые типы (составные) обладают той особенностью, что в языке предопределены средства спецификации таких типов и некоторый набор операций, дающих возможность доступа к компонентам составных значений. Любое значение конструируемого типа состоит из значений одного или нескольких других типов.

Наиболее распространенные разновидности конструируемых типов – это типы массивов, записей и множеств.

Массивы. Базовым типом массива может быть любой встроенный или определенный тип, в том числе и тип массива, тогда говорят о многомерных массивах или матрицах.

Записи. Позволяют определять и использовать нерегулярные структуры данных, эл-ты которых могут относиться к разным встроенным или явно определенным типам данных.

Множества. С использованием механизма множеств можно писать лаконичные и красивые программы, но нужно отдавать себе отчет в том, что для эффективной реализации множеств требуются серьезные ограничения их мощности.

5. Указательные типы дают возможность работы с типизированными множествами абстрактных адресов переменных, содержащих значения некоторого типа.

Понятие указателя в языках программирования является абстракцией понятия машинного адреса.

6. Абстрактные (определяемые пользователями) типы данных. Наличие перечисляемых, уточняемых и конструируемых типов данных в сочетании со средствами выделения динамической памяти позволяет конструировать и использовать структуры данных, достаточные для создания сложных программ

9. Типы и структуры данных, применяемые в реляционных и объектно-реляционных базах данных

1) Реляционные БД

Рассмотрим особенности использования типов и структур данных в СУБД с классической реляционной моделью данных. Одним из базовых

свойств этой модели является атомарность значений в каждом из столбцов таблиц, т.е. значения должны принадлежать к одному из встроенных типов, поддерживаемых СУБД.

Практически все современные реляционные СУБД опираются на стандартный язык БД - SQL и поддерживают встроенные типы данных, специфицированные в этом языке [5].

Значение любого типа является примитивным в том смысле, что в соответствии со стандартом оно не может быть логически разбито на другие значения. Значения могут быть определенными или неопределенными. Неопределенное значение – это зависящее от реализации значение, которое гарантированно отлично от любого определенного значения соответствующего типа. Можно считать, что имеется всего одно неопределенное значение, входящее в любой тип данных языка SQL. Для неопределенного значения отсутствует представляющий его литерал, хотя в некоторых случаях используется ключевое слово NULL для выражения того, что желательно именно неопределенное значение.

В стандарте SQL обычно определены типы данных, обозначаемые следующими ключевыми словами: CHARACTER, CHARACTER VARYING, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, DATE, TIME, TIMESTAMP и INTERVAL.

Типы данных CHARACTER и CHARACTER VARYING называются типами данных символьных строк; типы данных BIT и BIT VARYING - типами данных битовых строк. Типы данных символьных и битовых строк совместно называются строчными типами данных, а значения строчных типов называются строками.

Типы данных NUMERIC, DECIMAL, INTEGER и SMALLINT совместно называются типами данных точных чисел. Типы данных FLOAT, REAL и DOUBLE PRECISION называются типами данных приближенных чисел. Типы данных точных чисел и типы данных приближенных чисел называются числовыми типами. Значения числовых типов называются числами.

Типы данных DATE, TIME и TIMESTAMP называются типами даты-времени. Значения типов даты-времени называются "дата-время".

Тип данных INTERVAL называется интервальным типом.

Поскольку основным способом использования языка SQL при создании прикладных информационных систем является встраивание операторов SQL в программы, написанные на традиционных языках программирования, необходимо для всех потенциально используемых языков программирования

иметь правила соответствия встроенных типов SQL встроенным типам соответствующих языков.

Важным понятием реляционных БД в стандарте языка SQL является понятие домена.

Домен – это именованное множество значений некоторого встроенного типа, ограниченное условием, задаваемым при определении домена. Условие определяет вхождение значений базового типа во множество значений домена. В некотором смысле можно считать понятие домена расширением понятия ограниченного типа в языках программирования.

Значения всех упомянутых типов (и определенных на них доменов) имеют фиксированную или, по крайней мере, ограниченную длину. Даже для типов CHARACTER VARYING и BIT VARYING длина допустимого значения обычно ограничена размером страниц внешней памяти, используемых СУБД для хранения БД. В связи с потребностями современных приложений (географических, мультимедийных и т.д.) в большинстве СУБД поддерживается дополнительный, не специфицированный в стандарте SQL псевдотип данных BLOB (Binary Large Object). Значения этого типа представляют собой последовательности байт, на которые на уровне СУБД не накладывается более сложная структура и длина которых практически не ограничена (в 32-разрядных архитектурах - до 2 Гб). Традиционные СУБД обеспечивают очень примитивный набор операций со столбцами типа BLOB - выбрать значение столбца в основную память или в файл и занести в столбец значение из основной памяти или файла.

2) Объектно-реляционные БД

РБД обладают рядом ограничений, которые затрудняют их использование в приложениях, требующих богатого типового окружения. Это относится и к категорическому требованию использовать в столбцах таблиц только атомарные значения встроенных типов, и к невозможности определить новые типы данных (возможно, с атомарными значениями) с дополнительными или переопределенными операциями. Понятно, что ослабление этих ограничений приводит к потребности существенного пересмотра архитектуры серверных продуктов БД, в частности расширению системы типов и связанных с этим структур данных [5].

Строчные типы данных. Одним из недостатков классического реляционного подхода к построению БД является то, что при определении схемы (структуры) таблицы ее имя одновременно становится именем самой таблицы, т.е. отсутствует возможность отдельно определить именованную схему таблицы, а затем – одну или несколько таблиц с той же схемой. Для

устранения этого недостатка (а также получения некоторых дополнительных преимуществ) в объектно-реляционных БД появилось понятие строчного типа.

Фактически, строчный тип – это именованная спецификация одного или более столбцов (для каждого столбца указывается имя, а также его тип или домен). После определения строчного типа можно специфицировать таблицы, заголовок которых соответствует этому типу или включает его как свою часть.

Наследование таблиц и семантика включения. Если таблица определена на одном строчном типе (без добавления столбцов), то разрешается использовать ее как супертаблицу и производить на ее основе подтаблицы с добавлением столбцов. При этом используется семантика включения.

В ряде случаев использование механизма наследования таблиц с использованием семантики включения позволяет более правильно (без излишеств) спроектировать БД и обойтись без привлечения операторов объединения при формулировке сводных запросов.

Типы коллекций. Типы коллекций находятся ближе всего к конструируемым типам языков программирования и внедряются в объектно-реляционные БД, чтобы ликвидировать или, по крайней мере, смягчить ограничение первой нормальной формы (атомарности значений столбцов), накладываемое классической реляционной моделью данных. К типам коллекций относятся типы массива, списка и множества. Для каждой разновидности типа коллекции имеется предопределенный набор операций (например, доступ к элементу массива по индексу). После определения любого типа коллекции его можно использовать как встроенный тип. В частности, типом столбца таблицы может быть тип множества, базовым типом которого является строчный тип. Понятно, что с использованием типов коллекций можно организовывать БД с произвольно сложной иерархической структурой.

Объектные типы данных. Эта разновидность типов ближе всего к абстрактным типам данных в языках программирования. Идея состоит в том, что сначала специфицируется определяемый пользователем тип данных (переименовывается некоторый встроенный тип, определяется строчный тип или тип коллекции). Затем для этого типа можно специфицировать ряд определяемых пользователем функций. Строго говоря, после этого можно считать, что тип инкапсулирован этим набором функций, хотя не все разработчики считают строгую инкапсуляцию необходимой.

После полной спецификации объектного типа его можно использовать как встроенный или любой ранее определенный тип. В современных

объектно-реляционных СУБД реализована полная возможность наследования объектных типов.

10. Встроенные и определяемые пользователями типы данных

1) Встроенные

Встроенные типы данных, т.е. типы, предопределенные в языке программирования или языке БД (SQL). Обычно в состав встроенных типов данных включаются такие типы, операции над значениями которых поддерживаются командами компьютеров. В традиционный набор встроенных типов обычно входят следующие:

- символный тип CHARACTER (или CHAR) – это набор печатных символов из алфавита, зафиксированного в описании языка (для большинства языков англоязычного происхождения этот алфавит соответствует кодовому набору ASCII) либо произвольная комбинация нулей и единиц, размещаемых в одном байте.

- логический тип BOOLEAN обычно содержит два значения - TRUE (истина) и FALSE (ложь). Несмотря на то, что для хранения значений этого типа теоретически достаточно одного бита, обычно переменные занимают один байт памяти. Над булевскими значениями возможны операции конъюнкции (& или AND), дизъюнкции (| или OR) и отрицания (~ или NOT).

- тип целых чисел INTEGER в общем случае включает подмножество целых чисел, определяемое числом разрядов, которое используется для внутреннего представления значений. При определении типа целых чисел обычно стремятся к тому, чтобы множество его значений было симметрично относительно нуля (это стимулируется и стандартными свойствами машинной целочисленной арифметики). Отрицательные целые числа обычно представляют в дополнительном коде.

Наряду со знаковыми целыми типами в языках часто поддерживаются беззнаковые целые. Для поддержки численных вычислений в языках обычно специфицируется встроенный тип чисел с плавающей точкой с базовым названием REAL или FLOAT.

2) Определяемые пользователем

Абстрактные (определяемые пользователями) типы данных. Наличие перечисляемых, уточняемых и конструируемых типов данных в сочетании со средствами выделения динамической памяти позволяет конструировать и использовать структуры данных, достаточные для создания сложных программ. Ограниченность этих средств состоит в том, что при определении типов и создании структур невозможно зафиксировать правила их использования.

11. Уточняемые, перечисляемые и конструируемые типы данных

Уточняемые типы данных – это типы, которые определены на основе встроенного типа данных, значения которого каким-либо образом упорядочены или ограничены.

Основной проблемой уточняемых типов является потребность в динамическом контроле значений, формируемых при вычислении выражений и возвращаемых функциями. Если для значений базовых типов (по крайней мере, числовых) такой контроль, как правило, поддерживается аппаратной частью компьютера, то для уточняемых типов требуется программный контроль, вызывающий серьезные накладные расходы.

Перечисляемые типы данных представляют явно определяемые целые типы с конечным числом именованных значений. Это очень простой и легко реализуемый механизм, часто являющийся очень полезным.

Обычно для любого перечисляемого типа предопределяются операции получения значения по его номеру и получения номера по значению. Кроме

того, для перечисляемого типа предопределяются операции сравнения и получения следующего и предыдущего значения.

Конструируемые типы (составные) обладают той особенностью, что в языке предопределены средства спецификации таких типов и некоторый набор операций, дающих возможность доступа к компонентам составных значений. Любое значение конструируемого типа состоит из значений одного или нескольких других типов.

Наиболее распространенные разновидности конструируемых типов – это типы массивов, записей и множеств.

Массивы. Базовым типом массива может быть любой встроенный или определенный тип, в том числе и тип массива, тогда говорят о многомерных массивах или матрицах. Типы массивов позволяют работать с регулярными структурами данных, каждый элемент которых относится к одному и тому же базовому типу.

Записи. Позволяют определять и использовать нерегулярные структуры данных, элементы которых могут относиться к разным встроенным или явно определенным типам данных.

Множества. С использованием механизма множеств можно писать лаконичные и красивые программы, но нужно отдавать себе отчет в том, что для эффективной реализации множеств требуются серьезные ограничения их мощности. Обычно в реализациях языков допускаются множества, мощность базового типа которых не превосходит длину машинного слова, чтобы для выполнения операций над множествами можно было прямо использовать машинные команды.

12. Понятие транзакции. Управление транзакциями. Журнал. Журнализация изменений БД.

Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменения БД во внешней памяти, либо ни одно из изменений никак не отражается на состоянии БД. Понятие транзакции

необходимо для поддержания логической целостности БД и в однопользовательском режиме, но более важное значение оно имеет при работе с БД в многопользовательском.

Каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения. Это свойство, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может ощущать себя единственным.

С управлением транзакциями в многопользовательском режиме работы связаны понятия сериализации транзакций и сериального плана выполнения смеси транзакций.

Под сериализацией параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций – это такой план, который приводит к сериализации транзакций. Понятно, что если удастся добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, присутствие других транзакций будет незаметно (кроме некоторого замедления работы по сравнению с однопользовательским режимом).

Журнализация. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти, т.е. СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризующиеся потерей информации на носителях внешней памяти.

Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной.

В любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал – это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда существуют две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. Во всех случаях придерживается стратегия "упреждающей" записи в журнал (протокол Write Ahead Log – WAL). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем сам измененный объект попадет во внешнюю память основной части БД. Таким образом, если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД практически после любого сбоя.

13. Файловые системы, как дореляционные модели данных

Файловая система – это часть ОС, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В первых компьютерах использовались два вида устройств внешней памяти – магнитные ленты и барабаны. Емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (похожи на магнитные диски с фиксированными головками) давали возможность произвольного доступа к данным, но были ограниченного размера.

Эти ограничения не являлись слишком существенными для численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти (например, на последовательной магнитной ленте), обеспечивающее эффективное выполнение программы.

Но для ИС, в которых потребность в текущих данных определяется конечным пользователем, наличие только магнитных лент и барабанов было неудовлетворительно. Представьте себе покупателя билета, который, стоя у кассы, должен дожидаться полной перемотки магнитной ленты. Одним из

естественных требований к таким системам является удовлетворительная средняя скорость выполнения операций.

Именно требования нечисленных приложений вызвали появление съемных магнитных дисков с подвижными головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных.

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной памятью и устройствами внешней памяти с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ ОС). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

Историческим шагом явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы, **файл** - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

В широком смысле понятие файловой системы включает:

- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, удаление, чтение, запись, именование, поиск и другие операции над файлами.

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС, как на используемые символы, так и на длину имени. Все современные файловые системы поддерживают многоуровневое именование файлов за счет поддержания во внешней памяти дополнительных файлов со специальной структурой – каталогов. Каждый каталог содержит имена каталогов и/или файлов, содержащихся в данном каталоге.

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

14. Ограничения файловых систем на примере учета сотрудников*

ИС – информационная система.

Предположим, что необходимо реализовать простую ИС, поддерживающую учет сотрудников некоторой организации. Система должна выдавать списки сотрудников в соответствии с указанными номерами отделов, поддерживать регистрацию перевода сотрудника из одного отдела в другой, приема на работу новых сотрудников и увольнения работающих. Для каждого отдела должна поддерживаться возможность получения имени руководителя этого отдела, общей численности отдела, общей суммы выплаченной в последний раз зарплаты и т.д. Для каждого сотрудника должна поддерживаться возможность выдачи номера удостоверения по полному имени сотрудника, выдачи полного имени по номеру удостоверения, получения информации о текущем соответствии занимаемой должности сотрудника и о размере зарплаты.

Реализуем данную ИС на основе файловой системы и воспользуемся при этом одним файлом, расширив базовые возможности файловой системы за счет специальной библиотеки функций. Поскольку минимальной информационной единицей является сотрудник, естественно потребовать, чтобы в этом файле содержалась одна запись для каждого сотрудника. Очевидно, что поля таких записей должны содержать полное имя сотрудника (СОТР_ИМЯ), номер его удостоверения (СОТР_НОМЕР), информацию о его соответствии занимаемой должности (СОТР_СТАТ - для простоты, «да» или «нет»), размер зарплаты (СОТР_ЗАРП), номер отдела (СОТР_ОТД_НОМЕР) и эта же запись должна содержать имя руководителя отдела (СОТР_ОТД_РУК).

Для поддержания нормальной работы ИС требуется возможность многоключевого доступа к этому файлу по уникальным ключам СОТР_ИМЯ и СОТР_НОМЕР. Кроме того, должна обеспечиваться возможность выбора всех записей с общим заданным значением СОТР_ОТД_НОМЕР, то есть доступ по неуникальному ключу. Чтобы получить численность отдела или общий размер зарплаты, ИС должна будет каждый раз выбирать все записи о сотрудниках отдела и подсчитывать соответствующие общие значения.

Таким образом, для реализации даже такой простой ИС на базе файловой системы, во-первых, требуется создание достаточно сложной надстройки, обеспечивающей многоключевой доступ к файлам, и, во-вторых, неизбежны существенная избыточность хранения (для каждого сотрудника данного отдела повторяется имя руководителя отдела) и выполнение массовой выборки и вычислений для получения сводной информации об отделах. Кроме того, если в ходе эксплуатации системы возникнет потребность, например, выдавать списки сотрудников, получающих заданную зарплату, то придется

либо полностью просматривать файл, либо реструктурировать его, объявляя ключевым поле СОТР_ЗАРП.

Для решения проблем первое, что необходимо, это поддерживать два многоключевых файла СОТРУДНИКИ и ОТДЕЛЫ. Тогда первый файл должен содержать поля СОТР_ИМЯ, СОТР_НОМЕР, СОТР_СТАТ, СОТР_ЗАРП и СОТР_ОТД_НОМЕР, а второй - ОТД_НОМЕР, ОТД_РУК, СОТР_ЗАРП (общий размер зарплаты) и ОТД_РАЗМЕР (общее число сотрудников в отделе). Каждый из файлов будет содержать только недублируемую информацию, а необходимость в динамических вычислениях сводной информации не возникнет.

Обновленная таким образом ИС будет обладать возможностями, сближающими ее с СУБД. Прежде всего, теперь система должна знать, что работает с двумя информационно связанными файлами, ей должны быть известны структура и смысл каждого поля (например, что СОТР_ОТД_НОМЕР в файле СОТРУДНИКИ и ОТД_НОМЕР в файле ОТДЕЛЫ означают одно и то же), а также понимать, что в ряде случаев изменение информации в одном файле должно вызывать модификацию второго файла, чтобы общее содержимое файлов было согласованным. Например, если на работу принимается новый сотрудник, то необходимо добавить запись в файл СОТРУДНИКИ, а также соответствующим образом изменить поля ОТД_ЗАРП и ОТД_РАЗМЕР в записи файла ОТДЕЛЫ, описывающей отдел этого сотрудника.

Согласованность данных является ключевым понятием БД и если ИС (даже такая простая, как в примере) поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает БД. С другой стороны, если некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее СУБД, так как требование поддержания согласованности данных в нескольких файлах не позволяет обойтись библиотекой функций: система должна обладать некоторыми собственными данными (метаданными) и знаниями, определяющими целостность данных.

Рассмотрим еще несколько особенностей работы с данным, хранящимися в файлах. Представим, что обрабатывается операция регистрации нового сотрудника. Следуя требованиям согласованного изменения файлов, сначала вставим новую запись в файл СОТРУДНИКИ, а затем будем модифицировать запись файла ОТДЕЛЫ, но если именно в этот момент произойдет аварийное выключение электрического питания, то после перезапуска системы файловая БД будет находиться в рассогласованном состоянии. Потребуется явно проверить соответствие информации в файлах СОТРУДНИКИ и ОТДЕЛЫ и

привести информацию в согласованное состояние. Настоящие СУБД берут такую работу на себя.

Наконец, представим, параллельную работу с БД нескольких сотрудников. Для обеспечения корректности изменений на все время модификации любого из двух файлов одним пользователем, доступ других к этому файлу будет блокирован. Таким образом, зачисление на работу Петра Ивановича Сидорова существенно затормозит получение информации о сотруднике Иване Сидоровиче Петрове, даже если они будут работать в разных отделах. Настоящие СУБД обеспечивают более тонкую синхронизацию параллельного доступа к данным.

Таким образом, появление СУБД решает множество проблем, которые затруднительно или вообще невозможно было решить при использовании файловых систем. Современные системы управления файлами и управления базами данных представляют собой совершенные инструменты, каждый из которых может быть очень успешно применен в соответствующей области деятельности [4].

15. Иерархические модели данных

Организация данных в СУБД иерархического типа определяется в терминах: атрибут, запись (группа), групповое отношение.

Атрибут (элемент данных) – наименьшая единица структуры иерархических данных. Обычно каждому элементу при описании БД присваивается уникальное имя. По этому имени к нему обращаются при обработке. Такой элемент данных также часто называют полем.

Запись – именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются.

Тип записи определяется составом ее атрибутов. Экземпляр записи – это конкретная запись с конкретными значениями атрибутов.

Групповое отношение – это иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения, предок) называется исходной записью, а дочерние записи (члены группового отношения, потомки) – подчиненными. Таким образом, иерархическая БД состоит из упорядоченного набора деревьев.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей, начиная от корневой.

При графическом изображении (диаграмма Бахмана) групповые отношения изображают дугами ориентированного графа, а типы записей – вершинами.

Для групповых отношений в иерархической модели обеспечивается автоматический режим включения и фиксированное членство. Это означает, что для запоминания любой некорневой записи в БД должна существовать ее родительская запись. При удалении родительской записи автоматически удаляются все подчиненные.

Пример 1. Рассмотрим модель данных предприятия (рис. 5): предприятие состоит из отделов, в которых работают сотрудники. В каждом отделе может работать несколько сотрудников, но сотрудник не может работать более чем в одном отделе.

Поэтому, для ИС управления персоналом необходимо создать групповое отношение, состоящее из родительской записи ОТДЕЛ (НАИМЕНОВАНИЕ_ОТДЕЛА, ЧИСЛО_РАБОТНИКОВ) и дочерней записи СОТРУДНИК (ФАМИЛИЯ, ДОЛЖНОСТЬ, ОКЛАД). Это отношение для двух дочерних записей показано на рис. 5 а.

Для автоматизации учета контрактов с заказчиками необходимо создание еще одной иерархической структуры: заказчик - контракты с ним - сотрудники, задействованные в работе над контрактом. Это дерево будет включать записи ЗАКАЗЧИК (НАИМЕНОВАНИЕ_ЗАКАЗЧИКА, АДРЕС), КОНТРАКТ (НОМЕР, ДАТА, СУММА), ИСПОЛНИТЕЛЬ (ФАМИЛИЯ, ДОЛЖНОСТЬ, НАИМЕНОВАНИЕ_ОТДЕЛА) (рис. 5 б).

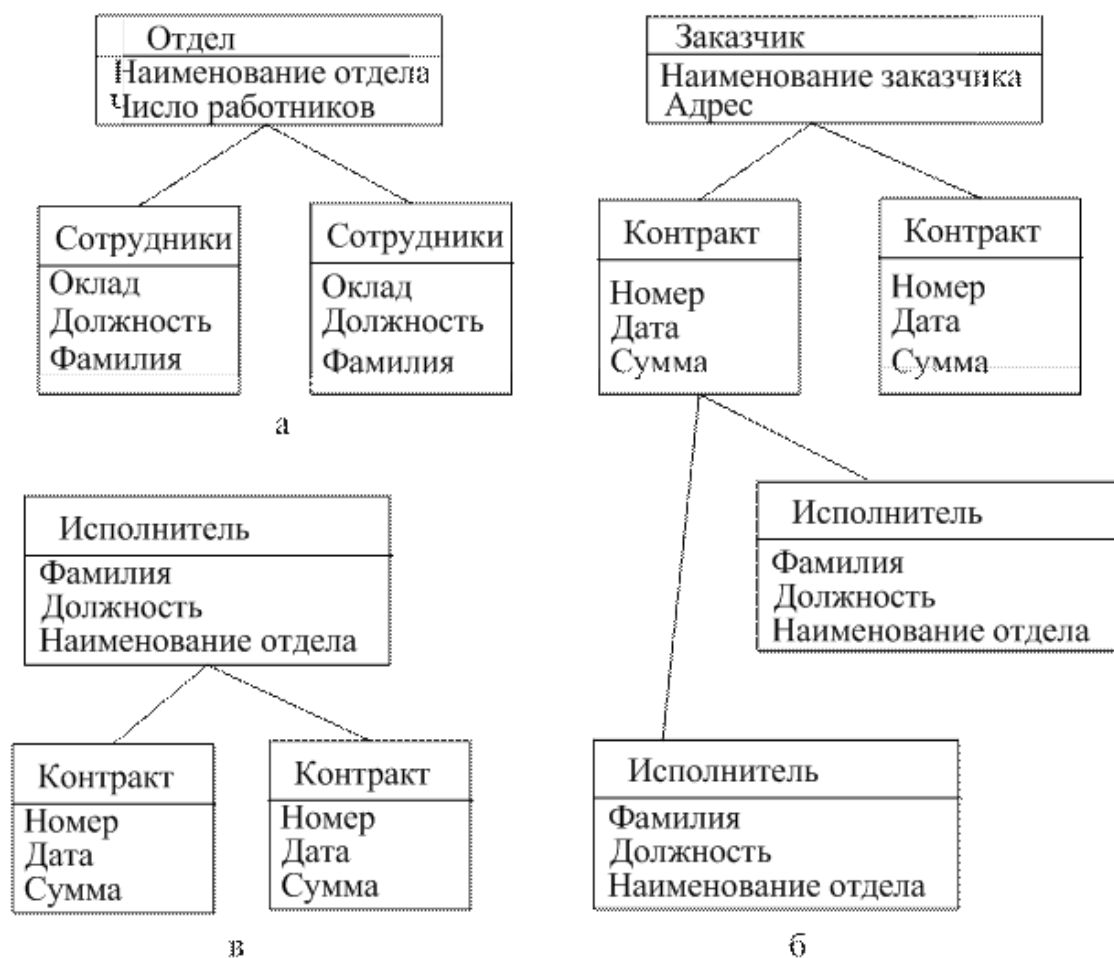


Рис. 5 – Иерархическая модель данных

На рис. 5 хорошо видны недостатки иерархических БД:

1) Частично дублируется информация между записями СОТРУДНИК и ИСПОЛНИТЕЛЬ (такие записи называют парными), причем в иерархической модели данных не предусмотрена поддержка соответствия между ними.

2) Иерархическая модель реализует отношение “один ко многим” между исходной и дочерней записью, то есть одной родительской записи может соответствовать любое число дочерних. Допустим, что исполнитель может принимать участие более чем в одном контракте (т.е. возникает связь “многие ко многим”). В этом случае в БД необходимо ввести еще одно групповое отношение, в котором ИСПОЛНИТЕЛЬ будет являться исходной записью, а КОНТРАКТ – дочерней (рис. 5 в), что снова приведет к дублированию информации.

Операции над данными, определенные в иерархической модели:

- ДОБАВИТЬ в БД новую запись. При этом для корневой записи обязательно формирование значения ключа.

- ИЗМЕНИТЬ значение данных предварительно извлеченной записи. При этом ключевые данные не должны подвергаться изменениям.
- УДАЛИТЬ некоторую запись и все подчиненные ей записи.
- ИЗВЛЕЧЬ: корневую запись по ключевому значению, следующую запись (извлекается в порядке левостороннего обхода дерева), допускается также последовательный просмотр корневых записей и задание условий выборки.

Все операции изменения применяются только к одной "текущей" записи, которая предварительно извлечена из БД. Такой подход к манипулированию данными получил название "навигационного".

Для обеспечения целостности данных поддерживается только одно ограничение: целостность связей между владельцами и членами группового отношения (никакой потомок не может существовать без предка).

В остальном аппарат обеспечения целостности иерархической модели данных имеет существенные недостатки, так как не обеспечивает автоматическое поддержание соответствия парных записей, входящих в разные иерархии; не реализует явное разделение логических и физических характеристик модели; а непредвиденные запросы могут требовать реорганизации БД.

16. Сетевые модели данных

Сети – это естественный способ представления отношений между объектами. Они широко применяются в математике, исследованиях операций, химии, физике, социологии и других областях знаний.

Сетевая модель данных – это представление данных сетевыми структурами типов записей и связанных отношениями “один к одному” или “один ко многим”.

Сетевой подход к организации данных является расширением иерархического подхода. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также может иметь вид “один ко многим”. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При

этом экземпляре записи не может быть членом двух экземпляров групповых отношений одного типа.

Иерархическая структура (рис. 5) преобразовывается в сетевую следующим образом (рис. 6):

- деревья на рис. 5 а, б заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
- для отображения отношения “многие ко многим” (рис. 5 в), вводится запись СОТРУДНИК_КОНТРАКТ, которая в данном примере не имеет собственных полей (в этой записи может храниться и полезная информация, например, доля вознаграждения сотрудника по данному контракту) и служит только для связи записей КОНТРАКТ и СОТРУДНИК.

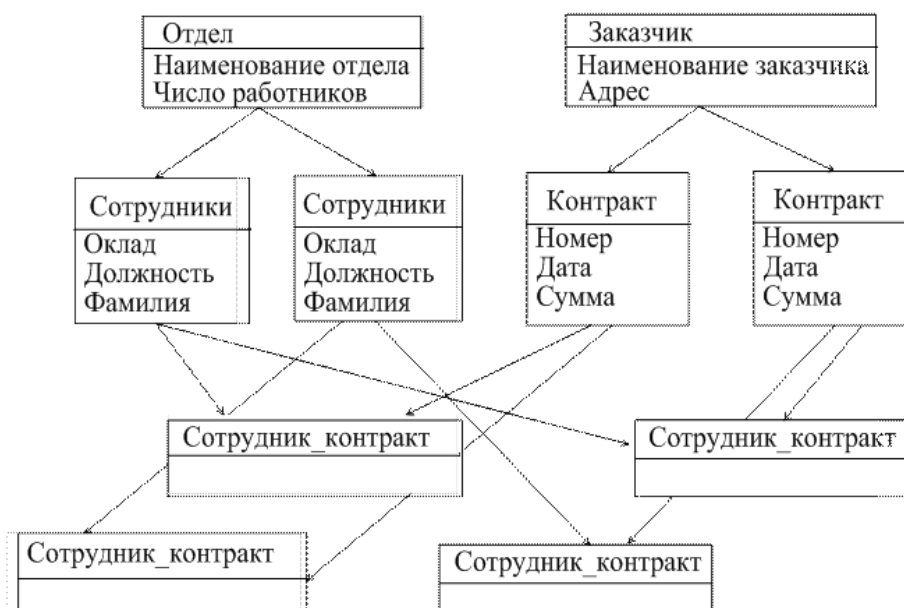


Рис. 6 – Сетевая модель данных

Каждый экземпляр группового отношения характеризуется способом упорядочения подчиненных записей, который может быть:

- произвольный;
- хронологический (очередь);
- обратный хронологический (стек);
- сортированный.

Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой режим упорядочивания:

- автоматический, когда невозможно занести в БД запись без того, чтобы она была сразу же закреплена за владельцем;
- ручной, который позволяет запоминать в БД подчиненную запись, не включая ее немедленно в экземпляр группового отношения;
- режим исключения.

Принято выделять три класса членства подчиненных записей в групповых отношениях.

1. **Фиксированное.** Подчиненная запись жестко связана с записью владельцем и ее можно исключить из группового отношения, только удалив. При удалении записи-владельца все подчиненные записи тоже автоматически удаляются. В рассмотренном примере фиксированное членство предполагается между записями КОНТРАКТ и ЗАКАЗЧИК, поскольку контракт не может существовать без заказчика.

2. **Обязательное.** Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи СОТРУДНИК и ОТДЕЛ. Если отдел расформировывается, все его сотрудники должны быть либо переведены в другие отделы, либо уволены.

3. **Необязательное.** Можно исключить запись из группового отношения, но сохранить ее в БД, не прикрепляя к другому владельцу. Примером может служить связь между записями СОТРУДНИК и КОНТРАКТ, поскольку в организации могут существовать работники, чья деятельность не связана с выполнением каких-либо договорных обязательств перед заказчиками.

Операции над данными, определенные в сетевой модели:

- **ДОБАВИТЬ** – внести запись в БД и, в зависимости от режима включения, либо включить ее в групповое отношение, где она объявлена подчиненной, либо не включать ни в какое групповое отношение.
- **ВКЛЮЧИТЬ В ГРУППОВОЕ ОТНОШЕНИЕ** – связать существующую подчиненную запись с записью-владельцем.
- **ПЕРЕКЛЮЧИТЬ** – связать существующую подчиненную запись с другой записью-владельцем в том же групповом отношении.
- **ОБНОВИТЬ** – изменить значение элементов предварительно извлеченной записи.

- **ИЗВЛЕЧЬ** – извлечь записи последовательно по значению ключа, а также, используя групповые отношения, от владельца можно перейти к записям-членам, а от подчиненной записи к владельцу набора.

- **УДАЛИТЬ** – убрать из БД запись. Если эта запись является владельцем группового отношения, то анализируется класс членства подчиненных записей. Обязательные члены должны быть предварительно исключены из группового отношения, фиксированные удалены вместе с владельцем, необязательные останутся в БД.

- **ИСКЛЮЧИТЬ ИЗ ГРУППОВОГО ОТНОШЕНИЯ** – разорвать связь между записью-владельцем и записью-членом.

Ограничения целостности. Как и в иерархической модели обеспечивается только поддержание целостности по ссылкам (владелец отношения – член отношения).

17. Реляционные модели данных, история возникновения, понятие реляционной алгебры и реляционного исчисления

Реляционная алгебра – это процедурный язык обработки реляционных таблиц, а реляционное исчисление – непроцедурный язык создания запросов.

В 1970-1971 годах Е. Ф. Кодд опубликовал две статьи, в которых впервые ввел понятия реляционной модели данных и реляционных языков обработки данных – реляционной алгебры и реляционного исчисления.

Будучи математиком по образованию, он предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение) и показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение (relation).

Основное преимущество этого предложения было в том, что все существующие к тому времени подходы связывания записей из разных файлов использовали физические указатели или адреса на диске. В своей работе Кодд продемонстрировал, что такие БД существенно ограничивают число типов манипуляций данными и они очень чувствительны к изменениям в физическом окружении. То есть, когда в компьютерной системе устанавливался новый накопитель или изменялись адреса хранения данных, требовалось дополнительное программное преобразование файлов БД. Также, если к формату записи в файле добавлялись новые поля, то физические адреса всех записей файла изменялись. Все эти проблемы преодолела реляционная модель, основанная на логических отношениях данных.

В статье утверждалось, что «реляционная модель предоставляет средства описания данных на основе только их естественной структуры, т.е. без

потребности введения какой-либо дополнительной структуры для целей машинного представления». Другими словами, представление данных в реляционной модели не зависит от способа их физической организации.

Таким образом, фундаментальным понятием реляционной модели данных является понятие отношения. Отношение представляют в виде таблицы. Например, таблица на рис. 7 содержит некоторые сведения о работниках предприятия.

Отношение	целое	строка		целое		Типы данных
	номер	имя	должность	деньги		Имена доменов
	Табельный номер	Имя	Должность	Оклад	Премия	Имена атрибутов
	2934	Иванов	инженер	112	40	Кортежи
	2935	Петров	вед инженер	144	50	
	2936	Сидоров	бухгалтер	92	35	

Рис. 7.— Основные компоненты реляционного отношения

18. Организация внешней памяти реляционной СУБД

СУБД обладает рядом особенностей, влияющих на организацию внешней памяти [4, 6]. К наиболее важным можно отнести следующие:

□ Наличие двух уровней системы: уровня непосредственного управления данными во внешней памяти и языкового уровня. При такой организации подсистема нижнего уровня должна поддерживать во внешней памяти набор базовых структур, конкретная интерпретация которых входит в число функций подсистемы верхнего уровня.

□ Поддержание системных каталогов. Информация, связанная с именованием объектов БД и их параметрами, поддерживается подсистемой языкового уровня.

□ Регулярность структур данных. Поскольку основным объектом реляционной модели данных является плоская таблица, главный набор объектов внешней памяти может иметь очень простую регулярную структуру. При этом необходимо обеспечить возможность эффективного выполнения операторов как над одной таблицей, так и над несколькими. Для этого во внешней памяти должны поддерживаться дополнительные «управляющие» структуры — индексы.

□ Надежное хранение. Для выполнения требования надежного хранения БД необходимо поддерживать избыточность хранения данных, что обычно реализуется в виде журнала изменений БД.

Наличие перечисленных особенностей привело к созданию во внешней памяти следующих разновидностей объектов базы данных:

□ строки таблиц – основная часть БД, непосредственно видимая пользователям;

□ управляющие структуры – индексы, создаваемые из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;

□ журнал - специальный файл, поддерживаемый для удовлетворения потребности в надежном хранении данных;

□ служебная информация - информация, которая поддерживается для удовлетворения внутренних потребностей нижнего уровня системы (например, информация о свободной памяти).

Хранение отношений. Существуют два принципиальных подхода к физическому хранению таблиц. Наиболее распространенным является строковое (страничное) хранение. Естественно, это обеспечивает быстрый доступ к целому кортежу, но при этом во внешней памяти дублируются общие значения разных кортежей одной таблицы и могут потребоваться лишние обмены с внешней памятью, если нужна часть кортежа.

Альтернативным (менее распространенным) подходом является хранение отношения по столбцам, т.е. единицей хранения является столбец таблицы с исключенными дубликатами. Естественно, что при такой организации суммарно в среднем тратится меньше внешней памяти, поскольку дубликаты значений не хранятся; за один обмен с внешней памятью в общем случае считывается больше полезной информации. Дополнительным преимуществом является возможность использования значений столбца для оптимизации выполнения операций соединения. Но при этом требуются существенные дополнительные действия для сборки целой строки (или ее части).

Опишем основные принципы организации хранения кортежей.

Каждый кортеж обладает уникальным идентификатором, не изменяемым во все время существования кортежа и позволяющим выбрать кортеж в основную память не более чем за два обращения к внешней памяти.

Обычно каждый кортеж хранится целиком в одной странице. Из этого следует, что максимальная длина кортежа любой таблицы ограничена размерами страницы и возникает вопрос хранения «длинных» данных. Для его

решения применяется несколько методов: хранение таких данных в отдельных файлах вне БД с заменой «длинного» данного в кортеже на имя соответствующего файла; хранение внутри БД в отдельном наборе страниц внешней памяти, связанном физическими ссылками; хранение в виде В-деревьев последовательностей байт.

Как правило, в одной странице данных хранятся кортежи только одной таблицы. Существуют варианты с возможностью хранения в одной странице кортежей нескольких таблиц. Хотя это вызывает некоторые дополнительные расходы по части служебной информации (при каждом кортеже нужно хранить информацию о соответствующей таблице), но при этом позволяет резко сократить число обменов с внешней памятью при выполнении соединений.

Изменение структуры хранимой таблицы с добавлением нового поля не вызывает потребности в физической реорганизации таблицы. Достаточно лишь изменить информацию в описателе таблицы, и расширяться кортежи только при занесении информации в новое поле.

Проблема распределения памяти в страницах данных связана с проблемами синхронизации и журнализации и не всегда тривиальна. Например, если в ходе выполнения транзакции некоторая страница данных опустошается, то ее нельзя перевести в статус свободных страниц до конца транзакции, поскольку при откате транзакции удаленные при прямом выполнении транзакции и восстановленные при ее откате кортежи должны получить те же самые идентификаторы.

Распространенным способом повышения эффективности СУБД является кластеризация таблицы по значениям одного или нескольких столбцов. Полезной для оптимизации соединений является совместная кластеризация нескольких таблиц. С целью использования возможностей распараллеливания обменов с внешней памятью иногда применяют схему декластеризованного хранения таблиц: кортежи с общим значением столбца декластеризации размещают на разных дисковых устройствах, обмены с которыми можно выполнять параллельно.

Индексы. Основное назначение индексов состоит в обеспечении эффективного прямого доступа к строке таблицы по ключу. На практике при объявлении первичного ключа таблицы автоматически создается уникальный индекс, а единственным способом объявления возможного ключа, отличного от первичного, является явное создание уникального индекса. Это связано с тем, что для проверки сохранения свойства уникальности возможного ключа, так или иначе, требуется индексная поддержка.

Поскольку при выполнении многих операций языкового уровня требуется сортировка отношений в соответствии со значениями некоторых атрибутов, полезным свойством индекса является обеспечение последовательного просмотра строк таблицы в диапазоне значений ключа в порядке возрастания или убывания этих значений.

Наконец, одним из способов оптимизации выполнения соединения таблиц по равенству значений некоторых столбцов является организация так называемых мультииндексов, обладающих общими атрибутами нескольких таблиц. Любой из этих атрибутов (или их набор) может выступать в качестве ключа мультииндекса. Значению ключа сопоставляется набор строк всех связанных мультииндексом отношений, значения выделенных атрибутов которых совпадают со значением ключа.

Общей идеей любой организации индекса, поддерживающего прямой доступ по ключу и последовательный просмотр в порядке возрастания или убывания значений ключа, является хранение упорядоченного списка значений ключа с привязкой к каждому значению ключа списка идентификаторов строк. Одна организация индекса отличается от другой главным образом по способу поиска ключа с заданным значением.

В-дерево. Наиболее популярным подходом к организации индексов в БД является использование техники В-деревьев. С точки зрения внешнего логического представления В-дерево – это сбалансированное сильноветвистое дерево во внешней памяти. Сбалансированность означает, что длина пути от корня дерева к любому его листу одна и та же. Ветвистость дерева – это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения физической организации В-дерево представляется как мультисписочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру.

Поиск в В-дереве – это прохождение от корня к листу в соответствии с заданным значением ключа. Заметим, что поскольку деревья сильноветвистые и сбалансированные, то для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью. Более точно: если в сбалансированном дереве на внутренней странице помещается n ключей, то при хранении m записей требуется дерево глубиной $\log_n(m)$, где \log_n – логарифм по основанию n . Если n достаточно велико, то глубина дерева невелика, и производится быстрый поиск.

Основной особенностью В-деревьев является автоматическое поддержание свойства сбалансированности. При выполнении операций

вставки и удаления свойство сбалансированности В-дерева сохраняется, а внешняя память расходуется достаточно экономно.

Проблемой использования В-деревьев является то, что при выполнении операций модификации слишком часто могут возникать расщепления и слияния.

Хеширование. Альтернативным, более популярным, но и более сложным в понимании и реализации подходом к организации индексов является использование техники хеширования. Общей идеей методов хеширования является применение к значению ключа некоторой функции свертки (хэш-функции), вырабатывающей значение меньшего размера, которое затем используется для доступа к записи.

В самом простом, классическом случае свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значения свертки. При возникновении коллизий (одна и та же свертка для нескольких значений ключа) образуются цепочки переполнения. Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, то возникнет слишком много цепочек переполнения, и главное преимущество хеширования – доступ к записи почти всегда за одно обращение к таблице – будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции со значением свертки большего размера.

В случае работы с БД такие действия являются неприемлемыми. Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

Чтобы избежать потребности полной переделки справочников, при их организации часто используют технику В-деревьев с расщеплениями и слияниями. Хэш-функция при этом меняется динамически, в зависимости от глубины В-дерева. Путем дополнительных технических ухищрений удастся добиться сохранения порядка записей в соответствии со значениями ключа.

Журнальная информация. Журнал обычно представляет собой последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти. В грамотно организованных системах структура (и тем более смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление.

Служебная информация. Для корректной работы подсистемы управления данными во внешней памяти необходимо поддерживать информацию, которая используется только этой подсистемой и не видна подсистеме языкового уровня. Набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных:

- системные каталоги, описывающие физические свойства объектов БД, например, число атрибутов отношения, их размер и типы данных; описание индексов, определенных для данного отношения и т. д;
- описатели свободной и занятой памяти в страницах таблиц, которые требуются для нахождения свободного места при занесении строки.
- данные для связывания страниц одной таблицы. Если в одном файле внешней памяти могут располагаться страницы нескольких таблиц (обычно к этому стремятся), то нужно каким-то образом связать страницы одной таблицы. Тривиальный способ использования прямых ссылок между страницами часто приводит к затруднениям при синхронизации транзакций (например, особенно трудно освобождать и заводить новые страницы отношения). Поэтому стараются использовать косвенное связывание страниц с использованием служебных индексов. В частности, известен общий механизм для описания свободной памяти и связывания страниц на основе В-деревьев.

19. Основные определения и компоненты реляционной модели

Рассмотрим основные понятия реляционной модели данных [3, 4, 6].

Наименьшая единица данных реляционной модели – это отдельное атомарное (неделимое) для данной модели **значение данных**. В одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой – как три различных значения.

Строки таблицы в терминах реляционной модели соответствуют кортежам. Каждый **кортеж отношения** представляет собой множество пар вида «имя_атрибута, значение_атрибута». Каждая строка фактически представляет собой описание одного объекта реального мира (в данном случае работника), характеристики которого содержатся в столбцах (атрибутах).

Атрибут отношения – это пара вида «имя_атрибута, имя_домена». Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Каждый атрибут определен на домене, поэтому **домен** можно рассматривать как множество допустимых значений данного атрибута. Другими словами доменом называется множество атомарных значений одного и того же типа.

Несколько атрибутов одного отношения и даже атрибуты разных отношений могут быть определены на одном и том же домене. В примере на рис. 7 атрибуты Оклад и Премия определены на домене Деньги. Поэтому, понятие домена имеет семантическую нагрузку: данные можно считать сравнимыми только тогда, когда они относятся к одному домену. Таким образом, сравнение атрибутов Табельный номер и Оклад является семантически некорректным, хотя они содержат данные одного типа.

Именованное множество пар «имя атрибута, имя домена» называется **схемой отношения**. Схема содержит две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество атрибутов отношения. Он описывает декартово произведение доменов, на котором задано отношение. Заголовок статичен, он не меняется во время работы с БД. Если в отношении изменены, добавлены или удалены атрибуты, то в результате получим уже другое отношение (пусть даже с прежним именем).

Набор заголовков отношений, входящих в БД называется **схемой реляционной БД**.

Тело отношения представляет собой набор кортежей, т.е. подмножество декартового произведения доменов. Таким образом, тело отношения собственно и является отношением в математическом смысле слова. Оно может изменяться во время работы с БД – кортежи могут изменяться, добавляться и удаляться.

Число атрибутов в отношении называют **степенью (или -арностью) отношения**.

Количество кортежей отношения называют **мощностью отношения** (кардинальность, кардинальное число).

Таким образом, **реляционной БД** называется набор отношений.

В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций (связей) между кортежами разных отношений используется дублирование их ключей. Рассмотренный ранее пример БД, содержащей сведения о подразделениях предприятия и работающих в них сотрудниках, применительно к реляционной модели будет иметь вид:



Рис. 8 – Пример реляционной модели данных

20. Свойства отношений и реляционные ключи

Любое отношение можно изобразить в виде таблицы, но нужно понимать, что отношения не являются просто таблицами. Это близкие, но не совпадающие понятия. То, что не каждая таблица может задавать отношение, наиболее очевидно следует из анализа свойств отношений.

Сформулируем фундаментальные свойства отношений [6].

1. В отношении нет одинаковых кортежей. Действительно, тело отношения есть множество кортежей и, как всякое множество, не может содержать неразличимые повторяющиеся элементы. Таблицы в отличие от отношений могут содержать одинаковые строки.

2. Кортежи не упорядочены (сверху вниз). Действительно, несмотря на то, что отношение **Сотрудники** представлено в виде таблицы, нельзя сказать, что сотрудник Иванов «предшествует» сотруднику Петрову. Причина этого, та же \square тело отношения есть множество, а множество не упорядочено. И это вторая причина, по которой нельзя отождествить отношения и таблицы \square строки в таблицах упорядочены, поэтому одно и то же отношение может быть изображено разными таблицами, в которых строки идут в различном порядке.

3. Атрибуты не упорядочены (слева направо). Так как каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения. Это свойство несколько отличает реляционное отношение от математического определения отношения (компоненты кортежей там упорядочены). Но это также третья причина, по которой нельзя отождествить отношения и таблицы – столбцы в таблице упорядочены и одно и то же

отношение может быть изображено разными таблицами, в которых столбцы идут в различном порядке.

4. Все значения атрибутов атомарны. То есть любой столбец отношения должен содержать данные только одного типа, а все используемые типы данных должны быть простыми. Это четвертое отличие отношений от таблиц – в ячейки таблиц можно поместить что угодно – массивы, структуры, и даже другие таблицы.

Атрибут, значение которого однозначно идентифицирует кортежи, называется **ключевым (ключом)**. В примере ключом является атрибут Табельный номер, поскольку его значение уникально для каждого работника предприятия. Если кортежи идентифицируются только сцеплением значений нескольких атрибутов, то говорят, что отношение имеет составной ключ.

Необходимость выбора ключевого атрибута связана с определением отношения. Поскольку отношение (обозначим его R) – это множество, а множества по определению не содержат совпадающих элементов, то никакие два кортежа отношения не могут быть дубликатами друг друга в любой произвольно-заданный момент времени. Пусть R отношение с атрибутами A_1, A_2, \dots, A_n , тогда множество атрибутов $\{A_1, A_2, \dots, A_k\}$ отношения является возможным ключом отношения тогда и только тогда, когда удовлетворяются два независимых от времени условия:

□ **уникальность**: в произвольный заданный момент времени никакие два различных кортежа отношения не имеют одного и того же значения для $\{A_1, A_2, \dots, A_k\}$.

□ **минимальность**: ни один из атрибутов $\{A_1, A_2, \dots, A_k\}$ не может быть исключен из $\{A_1, A_2, \dots, A_k\}$ без нарушения уникальности.

Каждое отношение обладает хотя бы одним возможным ключом, поскольку, по меньшей мере, комбинация всех его атрибутов удовлетворяет условию уникальности. Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Остальные возможные ключи, если они есть, называются альтернативными ключами.

В целом в реляционных моделях используются следующие виды реляционных ключей:

1. **Суперключ** – атрибут или множество атрибутов, которые единственным образом идентифицируют кортеж данного отношения.

2. Потенциальный ключ – суперключ, который не содержит подмножества, также являющегося суперключом данного отношения.

3. Первичный ключ – потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения.

4. Альтернативный ключ – потенциальный ключ, не являющийся первичным ключом.

5. Внешний ключ – атрибут или множество атрибутов внутри отношения, которое соответствует потенциальному ключу другого отношения.

21. Понятия целостности, аномалий и нормализации, основные виды

При описании реляционной модели использовалась наиболее распространенная трактовка, которая принадлежит К. Дейту. Также согласно Дейту, реляционная модель состоит из трех частей [6, 7]:

1. структурной части,
2. целостной части,
3. манипуляционной части.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения. Рассмотренные выше свойства отношений, по сути, отражают структурную часть реляционной модели данных.

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных БД. Под целостностью данных понимают правильность, актуальность данных в любой момент времени.

Поскольку каждый атрибут связан с некоторым доменом, для множества допустимых значений каждого атрибута отношения определяются так называемые ограничения домена.

Помимо этого задаются 3 группы правил целостности, которые являются ограничениям для всех допустимых состояний БД:

- ☐ целостность сущностей (отношений),
- ☐ целостность внешних ключей,
- ☐ целостность, которая определяется пользователем.

Для обеспечения целостности сущностей накладывают ограничения на значения первичных ключей базовых отношений, так что в базовом отношении ни один атрибут первичного ключа не может содержать отсутствующих значений, обозначаемых как NULL.

NULL указывает, что значение атрибута в настоящий момент неизвестно. Его не следует воспринимать как логическую величину “неизвестно” (никакое значение еще не задано) или понимать как нулевое численное значение или заполненную пробелами текстовую строку. Нули и пробелы представляют собой некоторые значения, тогда как ключевое слово NULL призвано обозначать отсутствие какого-либо значения.

По определению, первичный ключ (primary key) □ это уникальный идентификатор кортежей отношения. Это значит, что никакое подмножество ключа не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие значения NULL в любой части первичного ключа, это равноценно утверждению, что не все его атрибуты необходимы для идентификации, что противоречит определению.

Ссылочная целостность касается внешних ключей. Внешний ключ (foreign key) □ это атрибут или множество атрибутов внутри отношения, которое соответствует первичному ключу некоторого другого отношения.

Если некий атрибут присутствует в нескольких отношениях, то его наличие обычно отражает определенную связь между кортежами этих отношений. В этом случае один из ключевых атрибутов первичен (для таблицы предка), второй □ внешний, связующий.

Ограничение ссылочной целостности гласит, что если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению первичного ключа некоторого кортежа в его базовом отношении, либо задаваться как NULL.

Корпоративные ограничения целостности □ это дополнительные правила поддержки целостности данных, определяемые пользователями или администраторами БД.

Например, если в одном отделении предприятия не может работать свыше 20 человек, то пользователь может указать это правило, а СУБД следить за его исполнением.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными - реляционную алгебру и реляционное исчисление. Реляционная алгебра – это процедурный язык обработки реляционных таблиц, а реляционное исчисление – непроцедурный язык создания запросов.

АНОМАЛИИ ОБНОВЛЕНИЯ

Даже одного взгляда на табл. 2 СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ достаточно, чтобы увидеть, что данные хранятся в ней с большой избыточностью. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные о сотрудниках, об отделах, о проектах и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние БД, возникает большое количество проблем.

Проблемы получили название аномалий обновления. Аномалия обновления – это либо неадекватность модели данных предметной области, либо некоторые дополнительные трудности в реализации ограничений предметной области средствами СУБД.

Различают следующие виды аномалий:

- ☐ аномалии вставки (INSERT);
- ☐ аномалии обновления (UPDATE);
- ☐ аномалии удаления (DELETE).

Рассмотрим примеры аномалий в отношении СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ.

Аномалии вставки (INSERT). В отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ нельзя вставить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11, null, null, null). Это сделать невозможно, т.к. атрибут Н_ПРО (номер проекта) входит в состав потенциального ключа, и, следовательно, не может содержать null-значений. Точно также нельзя вставить данные о проекте, над которым пока не работает ни один сотрудник.

Аномалии обновления (UPDATE). Фамилии сотрудников, наименования проектов, номера телефонов повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, или проект меняет наименование,

или меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где эти данные встречаются, иначе отношение станет некорректным (например, один и тот же проект в разных кортежах будет называться по-разному). Таким образом, обновление БД одним действием реализовать невозможно. Для поддержания отношения в целостном состоянии необходимо написать триггер, который при обновлении одной записи корректно исправлял бы данные и в других местах. Как следствие, увеличивается сложность разработки и БД, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Аномалии удаления (DELETE). При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове. Если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11. Если по проекту временно прекращены работы, то при удалении данных о работах по этому проекту будут удалены и данные о самом проекте. При этом если бы был сотрудник, который работал только над этим проектом, то будут потеряны и данные об этом сотруднике.

Причина всех аномалий – хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту). Таким образом, логическая модель данных неадекватна модели предметной области и БД, основанная на такой модели, будет работать неправильно.

Нормализация – это разбиение таблицы на две или более, обладающих лучшими свойствами (по отношению к начальному) при добавлении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта БД, в котором каждый факт появляется лишь в одном месте, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

Каждая таблица в реляционной БД удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не может быть множества таких значений. Любая таблица, удовлетворяющая этому условию, называется нормализованной. Фактически, ненормализованные таблицы, т.е. таблицы, содержащие повторяющиеся группы, не допускаются в реляционной БД.

Всякая нормализованная таблица автоматически считается таблицей в первой нормальной форме (1NF). Таким образом, понятия «нормализованная» и «находящаяся в 1NF» означают одно и то же. Однако на практике термин «нормализованная» часто используется в более узком смысле – «полностью нормализованная», который означает, что в проекте не нарушаются никакие принципы нормализации.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- ☐ первая нормальная форма (1NF);
- ☐ вторая нормальная форма (2NF);
- ☐ третья нормальная форма (3NF);
- ☐ нормальная форма Бойса-Кодда (BCNF);
- ☐ четвертая нормальная форма (4NF);
- ☐ пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

В качестве общих свойств для всех нормальных форм можно отметить:

- ☐ каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- ☐ при переходе к следующей нормальной форме свойства предыдущих сохраняются.

22. Ограничения реляционных БД

Появление реляционных СУБД стало важным шагом вперед по сравнению с иерархическими и сетевыми СУБД, в этих системах стали использоваться непроецедурные языки манипулирования данными и была достигнута значительная степень независимости данных от обрабатывающих программ. Однако со временем выявился ряд недостатков реляционных систем [6].

Во-первых, сама реляционная модель ограничена в представлении данных: не допускает естественного представления данных со сложной (иерархической) структурой, поскольку в ее рамках возможно моделирование

лишь с помощью плоских отношений (таблиц). Все отношения принадлежат одному уровню, многие значимые связи между данными либо теряются, либо их поддержку приходится осуществлять в рамках конкретной прикладной программы.

По определению в реляционной модели поля кортежа могут содержать лишь атомарные значения. Однако, в таких приложениях как САПР, ГИС (геоинформационные системы), искусственный интеллект системы оперируют со сложно-структурированными объектами.

Например, представим объектами БД земельные участки. Каждый участок задается координатами узлов ломаной линии, ограничивающей его по периметру. В этом случае спроектировать реляционную таблицу невозможно, т.к. заранее не известно количество узлов для всех участков. Также невозможно написать общие процедуры (вычисление площади, нахождение пересечения и т.д.) для всех случаев.

Кроме того, даже в том случае, когда сложный объект удастся "уложить" в реляционную БД, его данные распределяются, как правило, по многим таблицам. Соответственно, извлечение каждого такого объекта требует выполнения многих операций соединения (join), что значительно замедляет работу СУБД.

Обойти эти ограничения можно было бы в том случае, если бы реляционная модель допускала:

- ☐ возможность определения новых типов данных;
- ☐ определение наборов операций, связанных с данными определенного типа.

Во-вторых, имеются определенные недостатки и в реализации тех возможностей, которые прямо не предусматриваются реляционной моделью, но стали непременным атрибутом всех современных СУБД:

1. Цикл существования реляционной БД состоит в переходе от одного целостного состояния к другому. Однако нельзя избежать такой ситуации, когда пользователь вводит данные, формально удовлетворяющие ограничениям целостности, но не соответствующие реальному состоянию предметной области. В этом случае происходит потеря правильного непротиворечивого состояния БД.

2. Реляционная СУБД выполняет над данными не только те действия, которые задает пользователь, но и дополнительные операции в соответствии с правилами, заложенными в БД. Этот механизм реализуется с помощью

триггеров, однако аппарат триггеров весьма сложен в отладке и полностью не реализован ни в одной системе.

23.Объектно-ориентированные СУБД

Появление ОО СУБД вызвано потребностями программистов, которым были необходимы средства для хранения объектов, не помещавшихся в оперативной памяти компьютера. Также важна была задача сохранения состояния объектов между повторными запусками прикладной программы. Поэтому, большинство ОО СУБД представляют собой библиотеку, процедуры управления данными которой включаются в прикладную программу [6].

В качестве примера рассмотрим ОО СУБД ObjectStore, которая обеспечивает долговременное хранение в БД объектов, созданных программами на языках C++ и Java. Вся работа с объектами ведется обычными средствами включающего ОО-языка. При этом СУБД как бы расширяет виртуальную память операционной системы. Происходит это следующим образом. Когда прикладная программа обращается к объекту, то ищет его по адресу в оперативной памяти. Нужная страница оперативной памяти может быть вытеснена в виртуальную память (область хранения неиспользуемых страниц оперативной памяти на диске). Если объекта с таким адресом в виртуальной памяти не существует, то операционная система генерирует ошибку. СУБД эту ошибку перехватывает и извлекает объект из БД.

ObjectStore поддерживает транзакции (в один момент времени может существовать только одна транзакция), допускает следующие методы доступа: хеш-таблица для несортированных данных и В-дерево для сортированных, также возможно использование SQL.

24. Стандарт ODMG

ODMG (Object Data Management Group) – консорциум поставщиков ОО БД и других заинтересованных организаций, созданный в 1991 г. Его задачей является разработка стандарта на хранение объектов в БД. Рассмотрим кратко основные положения этого документа.

Стандарт на хранение объектов ODMG разработан на основе трех существующих стандартов: языка управления БД (SQL), языка управления объектами (стандарты OMG – Object Management Group) и стандарты на ОО языки программирования (C++, Smalltalk, Java).

ODMG добавляет возможности взаимодействия с БД в ОО языки программирования: определяются средства долговременного хранения объектов, и расширяется семантика языка, вносятся операторы управления данными. Стандарт состоит из нескольких частей:

1. Объектная модель – унифицированная основа всего стандарта. Она расширяет объектную модель консорциума OMG за счет введения таких свойств как связи и транзакции для обеспечения функциональности, требуемой при взаимодействии с БД. Ключевые концепции объектной модели ODMG:

- ☐ наделение объектов такими свойствами как атрибуты и связи;
- ☐ методы объектов (поведение);
- ☐ множественное наследование;
- ☐ идентификаторы объектов (ключи);
- ☐ определение таких совокупностей объектов как списки, наборы, массивы и т.д.
- ☐ блокировка объектов и изоляция доступа.

2. Язык описания объектов (ODL ☐ Object Definition Language) – средство определения схемы БД (по аналогии с DDL в реляционных СУБД). ODL является расширением IDL (Interface Definition Language ☐ язык описания интерфейсов) модели OMG и предоставляет средства для определения объектных типов, их атрибутов, связей и методов. ODL создает слой абстрактных описаний так, что схема БД становится независима как от языка программирования, так и от СУБД. ODL рассматривает только описание объектных типов данных, не вдаваясь в детали реализации их методов. Это позволяет переносить схему БД между различными ODMG-совместимыми СУБД и языками программирования, а также транслировать ее в другие DDL.

3. Язык объектных запросов (OQL ☐ Object Query Language) – это SQL-подобный декларативный язык, который предоставляет эффективные средства для извлечения объектов из БД, включая высокоуровневые примитивы для наборов объектов и объектных структур. OQL-запросы могут вызываться из ОО языка, точно также из OQL-запросов могут делаться обращения к процедурам, написанным на ОО языке. OQL предоставляет средства обеспечения целостности объектов (вызов объектных методов и использование собственных операторов изменения данных).

4. Связывание с ОО языками. Стандарт связывания с C++, Smalltalk и Java определяет Object Manipulation Language (OML) ☐ язык манипулирования объектами, который расширяет базовые ОО языки средствами

манипулирования и хранения объектов. Также включаются OQL, средства навигации и поддержка транзакций. Каждый ОО язык имеет свой собственный OML, поэтому разработчик остается в одной языковой среде, ему нет необходимости разделять средства программирования и доступа к данным.

33. NFBK (Нормальная Форма Бойса-Кодда)

В алгоритме приведения отношений к 3NF неявно предполагалось, что все отношения содержат один потенциальный ключ. Это не всегда верно. Рассмотрим пример отношения, содержащего два потенциальных ключа.

Пример 3. Пусть требуется хранить данные о поставках деталей некоторыми поставщиками. Предположим, что наименования поставщиков являются уникальными. Кроме того, каждый поставщик имеет свой уникальный номер. Данные о поставках можно хранить в следующем отношении:

Таблица 18. – Отношение ПОСТАВКИ

PNUM	PNAME	DNUM	VOLUME
1	Фирма 1	1	100
1	Фирма 1	2	200
1	Фирма 1	3	300
2	Фирма 2	1	150
2	Фирма 2	2	250
3	Фирма 3	1	1000

Данное отношение содержит два потенциальных ключа – (PNUM, DNUM) и (PNAME, DNUM). Видно, что данные хранятся в отношении с избыточностью – при изменении наименования поставщика, его наименование нужно изменить во всех кортежах, где оно встречается. Попробуем устранить эту аномалию при помощи алгоритма нормализации, описанного ранее. Выявим имеющиеся функциональные зависимости:

-PNUMPNAME – наименование поставщика зависит от номера поставщика;

-PNAMEPNUM – номер поставщика зависит от наименования поставщика;

-(PNUM, DNUM) VOLUME – поставляемое количество зависит от первого ключа отношения;

-(PNUM, DNUM) PNAME – наименование поставщика зависит от первого ключа отношения;

-(PNAME, DNUM) VOLUME – поставляемое количество зависит от второго ключа отношения;

-(PNAME, DNUM) PNUM – номер поставщика зависит от второго ключа отношения.

Данное отношение не содержит неключевых атрибутов, зависящих от части сложного ключа. Действительно, от части сложного ключа зависят атрибуты PNAME и PNUM, но они сами являются ключевыми. Таким образом, отношение находится в 2NF.

Кроме того, отношение не содержит зависимых друг от друга неключевых атрибутов, т.к. неключевой атрибут всего один – VOLUME. Таким образом, показано, что отношение ПОСТАВКИ находится в 3NF.

Таким образом, получается, что описанный ранее алгоритм нормализации неприменим к данному отношению. Очевидно, однако, что аномалия данного отношения устраняется путем декомпозиции его на два следующих отношения:

Таблица 19.– Отношение ПОСТАВЩИКИ

PNUM	PNAME
1	Фирма 1
2	Фирма 2
3	Фирма 3

Таблица 20.– Отношение ПОСТАВКИ2

PNUM	DNUM	VOLUME
------	------	--------

1	1	100
1	2	200
1	3	300
2	1	150
2	2	250
3	1	1000

Отношение находится в нормальной форме Бойса-Кодда (NFBK) тогда и только тогда, когда детерминанты всех функциональных зависимостей являются потенциальными ключами.

Если отношение находится в NFBK, то оно автоматически находится и в 3NF.

Отношение ПОСТАВКИ не находится в NFBK, т.к. имеются зависимости PNUMPNAME и PNAMEPNUM, детерминанты которых не являются потенциальными ключами.

Для того чтобы устранить зависимость от детерминантов, не являющихся потенциальными ключами, необходимо провести декомпозицию, вынося эти детерминанты и зависимые от них части в отдельное отношение. Отношения ПОСТАВЩИКИ и ПОСТАВКИ2, полученные в результате декомпозиции, находятся в NFBK.

Приведенная декомпозиция отношения ПОСТАВКИ на отношения ПОСТАВЩИКИ и ПОСТАВКИ2 не является единственно возможной. Альтернативной декомпозицией является декомпозиция на следующие отношения:

Таблица 21. – Отношение ПОСТАВЩИКИ

PNUM	PNAME
1	Фирма 1
2	Фирма 2
3	Фирма 3

Таблица 22. – Отношение ПОСТАВКИ3

PNAME	DNUM	VOLUME
Фирма 1	1	100
Фирма 1	2	200
Фирма 1	3	300
Фирма 2	1	150
Фирма 2	2	250
Фирма 3	1	1000

На первый взгляд, такая декомпозиция хуже, чем первая. Действительно, наименования поставщиков по-прежнему повторяются, и при изменении наименования поставщика, это наименование придется менять одновременно в нескольких местах сразу в двух отношениях. Кажется, что ситуация стала хуже, чем была до декомпозиции. Однако такое ощущение возникает от того, что наименования поставщиков могут меняться, а номера нет. Если же предположить, что номера поставщиков тоже можно изменять, то первая декомпозиция получается такой же «плохой» как и вторая.

На самом деле никакого противоречия нет. В отношении ПОСТАВКИЗ атрибут PNAME является внешним ключом, служащим для связи с отношением ПОСТАВЩИКИ. Поэтому, при изменении наименования поставщика, это изменение происходит в отношении ПОСТАВЩИКИ и каскадно распространяется на отношение ПОСТАВКИЗ совершенно так, как изменение номера поставщика каскадно распространяется на отношение ПОСТАВКИ2. Поэтому, формально обе декомпозиции совершенно равноправны. В реальной работе разработчик выберет, конечно, первую декомпозицию, но тут важно подчеркнуть, что его выбор основан совсем на других соображениях, не имеющих отношения к формальной теории нормальных форм.

34. 4 NF (Четвертая Нормальная Форма), понятие многозначной зависимости, теорема Фейджина

Понятие четвертой нормальной формы (4NF) рассмотрим на примере.

Пример 4. Пусть требуется учитывать данные об абитуриентах, поступающих в ВУЗ, и при анализе предметной области были выделены следующие требования:

-каждый абитуриент имеет право сдавать экзамены на несколько факультетов одновременно;

-каждый факультет имеет свой список сдаваемых предметов;

-один и тот же предмет может сдаваться на нескольких факультетах;

-абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.

Предположим, что нам требуется хранить данные о том, какие предметы должен сдавать каждый абитуриент. Попробуем хранить данные в одном отношении АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ:

Таблица 23. – Отношение АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ

Абитуриент	Факультет	Предмет
Иванов	Математический	Математика
Иванов	Математический	Информатика
Иванов	Физический	Математика
Иванов	Физический	Физика
Петров	Математический	Математика
Петров	Математический	Информатика

В данный момент в отношении хранится информация о том, что абитуриент Иванов поступает на два факультета – математический и физический, а абитуриент Петров только на математический. Кроме того, можно сделать вывод, что на математическом факультете нужно сдавать математику и информатику, а на физическом математику и физику.

Кажется, что в отношении имеется аномалия обновления, связанная с тем, что дублируются фамилии абитуриентов, наименования факультетов и наименования предметов. Однако эта аномалия легко устраняется стандартным способом: вынесением всех наименований в отдельные отношения, когда в исходном отношении остаются только соответствующие номера. В этом случае получим четыре новых отношения (табл. 24-27):

Таблица 24. – Новое отношение АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ

Номер абитуриента	Номер факультета	Номер предмета
1	1	1
1	1	2
1	2	1
1	2	3
2	1	1
2	1	2

Таблица 25. – Отношение АБИТУРИЕНТЫ

Номер абитуриента	Абитуриент
1	Иванов
2	Петров

Таблица 26. – Отношение ФАКУЛЬТЕТЫ

Номер факультета	Факультет
1	Математический
2	Физический

Таблица 27. – Отношение ПРЕДМЕТЫ

Номер предмета	Предмет
1	Математика
2	Информатика
3	Физика

Теперь каждое наименование встречается только в одном месте. И все-таки, как в исходном, так и в новом отношении имеются аномалии обновления, возникающие при попытке вставить или удалить кортежи.

Аномалия вставки. При попытке добавить в отношение АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ новый кортеж, например (Сидоров, Математический, Математика), мы обязаны добавить также и кортеж (Сидоров, Математический, Информатика), т.к. все абитуриенты математического факультета обязаны иметь один и тот же список сдаваемых предметов. Соответственно, при попытке вставить в новое отношение кортеж (3, 1, 1), мы обязаны вставить в него также и кортеж (3, 1, 2).

Аномалия удаления. При попытке удалить кортеж (Иванов, Математический, Математика), мы обязаны удалить также и кортеж (Иванов, Математический, Информатика) по той же причине, что и для вставки.

Таким образом, вставка и удаление кортежей не может быть выполнена независимо от других кортежей отношения.

Кроме того, если удалить кортеж (Иванов, Физический, Математика), а вместе с ним и кортеж (Иванов, Физический, Физика), то будет потеряна информация о предметах, которые должны сдаваться на физическом факультете.

Декомпозиция отношения АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ для устранения указанных аномалий не может быть выполнена на основе функциональных зависимостей, т.к. это отношение не содержит никаких функциональных зависимостей. Это отношение является полностью ключевым, т.е. ключом отношения является все множество атрибутов. Но ясно, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь описывается понятием многозначной зависимости.

Пусть R – отношение и X, Y, Z – некоторые из его атрибутов (или непересекающиеся множества атрибутов).

Тогда атрибуты (множества атрибутов) Y и Z многозначно зависят от X (обозначается $X \twoheadrightarrow Y|Z$), тогда и только тогда, когда из того, что в отношении R содержатся кортежи $r_1 = (x, y, z_1)$ и $r_2 = (x, y_1, z)$ следует, что в отношении содержится также и кортеж $r_3 = (x, y, z)$.

Меняя местами, кортежи r_1 и r_2 в определении многозначной зависимости, получим, что в отношении R должен содержаться также и кортеж $r_4 = (x, y_1, z_1)$. Таким образом, атрибуты Y и Z , многозначно зависящие от X , ведут себя «симметрично» по отношению к атрибуту.

В отношении АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ имеется многозначная зависимость $\text{ФАКУЛЬТЕТ} \twoheadrightarrow \text{АБИТУРИЕНТ} | \text{ПРЕДМЕТ}$.

Словами это можно выразить так – для каждого факультета (для каждого значения из X) каждый поступающий на него абитуриент (значение из Y) сдает

один и тот же список предметов (набор значений из Z), и для каждого факультета (для каждого значения из X) каждый сдаваемый на факультете экзамен (значение из Z) сдается одним и тем же списком абитуриентов (набор значений из Y). Именно наличие этой зависимости не позволяет независимо вставлять и удалять кортежи. Кортежи обязаны вставляться и удаляться одновременно целыми наборами.

Если в отношении R имеются не менее трех атрибутов X, Y, Z и есть функциональная зависимость $X \rightarrow Y$, то есть и многозначная зависимость $X \rightarrow Y|Z$.

Таким образом, понятие многозначной зависимости является обобщением понятия функциональной зависимости.

Многозначная зависимость $X \twoheadrightarrow Y|Z$ называется нетривиальной многозначной зависимостью, если не существует функциональных зависимостей $X \rightarrow Y$ и $X \rightarrow Z$.

В отношении АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ имеется именно нетривиальная многозначная зависимость $\text{ФАКУЛЬТЕТ} \twoheadrightarrow \text{АБИТУРИЕНТ}|\text{ПРЕДМЕТ}$. В силу нетривиальности этой зависимости мы не можем воспользоваться теоремой Хеза для декомпозиции отношения.

Теорема Фейджина. Пусть X, Y, Z – непересекающиеся множества атрибутов отношения $R(X, Y, Z)$. Декомпозиция отношения R на проекции $R_1 = R[X, Y]$ и $R_2 = R[X, Z]$ будет декомпозицией без потерь тогда и только тогда, когда имеется многозначная зависимость $X \twoheadrightarrow Y|Z$.

Если зависимость $X \twoheadrightarrow Y|Z$ является тривиальной, т.е. существует одна из функциональных зависимостей $X \rightarrow Y$ или $X \rightarrow Z$, то получаем теорему Хеза.

Отношение R находится в четвертой нормальной форме (4NF) тогда и только тогда, когда отношение находится в НФБК и не содержит нетривиальных многозначных зависимостей.

Отношение АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ находится в НФБК, но не в 4NF. Согласно теореме Фейджина, это отношение можно без потерь декомпозировать на отношения ФАКУЛЬТЕТЫ-АБИТУРИЕНТЫ и ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ:

Таблица 28. – Отношение ФАКУЛЬТЕТЫ-АБИТУРИЕНТЫ

Факультет	Абитуриент
-----------	------------

Математический	Иванов
Физический	Иванов
Математический	Петров

Таблица 29. – Отношение ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ

Факультет	Предмет
Математический	Математика
Математический	Информатика
Физический	Математика
Физический	Физика

В полученных отношениях устранены аномалии вставки и удаления, характерные для отношения АБИТУРИЕНТЫ-ФАКУЛЬТЕТЫ-ПРЕДМЕТЫ.

Заметим, что полученные отношения остались полностью ключевыми, и в них по-прежнему нет функциональных зависимостей.

Отношения с нетривиальными многозначными зависимостями возникают, как правило, в результате естественного соединения двух отношений по общему полю, которое не является ключевым ни в одном из отношений. Фактически это приводит к попытке хранить в одном отношении информацию о двух независимых сущностях.

В качестве еще одного примера можно привести ситуацию, когда сотрудник может иметь много работ и много детей. Хранение информации о работах и детях в одном отношении приводит к возникновению нетривиальной многозначной зависимости $\text{Работник} \twoheadrightarrow \text{Работа} | \text{Дети}$.

35. 5 NF, понятие зависимости соединения

Функциональные и многозначные зависимости позволяют произвести декомпозицию исходного отношения без потерь на две проекции. Можно, однако, привести примеры отношений, которые нельзя декомпозировать без потерь ни на какие две проекции.

Рассмотрим следующее отношение R :

Таблица 30. – Отношение

X	Y	Z
1	1	2
1	2	1
2	1	1
1	1	1

Все проекции отношения R , включающие по два атрибута, имеют вид $R_1 = R[X, Y]$, $R_2 = R[X, Z]$, $R_3 = R[Y, Z]$ и приведены в табл. 31-33.

Таблица 31. Проекция

X	Y
1	1
1	2
2	1

Таблица 32. Проекция

X	Z
1	2
1	1
2	1

Таблица 33. Проекция

Y	Z
1	2
2	1
1	1

Можно заметить, что отношение R не восстанавливается ни по одному из парных соединений $R_1 \text{ join } R_2$, $R_1 \text{ join } R_3$ или $R_2 \text{ join } R_3$. Действительно, соединение $R_1 \text{ join } R_2$ имеет вид, представленный в табл. 34:

Таблица 34. – Отношение $R_1 \text{ join } R_2$

X	Y	Z
1	1	2
1	1	1
1	2	2
1	2	1
2	1	1

Серым цветом выделен лишний кортеж, отсутствующий в отношении R . Аналогично (в силу соображений симметрии) и другие парные соединения не восстанавливают отношения R .

Однако отношение R восстанавливается соединением всех трех проекций:

$$R_1 \text{ join } R_2 \text{ join } R_3.$$

Это говорит о том, что между атрибутами этого отношения также имеется некоторая зависимость, но эта зависимость не является ни функциональной, ни многозначной зависимостью.

Пусть R является отношением, а A, B, \dots, Z — произвольными (возможно пересекающимися) подмножествами множества атрибутов отношения R . Тогда отношение R удовлетворяет зависимости соединения $*$ (A, B, \dots, Z) тогда и только тогда, когда оно равносильно соединению всех своих проекций с подмножествами атрибутов A, B, \dots, Z , т.е. $R = R[A] \text{ join } R[B] \text{ join } \dots \text{ join } R[Z]$.

Можно предположить, что отношение R в рассматриваемом примере удовлетворяет следующей зависимости соединения:

$$* (XY, XZ, YZ)$$

Утверждать, что это именно так сразу нельзя, т.к. определение зависимости соединения должно выполняться для любого состояния отношения R , а не только для состояния, приведенного в примере.

Зависимость соединения является обобщением понятия многозначной зависимости, и теорема Фейджина для зависимости соединения может быть переформулирована следующим образом:

Отношение $R(X, Y, Z)$ удовлетворяет зависимости соединения $*$ (XY, XZ) тогда и только тогда, когда имеется многозначная зависимость $X \twoheadrightarrow Y|Z$.

Это значит, что многозначная зависимость является частным случаем зависимости соединения, т.е., если в отношении имеется многозначная зависимость, то имеется и зависимость соединения. Обратное, конечно, неверно.

Зависимость соединения $*$ (A, B, \dots, Z) называется нетривиальной зависимостью соединения, если выполняется два условия:

1. Одно из множеств атрибутов A, B, \dots, Z не содержит потенциального ключа отношения R .

2. Ни одно из множеств атрибутов не совпадает со всем множеством атрибутов отношения R .

Для лучшего понимания сформулируем это определение так же и в отрицательной форме:

Зависимость соединения $*$ (A, B, \dots, Z) называется тривиальной зависимостью соединения, если выполняется одно из условий:

1. Либо все множества атрибутов A, B, \dots, Z содержат потенциальный ключ отношения R .

2. Либо одно из множеств атрибутов совпадает со всем множеством атрибутов отношения R .

Отношение R находится в пятой нормальной форме (5NF) тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

Определения 5NF может стать более понятным, если сформулировать его в отрицательной форме:

Отношение R не находится в 5NF, если в отношении найдется нетривиальная зависимость соединения.

Таким образом, в приведенном примере не зная ничего о том, какие потенциальные ключи имеются в отношении и как взаимосвязаны атрибуты, нельзя делать выводы о том, находится ли данное отношение в 5NF (как, впрочем, и в других нормальных формах).

По данному конкретному примеру можно только предположить, что отношение не находится в 5NF и что анализ предметной области позволил выявить следующие зависимости атрибутов:

а) отношение является полностью ключевым (т.е. потенциальным ключом отношения является все множество атрибутов);

б) имеется следующая зависимость (довольно странная, с практической точки зрения): если в отношении содержатся кортежи $r_1 = (x, y, z_1)$, $r_2 = (x, y_1, z)$, $r_3 = (x_1, y, z)$, то отсюда следует, что в отношении содержится также и кортеж $r_4 = (x, y, z)$.

Тогда можно доказать [5], что при наличии ограничений а) и б) отношение находится в 4НФ, но не в 5НФ.

36. Общий алгоритм приведения к 5 NF)))))))))))

Шаг 4))) (приведение к NFBK). Если имеются отношения, содержащие несколько потенциальных ключей, то необходимо проверить, имеются ли функциональные зависимости, детерминанты которых не являются потенциальными ключами. Если такие функциональные зависимости имеются, то необходимо провести дальнейшую декомпозицию отношений. Те атрибуты, которые зависят от детерминантов, не являющихся потенциальными ключами, выносятся в отдельное отношение вместе с детерминантами.

Шаг 5 (приведение к 4NF). Если в отношениях обнаружены нетривиальные многозначные зависимости, то необходимо провести декомпозицию для исключения таких зависимостей.

Шаг 6 (приведение к 5NF). Если в отношениях обнаружены нетривиальные зависимости соединения, то необходимо провести декомпозицию для исключения и таких зависимостей.

Подводя итоги, отметим, что обобщением 3NF на случай, когда отношение имеет более одного потенциального ключа, является нормальная форма Бойса-Кодда.

Отношение находится в NFBK тогда и только тогда, когда детерминанты всех функциональных зависимостей являются потенциальными ключами.

Нормализация отношений вплоть до NFBK основывается на понятии функциональной зависимости и теореме Хеза, гарантирующей, что декомпозиция будет происходить без потерь информации.

Дальнейшая нормализация связана с обобщением понятия функциональной зависимости.

Атрибуты (множества атрибутов) и многозначно зависят от R тогда и только тогда, когда из того, что в отношении содержатся кортежи t и s , следует, что в отношении содержится также и кортеж k .

Корректность дальнейшей декомпозиции основывается на теореме Фейджина, которая говорит о том, что декомпозиция отношения на две проекции является декомпозицией без потерь тогда и только тогда, когда в отношении имеется некоторая многозначная зависимость.

Если в отношении имеется функциональная зависимость, то автоматически имеется и тривиальная многозначная зависимость, определяемая этой функциональной зависимостью.

Многозначная зависимость называется нетривиальной многозначной зависимостью, если не существует функциональных зависимостей и .

Отношение находится в четвертой нормальной форме (4NF) тогда и только тогда, когда отношение находится в NFBK и не содержит нетривиальных многозначных зависимостей.

Имеют также место зависимости специального вида, когда отношение не может быть подвергнуто декомпозиции без потерь на две проекции, но может быть декомпозировано на большее число проекций. Такие зависимости называются зависимостями соединения и являются обобщением понятия многозначной зависимости.

Отношение находится в пятой нормальной форме (5NF) тогда и только тогда, когда любая имеющаяся зависимость соединения является тривиальной.

37. Реляционная алгебра: операторы объединения, пересечения, разности

Реляционная алгебра \square это коллекция операций, которые принимают отношения в качестве операндов и возвращают отношение в качестве результата. Первая версия этой алгебры была определена Коддом и включала восемь операций, которые подразделялись на две группы с четырьмя операциями каждая.

1. Традиционные операции с множествами \square объединение, пересечение, разность и декартово произведение (все они были немного модифицированы с учетом того факта, что их операндами являются именно отношения, а не произвольные множества).

2. Специальные реляционные операции, такие как выборка, проекция, соединение и деление.

В реализациях конкретных реляционных СУБД сейчас не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SEQUEL(SQL \square Structured Query Language), который представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении.

Современные языки манипулирования данными SQL, QBE, ISBL и др., используемые в СУБД, реализуют широкий набор операций:

- ☐ операции с данными: включение, модификация, удаление данных;
- ☐ операции обработки данных:
- ☐ арифметические выражения (вычисления, сравнения);
- ☐ команды присваивания и печати;
- ☐ агрегатные функции.

Агрегатными функциями называются функции, применяемые к доменам отношений для вычисления единственного значения. Например, определение максимального или минимального значения домена, определение суммы элементов домена, определение среднего значения домена и т.п.

Оператор объединения (*union*). Для двух РТ R1 и R2 одинаковой арности и с совпадающими типами полей формируется РТ R с записями, входящими хотя бы в одну исходную РТ.

$$R = R1 \text{ union } R2.$$

Пример 6. Объединить две РТ: R1 и R2 в одну РТ R.

R1:	<table><tr><td>A</td><td>B</td></tr><tr><td>a</td><td>4</td></tr><tr><td>b</td><td>6</td></tr></table>	A	B	a	4	b	6	R2:	<table><tr><td>A</td><td>B</td></tr><tr><td>c</td><td>4</td></tr><tr><td>d</td><td>8</td></tr></table>	A	B	c	4	d	8	Ответ R:	<table><tr><td>A</td><td>B</td></tr><tr><td>a</td><td>4</td></tr><tr><td>b</td><td>6</td></tr><tr><td>c</td><td>4</td></tr><tr><td>d</td><td>8</td></tr></table>	A	B	a	4	b	6	c	4	d	8
A	B																										
a	4																										
b	6																										
A	B																										
c	4																										
d	8																										
A	B																										
a	4																										
b	6																										
c	4																										
d	8																										

Оператор пересечения (*intersection*). Для двух РТ R1 и R2 одинаковой арности и с совпадающими типами полей формируется РТ R с записями, входящими в обе РТ.

$$R = R1 \text{ intersection } R2.$$

Пример 8. Выполнить пересечение таблиц R1 и R2, результат записать в таблицу R.

R1:	<table><tr><td>A</td><td>B</td></tr><tr><td>a</td><td>4</td></tr><tr><td>b</td><td>6</td></tr></table>	A	B	a	4	b	6	R2:	<table><tr><td>A</td><td>B</td></tr><tr><td>a</td><td>5</td></tr><tr><td>b</td><td>6</td></tr></table>	A	B	a	5	b	6	Ответ R:	<table><tr><td>A</td><td>B</td></tr><tr><td>b</td><td>6</td></tr></table>	A	B	b	6
A	B																				
a	4																				
b	6																				
A	B																				
a	5																				
b	6																				
A	B																				
b	6																				

Оператор *вычитание (difference)*. Для двух РТ R1 и R2 одинаковой арности и с совпадающими типами полей формируется РТ R с записями, содержащимися в уменьшаемом – таблице R1, но отсутствующими в вычитаемом – таблице R2.

$$R=R1 \text{ difference } R2.$$

Пример 7. Вычесть из РТ R1 таблицу R2, результат записать в РТ R.

R1:	<table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>4</td></tr><tr><td>b</td><td>6</td></tr></table>	A	B	a	4	b	6	R2:	<table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>3</td></tr><tr><td>b</td><td>6</td></tr><tr><td>c</td><td>6</td></tr></table>	A	B	a	3	b	6	c	6	Ответ R:	<table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>4</td></tr></table>	A	B	a	4
A	B																						
a	4																						
b	6																						
A	B																						
a	3																						
b	6																						
c	6																						
A	B																						
a	4																						

38. Реляционная алгебра: операторы проектирования и выбора (пример комбинированного запроса)

Оператор проектирование (*proj*). Из РТ R1 оператор выбирает заданные типы полей и размещает в указанном порядке в таблице R

$$R = proj\ a_1, \dots a_k(R1).$$

Пример 9. Из РТ РАЗРАБОТЧИКИ (R1) выбрать поля ФИО и Г_Р.

$$R = proj\ \text{ФИО}, \text{Г_Р} (R1).$$

Ответ R:

ФИО	Г_Р
Белов А.	1940
Крылов Г.	1962
Фатов Р.	1964
Белов А.	1953
Крылов Г.	1964

Оператор выбор (*sel*). Из РТ R1 по заданному критерию выбирается требуемая совокупность записей. В частном случае результатом выборки может быть и пустое множество.

$$R = sel\ (F)\ (R1), \text{ где } F \text{ — критерий выбора.}$$

Критерии выбора формулируются на основе операторов двух типов:

- ☐ операторы сравнения: =, <>, <, >, <=, >=.
- ☐ логические операторы: И (AND), ИЛИ (OR), НЕ (NOT).

Пример 10. В БД «Разработчики проектов» выбрать данные о Белове А.

$$R = sel\ (\text{ФИО} = \text{'Белов А.'}) (R1).$$

Ответ R:

N_R	ФИО	Г_Р	Стаж
R1	Белов А.	1940	21
R4	Белов А.	1953	21

Пример 11. В БД «Разработчики проектов» выбрать программистов выше третьей категории, работающих на Pascal.

$R = sel (Я_П = 'Pas' \text{ AND } \text{Категория} < 3) (R2).$

Ответ R:

N_A	N_R_A	Я_П	Категория
A1	R4	Pas	1
A2	R2	Pas	2

Пример. 12. Выполним комбинированный запрос с операторами proj и sel. Пусть необходимо выбрать ФИО разработчиков со стажем от 12 лет, год рождения которых 1960 и позже.

Для запроса необходимо использовать РТ РАЗРАБОТЧИКИ, тогда запрос можно записать в виде:

$R = proj \text{ ФИО } sel (\text{Стаж} \geq 12 \text{ AND } \text{Г_Р} \geq 1960) (R1).$

Ответ R:

ФИО
Крылов Г.

39. Реляционная алгебра: оператор соединения, примеры запросов с соединением по одному и по двум полям

Оператор *соединение* (*join*). На основе двух реляционных таблиц R1 и R2 имеющих одно или несколько полей идентичных типов, формируется реляционная таблица R, состоящая из записей, каждая из которых является конкатенацией, т.е. соединением в одну тех записей таблиц R1 и R2, у которых совпадают значения полей всех идентичных типов.

Следовательно, соединение может быть реализовано двумя способами:

- ☐ по одному полю;
- ☐ по нескольким полям.

Если в отношениях R1 и R2 типы полей, по которым выполняется соединение, имеют идентичные имена, то соединение называется естественным. В этом случае в результирующей записи общий тип поля помещается только один раз. Если имена различны, то в результирующей таблице остается каждый тип поля.

$$R = R1 \text{ join } R2.$$

Пример 13. Соединить таблицы R1 и R2 с одним общим полем.

R1:	<table><tr><th>A</th><th>B</th></tr><tr><td>d</td><td>3</td></tr><tr><td>h</td><td>7</td></tr><tr><td>y</td><td>4</td></tr></table>	A	B	d	3	h	7	y	4	R2:	<table><tr><th>B</th><th>C</th></tr><tr><td>3</td><td>a</td></tr><tr><td>3</td><td>b</td></tr><tr><td>7</td><td>a</td></tr><tr><td>9</td><td>c</td></tr></table>	B	C	3	a	3	b	7	a	9	c	Ответ R:	<table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>d</td><td>3</td><td>a</td></tr><tr><td>d</td><td>3</td><td>b</td></tr><tr><td>h</td><td>7</td><td>a</td></tr></table>	A	B	C	d	3	a	d	3	b	h	7	a
A	B																																		
d	3																																		
h	7																																		
y	4																																		
B	C																																		
3	a																																		
3	b																																		
7	a																																		
9	c																																		
A	B	C																																	
d	3	a																																	
d	3	b																																	
h	7	a																																	

Пример 14. Соединить таблицы R1 и R2 с двумя общими полями

R1:	<table><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>d</td><td>3</td><td>a</td></tr></table>	A	B	C	d	3	a	R2:	<table><tr><td>B</td><td>C</td><td>D</td></tr><tr><td>3</td><td>a</td><td>c</td></tr></table>	B	C	D	3	a	c	Ответ R:	<table><tr><td>A</td><td>B</td><td>C</td><td>D</td></tr><tr><td>d</td><td>3</td><td>a</td><td>c</td></tr></table>	A	B	C	D	d	3	a	c
A	B	C																							
d	3	a																							
B	C	D																							
3	a	c																							
A	B	C	D																						
d	3	a	c																						

h	7	b
y	4	a
g	2	c

3	a	d
4	a	e

d	3	a	d
y	4	a	e

Пример 15. Рассмотрим запрос с соединением по одному полю к БД «Разработчики проектов». Пусть необходимо вывести названия проектов, созданных в период с 1970 по 1984 гг. включительно, а также ФИО и стаж разработчиков.

Алгоритм реализации:

1. Выделить названия РТ, задействованных в реализации запроса: ПРОЕКТЫ (R3) и РАЗРАБОТЧИКИ (R1).

2. Из R3 выбрать записи по условию запроса:

$RT1 = sel (\Gamma_Cозд \geq 1970 \text{ AND } \Gamma_Cозд \leq 1984)(R3).$

RT1:

N_P	Назв_P	N_R_P	$\Gamma_Cозд$
P1	ПП-1	R5	1982
P2	ПП-2	R2	1984

3. Соединить RT1 и R1 по общему полю N_R_P (в RT1) и N_R (в R1):

$RT2 = RT1 \text{ join } R1.$

Результирующая таблица RT2 будет иметь вид:

RT2:

N_P	Назв_P	N_R_P (N_R)	$\Gamma_Cозд$	ФИО	Γ_P	Стаж
P1	ПП-1	R5	1982	Крылов Г.	1964	10
P2	ПП-2	R2	1984	Крылов Г.	1962	17

4) Выбрать из RT2 Назв_P, ФИО и Стаж разработчиков:

$R = proj \text{ Назв_Р, ФИО, Стаж } (RT2).$

Результирующая таблица R имеет вид:

R:

Назв_Р	ФИО	Стаж
ПР-1	Крылов Г.	10
ПР-2	Крылов Г.	17

Общий алгоритм реализации запроса можно представить в виде:

$R = proj \text{ Назв_Р, ФИО, Стаж } (sel \text{ } (\Gamma_Созд \geq 1970 \text{ AND } \Gamma_Созд \leq 1984) (R3) join R1).$

Пример 16. Рассмотрим запрос с соединением по нескольким полям. В БД «Разработчики проектов» выбрать Назв_Р, N_Р и ФИО разработчиков, выступающих и как руководители ВТК, и как программисты в них.

Алгоритм реализации:

1) Выделить названия РТ, задействованных в реализации запроса: ПРОЕКТЫ (R3), РАЗРАБОТЧИКИ (R1), ВТК (R4), Составы_ВТК (R5), ПРОГРАММИСТЫ (R2).

2) Для нахождения разработчиков проектов, выступающих одновременно и как руководители ВТК, объединить таблицы R3 и R4 по общему полю N_Р_Р (в R3) и N_рук_ВТК (в R4).

$RT1 = R3 join R4.$

RT1:

N_Р	Назв_Р	N_Р_Р (N_рук_ВТК) *	Г.Созд .	N_ВТК	Назв_ВТК	N_Комн
P1	ПР-1	R5	1982	B1	Луч	12
P2	ПР-2	R2	1984	B3	Взлет	12

P3	ПП-3	R2	1987	B3	Взлет	12
P4	ПП-4	R3	1985	B2	Стрела	18

* далее будем считать для упрощения алгоритма, что общие поля при соединении имеют одинаковые имена и не повторяются в результирующем запросе.

3) Для экономии памяти ПК и увеличения быстродействия выбрать из RT1 только участвующие в запросе поля:

$RT2 = proj \text{ Назв_P, N_R_P, N_BTK } (RT1).$

RT2:

Назв_P	N_R_P	N_BTK
ПП-1	R5	B1
ПП-2	R2	B3
ПП-3	R2	B3
ПП-4	R3	B2

4) Для определения разработчиков-руководителей ВТК, являющихся одновременно и программистами, соединить RT2 и R2 по одному общему полю N_R_P (в RT2) и N_R_A (в R2):

$RT3 = RT2 \text{ join } R2.$

RT3:

Назв_P	N_R_P	N_BTK	N_A	Я_П	Категория
ПП-1	R5	B1	A3	Pas	3
ПП-2	R2	B3	A2	C	2
ПП-2	R2	B3	A5	Pas	2
ПП-3	R2	B3	A2	C	2
ПП-3	R2	B3	A5	Pas	2

5) Выберем из RT3 только участвующие в запросе поля:

$$RT4 = proj \text{ Назв_P, N_R_P, N_BTK, N_A } (RT3).$$

RT4:

Назв_P	N_R_P	N_BTK	N_A
ПП-1	R5	B1	A3
ПП-2	R2	B3	A2
ПП-2	R2	B3	A5
ПП-3	R2	B3	A2
ПП-3	R2	B3	A5

6) Для определения разработчиков-руководителей ВТК, являющихся программистами именно в своих ВТК, сделать соединение таблиц RT4 и R5 по двум полям: N_BTK и N_A

$$RT5 = RT4 \text{ join } R5.$$

RT5:

Назв_P	N_R_P	N_BTK	N_A_BTK
ПП-1	R5	B1	A3
ПП-2	R2	B3	A5
ПП-3	R2	B3	A5

7) Выбрать из RT5 участвующие в запросе поля:

$$RT6 = proj \text{ Назв_P, N_R_P } (RT5).$$

RT6:

Назв_P	N_R_P
ПП-1	R5

ПР-2	R2
ПР-3	R2

8) Для полного ответа на запрос объединим таблицы RT6 и R1 по общему полю N_R_P (в RT6) и N_R (в R1) с последующим выделением полей, указанных в запросе:

$$RT7 = RT6 \text{ join } R1.$$

RT7:

Назв_P	N_R_P	ФИО	Г_P	Стаж
ПР-1	R5	Крылов Г.	1964	10
ПР-2	R2	Крылов Г.	1962	17
ПР-3	R2	Крылов Г.	1962	17

9) Выберем из RT7 участвующие в запросе поля:

$$RT8 = \text{proj Назв_P, N_R, ФИО (RT7)}.$$

RT8:

Назв_P	N_R_P	ФИО
ПР-1	R5	Крылов Г.
ПР-2	R2	Крылов Г.
ПР-3	R2	Крылов Г.

Таким образом, алгоритм ответа на запрос можно записать в виде:

$$\text{proj Назв_P, N_R_P, ФИО (proj Назв_P, N_R_P}$$

$(proj \text{ Ha3B_P, N_R_P, N_BTK, N_A}$

4

$(proj \text{ Ha3B_P, N_R_P (R3 join R4) join R2) join R5) join R1}).$

2

1

3

5

7

40. Реляционная алгебра: операторы соединения по условию и умножения

Оператор соединения по условию. Оператором *join* из двух РТ R1 и R2 формируется результирующая R, если выполняется заданное условие V:

$$V = P_{1i} \ G \ P_{2j},$$

где G — арифметический оператор сравнения, выбираемый из множества =, <>, <, >, <=, >=;

P_{1i}, P_{2j} — типы полей в реляционных таблицах R1 и R2 соответственно.

Для оператора G типа равенство оператор *join* называется эквисоединением.

$$R = R1 \ join \ R2.$$

$$P_{1i} \ G \ P_{2j}$$

Пример 17. Соединить таблицы R1 и R2 по условию $B = D$:

$$R = R1 \ join \ R2.$$

$$B = D$$

R1:	<table><tr><td>A</td><td>B</td><td>C</td></tr><tr><td>a</td><td>b</td><td>c</td></tr><tr><td>d</td><td>u</td><td>p</td></tr><tr><td>g</td><td>e</td><td>z</td></tr></table>	A	B	C	a	b	c	d	u	p	g	e	z	R2:	<table><tr><td>D</td><td>E</td></tr><tr><td>a</td><td>u</td></tr><tr><td>e</td><td>k</td></tr></table>	D	E	a	u	e	k	Ответ R:	<table><tr><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr><tr><td>g</td><td>e</td><td>z</td><td>e</td><td>k</td></tr></table>	A	B	C	D	E	g	e	z	e	k
A	B	C																															
a	b	c																															
d	u	p																															
g	e	z																															
D	E																																
a	u																																
e	k																																
A	B	C	D	E																													
g	e	z	e	k																													

Оператор умножение (*product*)

Для двух РТ R1 и R2 соответственно арности K1 и K2 формируется результирующая РТ R арности $(K1 + K2)$, записи которой представляют собой конкатенацию каждой записи из таблицы R1 с каждой записью из таблицы R2. В таблице R имена полей

формируются из двух частей, разделенных точкой. Префиксом имени поля принимается имя таблицы R1 или R2, в зависимости от того, из какой таблицы взято значение поля, а афиксом соответствующие имена полей из этой таблицы.

$$R = R1 \text{ product } R2.$$

Пример 18.

R1:	<table><tr><td>A</td><td>B</td></tr><tr><td>b</td><td>4</td></tr><tr><td>d</td><td>7</td></tr></table>	A	B	b	4	d	7	R2:	<table><tr><td>C</td><td>D</td></tr><tr><td>c</td><td>4</td></tr><tr><td>d</td><td>R</td></tr></table>	C	D	c	4	d	R	Отвѣт R:	<table><tr><td>R1.A</td><td>R1.B</td><td>R2.C</td><td>R2.D</td></tr><tr><td>b</td><td>4</td><td>c</td><td>4</td></tr><tr><td>b</td><td>4</td><td>d</td><td>R</td></tr><tr><td>d</td><td>7</td><td>c</td><td>4</td></tr><tr><td>d</td><td>7</td><td>d</td><td>R</td></tr></table>	R1.A	R1.B	R2.C	R2.D	b	4	c	4	b	4	d	R	d	7	c	4	d	7	d	R
A	B																																				
b	4																																				
d	7																																				
C	D																																				
c	4																																				
d	R																																				
R1.A	R1.B	R2.C	R2.D																																		
b	4	c	4																																		
b	4	d	R																																		
d	7	c	4																																		
d	7	d	R																																		

Запросы с оператором умножения используются в ответах на запросы, требующие сравнения значений полей различных таблиц.

Пример 19. В БД найти названия и годы создания проектов, разработанных до года рождения Фатова Р. с помощью оператора умножения, как одного из возможных вариантов решения данной задачи.

Алгоритм реализации:

1) Выделить названия РТ, задействованных в реализации запроса: РАЗРАБОТЧИКИ (R1), ПРОЕКТЫ (R3).

2) Сформировать таблицу с годом рождения Фатова Р. Для этого сначала выделить запись о нем из таблицы R1, а затем выбрать поле Г_Р

$$RT1 = sel (ФИО = 'Фатов Р.') (R1).$$

RT1:	N_R	ФИО	Г_Р	Стаж
	R3	Фатов О.	1964	11

$$RT2 = proj \Gamma_P (RT1)$$

RT2:

Γ_P
1964

3) Из R3 выделим поля Назв_P и $\Gamma_Cозд$:

$$RT3 = proj \text{ Назв_P, } \Gamma_Cозд (R3).$$

RT3:

Назв_P	$\Gamma_Cозд$
ПР-1	1982
ПР-2	1984
ПР-1	1960
ПР-3	1987
ПР-4	1985

4) Для реализации сравнения выполнить операцию перемножения таблиц RT2 и RT3:

$$RT4 = RT2 \text{ product } RT3.$$

RT4:

RT2. Γ_P	RT3.Назв_P	RT3. $\Gamma_Cозд$
1964	ПР-1	1982
1964	ПР-2	1984
1964	ПР-1	1960
1964	ПР-3	1987
1964	ПР-4	1985

5) Из RT4 выберем запись по критерию запроса:

$$RT5 = sel (RT3. \Gamma_Cозд < RT2. \Gamma_P) (RT4)$$

RT5:

RT2.Γ_P	RT3.Назв_P	RT3.Γ_Созд
1964	ПП-1	1960

б) Из RT5 выбрать поля, необходимые и достаточные для ответа на запрос:

$$RT6 = proj RT3.Назв_P, RT3.Γ_Созд (RT5).$$

RT6:

RT3.Назв_P	RT3.Γ_Созд
ПП-1	1960

Таким образом, алгоритм ответа на запрос можно записать в виде:

$$proj RT3.Назв_P, RT3.Γ_Созд (sel (RT3.Γ_Созд < RT2.Γ_P)$$

$$6 \qquad \qquad \qquad 5$$

$$(proj \Gamma.P. (sel(ФИО = 'Фатов P.') (R1)) product$$

$$2 \qquad 1 \qquad \qquad \qquad 4$$

$$(proj Назв_P, \Gamma_Созд (R3))).$$

41.Реляционная алгебра: оператор деления

10) Оператор *деление (division)*

Для двух РТ соответственно делимого R1 и делителя R2 таких, что:

а) каждому типу поля в делителе найдется соответствующий тип поля в делимом;

б) в экземплярах записей делимого имеются типы полей, отсутствующих в делителе, отдельные значения которых в этих экземплярах записей последовательно конкатенированы с каждым экземпляром записи в делителе, будет получена таблица-частное R, которая:

а) включает типы полей из числа содержащихся в делимом, но отсутствующих в делителе;

б) включает значения полей каждого экземпляра записи частного R лишь в том случае, если они в экземплярах записей делимого конкатенированы с каждым экземпляром записи в делителе.

$$R = R1 \text{ division } R2.$$

.

Пример 20.

R1:	<table><tr><th>D</th><th>E</th><th>F</th></tr><tr><td>A</td><td>h</td><td>3</td></tr><tr><td>C</td><td>a</td><td>7</td></tr><tr><td>B</td><td>a</td><td>7</td></tr><tr><td>B</td><td>e</td><td>4</td></tr><tr><td>A</td><td>e</td><td>4</td></tr><tr><td>A</td><td>a</td><td>7</td></tr><tr><td>B</td><td>h</td><td>3</td></tr><tr><td>C</td><td>e</td><td>4</td></tr></table>	D	E	F	A	h	3	C	a	7	B	a	7	B	e	4	A	e	4	A	a	7	B	h	3	C	e	4	R2:	<table><tr><th>E</th><th>F</th></tr><tr><td>h</td><td>3</td></tr><tr><td>a</td><td>7</td></tr><tr><td>e</td><td>4</td></tr></table>	E	F	h	3	a	7	e	4	Ответ R:	<table><tr><th>D</th></tr><tr><td>A</td></tr><tr><td>B</td></tr></table>	D	A	B
D	E	F																																									
A	h	3																																									
C	a	7																																									
B	a	7																																									
B	e	4																																									
A	e	4																																									
A	a	7																																									
B	h	3																																									
C	e	4																																									
E	F																																										
h	3																																										
a	7																																										
e	4																																										
D																																											
A																																											
B																																											

Примечания:

1) В таблице R лишь одно поле D, так как оно входит в делимое и отсутствует в делителе;

2) Значения A и B входят в R, так как в делимом они соответственно конкатенированы с каждой из трех записей делителя (см. в табл. R записи №№ 1, 5, 6 для D = A и №№ 3, 4, 7 для D = B);

3) Значение C отсутствует в R, так как в R1 нет записи Ch3

42.CASE – технологии: история возникновения

Проектирование ИС охватывает три области:

1. Область проектирования объектов данных, которые будут реализованы в БД (ИО).
2. Разработка программ, экранных форм и отчетов, запросов (ПО).
3. Учет конкретной среды или технологий (топология сети, конфигурация аппаратных средств и т.д.) (ТО).

Существует два способа проектирования: ручной и автоматизированный (с использованием CASE-технологий).

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности ИС, создаваемых в различных областях экономики. Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

- ☐ сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
- ☐ наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

- ☐ уникальность ИС или отсутствие прямых аналогов, ограничивающие возможность использования каких-либо типовых проектных решений и прикладных систем;
- ☐ наследуемость ИС или необходимость интеграции существующих и вновь разрабатываемых приложений;
- ☐ функционирование в неоднородной среде на нескольких аппаратных платформах;
- ☐ разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;
- ☐ существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта объект проектирования должен быть, прежде всего, адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели ИС. Накопленный опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

Ручная разработка обычно порождает следующие проблемы:

- ☐ неадекватная спецификация требований;
- ☐ неспособность обнаруживать ошибки в проектных решениях;
- ☐ низкое качество документации, снижающее эксплуатационные качества;
- ☐ затяжной цикл проектирования и неудовлетворительные результаты тестирования.

С другой стороны, разработчики ИС исторически всегда стояли последними в ряду тех, кто использовал компьютерные технологии для

повышения качества, надежности и производительности в своей собственной работе (феномен «сапожника без сапог»).

Перечисленные проблемы способствовали появлению программно-технологических средств специального класса – CASE-средств, реализующих CASE-технологии создания и сопровождения ИС. Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь ПО, в последнее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом.

43. Понятие CASE – технологии и CASE – средств, достоинства и недостатки использования

Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО и БД, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

Появлению CASE-технологии и CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д. Кроме того, появлению CASE-технологии способствовали и такие факторы, как:

- ☐ подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;
- ☐ широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;
- ☐ внедрение сетевых технологий, предоставивших возможность объединения усилий отдельных исполнителей в единый процесс проектирования путем использования, например, разделяемой БД, содержащей необходимую информацию о проекте.

CASE-технология – это методология проектирования ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методологиях структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

Несмотря на все потенциальные возможности CASE-средств, существует множество примеров их неудачного внедрения, в результате которых CASE-средства становятся «полочным» ПО (shelfware). Причина такого явления кроется в следующих особенностях:

- ☐ CASE-средства не обязательно дают немедленный эффект; он может быть получен только спустя какое-то время;
- ☐ реальные затраты на внедрение CASE-средств обычно намного превышают затраты на их приобретение;
- ☐ CASE-средства обеспечивают возможности для получения существенной выгоды только после успешного завершения процесса их внедрения.

Для успешного внедрения CASE-средств организация должна обладать следующими качествами:

- ☐ технологичность, т.е. понимание ограниченности существующих возможностей и способность принять новейшие технологии;
- ☐ культура, т.е. потенциальная готовность к внедрению новых процессов и взаимоотношений между разработчиками и пользователями;
- ☐ управление, т.е. четкое руководство и организованность по отношению к наиболее важным этапам и процессам внедрения.

Если организация не обладает хотя бы одним из перечисленных качеств, то внедрение CASE-средств может закончиться неудачей независимо от степени тщательности следования различным рекомендациям по внедрению.

Для того чтобы принять взвешенное решение относительно инвестиций в CASE-технологии, пользователи вынуждены производить оценку отдельных CASE-средств, опираясь на неполные и противоречивые данные.

Пользователи CASE-средств должны быть готовы к необходимости долгосрочных затрат на эксплуатацию, частому появлению новых версий и возможному быстрому моральному старению средств, а также постоянным затратам на обучение и повышение квалификации персонала.

Несмотря на все высказанные предостережения и некоторый пессимизм, грамотный и разумный подход к использованию CASE-средств может преодолеть все перечисленные трудности. Успешное внедрение CASE-средств должно обеспечить такие выгоды как:

- ☐ высокий уровень технологической поддержки процессов разработки и сопровождения ПО;
- ☐ положительное воздействие на некоторые или все из перечисленных факторов: производительность, качество продукции, соблюдение стандартов, документирование;
- ☐ приемлемый уровень отдачи от инвестиций в CASE-средства [10].

44.Жизненный цикл ПО ИС

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО).

ЖЦ ПО – это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Модель жизненного цикла ПО – это структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла. Модель жизненного цикла зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система создается и функционирует.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207: 1995 (русский аналог — ГОСТ Р ИСО/МЭК 12207-99). ISO (International Organization of Standardization) – это Международная организация по стандартизации, а IEC (International Electrotechnical Commission) – Международная комиссия по электротехнике.

По стандарту модель ЖЦ ПО включает в себя:

- ☐ стадии;
- ☐ результаты выполнения работ на каждой стадии;
- ☐ ключевые события – точки завершения работ и принятия решений.

Стадия – это часть процесса создания ПО, ограниченная определенными временными рамками и заканчивающаяся выпуском конкретного продукта (моделей, программных компонентов, документации), определяемого заданными для данной стадии требованиями.

На каждой стадии могут выполняться несколько процессов, определенных в стандарте, и наоборот, один и тот же процесс может

выполняться на различных стадиях. Соотношение между процессами и стадиями также определяется используемой моделью жизненного цикла ПО.

Каждый процесс разделен на набор действий, каждое действие – на набор задач. Каждый процесс, действие или задача инициируется и выполняется другим процессом по мере необходимости, причем не существует заранее определенных последовательностей выполнения. Связи по входным данным при этом сохраняются.

Структура ЖЦ ПО базируется на трех группах процессов:

- ☐ основные (приобретение, поставка, разработка, эксплуатация, сопровождение);
- ☐ вспомогательные (документирование, управление конфигурацией, обеспечение качества, верификация, совместная оценка, разрешение проблем);
- ☐ организационные (управление, создание инфраструктуры, совершенствование, обучение).

Каждый процесс включает ряд действий. Например, процесс приобретения охватывает следующие действия: инициирование приобретения, подготовка заявочных предложений, подготовка и корректировка договора, надзор за деятельностью поставщика, приемка и завершение работ.

Каждое действие включает ряд задач. Например, подготовка заявочных предложений должна предусматривать: формирование требований к системе, формирование списка программных продуктов, установление условий и соглашений, описание технических ограничений (среда функционирования системы и т. д.).

Рассмотрим некоторые процессы подробнее.

Разработка включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала (руководства пользователя) и т.д. Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование БД и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе

локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Сопровождение включает действия и задачи, выполняемые сопровождающей организацией, то есть службой сопровождения. Под сопровождением понимают внесение изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования ПО. Верификация – это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям. В процессе реализации проекта важное место занимают вопросы идентификации, описания и контроля конфигурации отдельных компонентов и всей системы в целом.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ.

Аттестация – это определение полноты соответствия заданных требований и созданной системы их конкретному функциональному назначению.

Аудит позволяет определить соответствие проекта требованиям, планам и условиям договора.

Разрешение проблем – это анализ и решение проблем, независимо от их происхождения или источника, которые обнаружены в ходе разработки, эксплуатации, сопровождения или других процессов.

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы. ЖЦ ПО

имеет итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах [10].

В РБ до настоящего времени при разработке ИС принято использовать советский стандарт [ГОСТ](#) 34.601-90, который предусматривает следующие стадии создания автоматизированной системы (АС): формирование требований, разработка концепции АС, разработка и утверждение технического задания на создание АС, эскизный проект, технический проект, рабочая документация, ввод в действие и сопровождение АС. Все стадии разбиты на этапы, которые подробно описываются в тексте стандарта. Однако данный стандарт не вполне подходит для проведения разработок в настоящее время: многие процессы отражены недостаточно, а некоторые положения устарели.

45. Модели жизненного цикла ПО

Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

Существует три основных модели ЖЦ ПО: каскадная модель (70-85 гг.), каскадная модель с промежуточным контролем (итерационная), спиральная модель (86-90 гг.) [10].

Суть каскадного метода (рис. 9) заключается в разбиении всей разработки на этапы, причем переход от предыдущего этапа к последующему осуществляется только после полного завершения работ предыдущего этапа, а переходов назад либо вперед или перекрытия фаз – не происходит. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.



Рис. 9. □ Каскадная модель ЖЦ ПО

Положительные стороны применения каскадного подхода заключаются в следующем:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи.

Однако у каскадной модели есть один существенный недостаток – очень сложно уложить реальный процесс создания ПО в такую жесткую схему и поэтому постоянно возникает необходимость возврата к предыдущим этапам с целью уточнения и пересмотра ранее принятых решений. Результатом такого конфликта стало появление модели с промежуточным контролем (рис. 10), которую представляют или как самостоятельную модель, или как вариант

каскадной модели. Эта модель характеризуется межэтапными корректировками, удлиняющими период разработки изделия, но повышающими надежность.

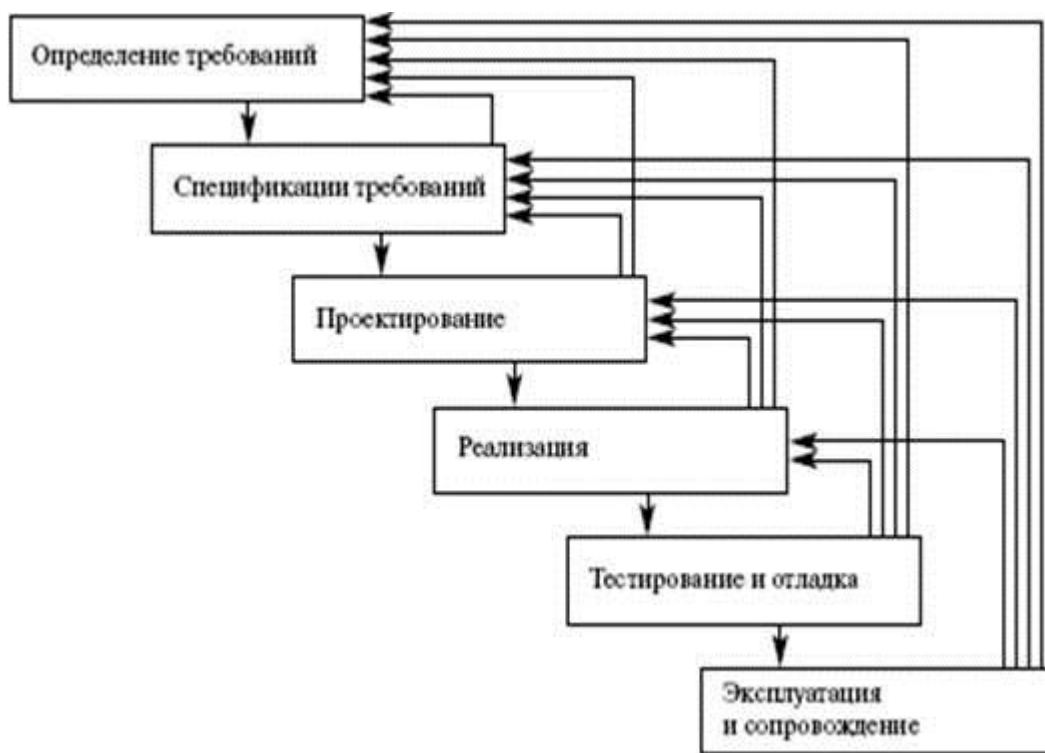


Рис. 10. □ Каскадная модель ЖЦ с промежуточным контролем

Однако и каскадная модель, и модель с промежуточным контролем обладают общим серьезным недостатком – запаздыванием с получением результатов. Это объясняется тем, что согласование результатов возможно только после завершения каждого этапа работ. На время же проведения каждого этапа требования жестко задаются в виде технического задания. Так что существует опасность, что из-за неточного изложения требований или их изменения за длительное время создания ПО конечный продукт окажется невостребованным.

Для преодоления этого недостатка и была создана спиральная модель, ориентированная на активную работу с пользователями и представляющая разрабатываемую ИС как постоянно корректируемую во время разработки. В спиральной модели (рис. 11) основной упор делается на этапы анализа и проектирования, на которых реализуемость технических решений проверяется путем создания прототипов. Спиральная модель позволяет начинать работу над следующим этапом, не дожидаясь завершения предыдущего. Спиральная

модель имеет целью, как можно раньше ознакомить пользователей с работоспособным продуктом, корректируя при необходимости требования к разрабатываемому продукту и каждый «виток» спирали означает создание фрагмента или версии. Основная проблема спирального цикла – определение момента перехода на следующий этап, и возможным ее решением является принудительное ограничение по времени для каждого из этапа ЖЦ. Наиболее полно достоинства такой модели проявляются при обслуживании программных средств.

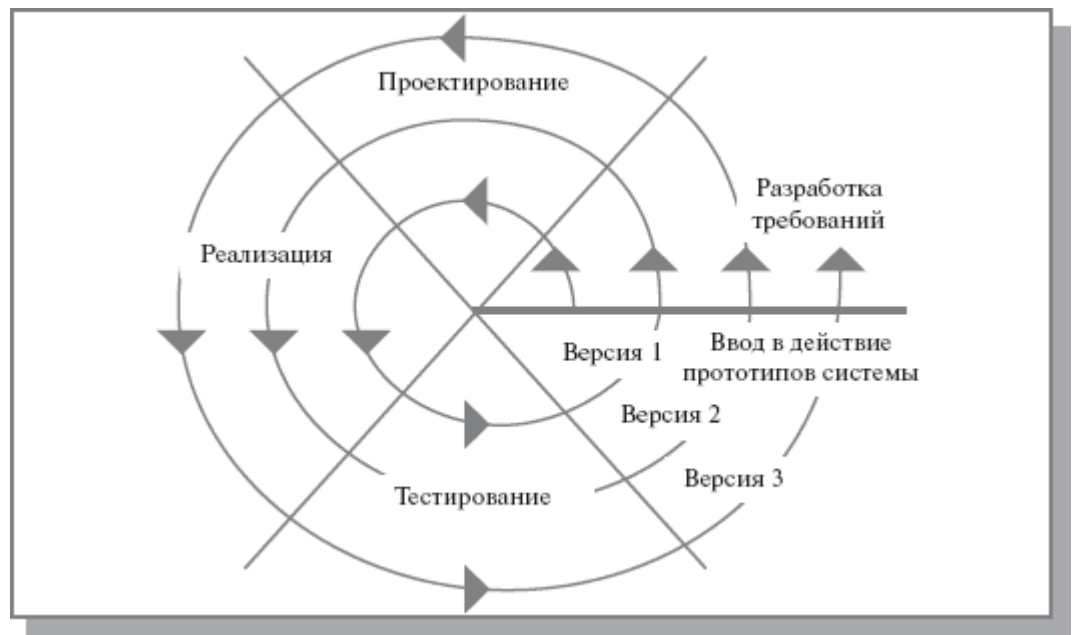


Рис. 11. □ Спиральная модель ЖЦ ПО

Основные недостатки спиральной модели:

- наличие неработоспособных версий;
- не применяется при разработке систем, требующих высокой надежности, т.е. в системах управления.

46.Методология RAD

Одним из возможных подходов к разработке ПО в рамках спиральной модели является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development) [10].

Под термином RAD обычно понимается процесс разработки ПО, содержащий 3 основных элемента:

- ☐ небольшую команду программистов (от 2 до 10 человек);
- ☐ короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);
- ☐ повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять из себя группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании ПО с использованием CASE-средств. Члены коллектива должны также уметь трансформировать в рабочие прототипы предложения конечных пользователей.

Жизненный цикл ПО по методологии RAD состоит из четырех фаз:

1. фаза анализа и планирования требований;

2. фаза проектирования;
3. фаза построения;
4. фаза внедрения.

На фазе анализа и планирования требований пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Определение требований выполняется в основном силами пользователей под руководством специалистов-разработчиков, или так называемых, аналитиков. Ограничивается масштаб проекта, определяются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации данного проекта в установленных рамках финансирования, на данных аппаратных средствах и т.п. Результатом данной фазы должны быть список и приоритетность функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этой же фазе происходит определение набора необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время – порядка 60-90 дней. С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

- ☐ общая информационная модель системы;
- ☐ функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;

- ☐ точно определенные с помощью CASE-средства интерфейсы между автономно разрабатываемыми подсистемами;
- ☐ построенные прототипы экранов, отчетов, диалогов.

Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой опасности.

В отличие от традиционного подхода, при котором использовались специфические средства прототипирования, не предназначенные для построения реальных приложений, а прототипы выбрасывались после того, как выполняли задачу устранения неясностей в проекте, в подходе RAD каждый прототип развивается в часть будущей системы. Таким образом, на следующую фазу передается более полная и полезная информация.

На фазе построения выполняется непосредственно сама быстрая разработка приложения. Разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средства. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется непосредственно в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а затем тестирование системы в целом. Завершается физическое проектирование системы:

- ☐ определяется необходимость распределения данных;
- ☐ производится анализ использования данных;
- ☐ производится физическое проектирование БД;
- ☐ определяются требования к аппаратным ресурсам;
- ☐ определяются способы увеличения производительности;

□ завершается разработка документации проекта.

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям.

На фазе внедрения производится обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется работа с существующей системой (до полного внедрения новой). Так как фаза построения достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы.

Приведенная схема разработки ИС не является абсолютной. Возможны различные варианты, зависящие, например, от начальных условий, в которых ведется разработка: разрабатывается совершенно новая система; уже было проведено обследование предприятия и существует модель его деятельности; на предприятии уже существует некоторая ИС, которая может быть использована в качестве начального прототипа или должна быть интегрирована с разрабатываемой.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика. Если же разрабатывается типовая система, которая не является законченным продуктом, а представляет собой комплекс типовых компонент, централизованно сопровождаемых, адаптируемых к программно-техническим платформам, СУБД, средствам телекоммуникации, организационно-экономическим особенностям объектов внедрения и интегрируемых с существующими разработками, на первый план выступают такие показатели проекта, как управляемость и качество, которые могут войти в противоречие с простотой и скоростью разработки. Для таких проектов необходимы высокий уровень планирования и жесткая дисциплина проектирования, строгое следование заранее разработанным протоколам и интерфейсам, что снижает скорость разработки.

Методология RAD неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т.е. программ, требующих написания большого объема (сотни тысяч строк) уникального кода.

Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени) и приложения, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход

предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

В качестве итога перечислим основные принципы методологии RAD:

- ☐ разработка приложений итерациями;
- ☐ необязательность полного завершения работ на каждом из этапов жизненного цикла;
- ☐ обязательное вовлечение пользователей в процесс разработки ИС;
- ☐ необходимое применение CASE-средств, обеспечивающих целостность проекта;
- ☐ применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;
- ☐ необходимое использование генераторов кода;
- ☐ использование прототипирования, позволяющее полнее выяснить и удовлетворить потребности конечного пользователя;
- ☐ тестирование и развитие проекта, осуществляемые одновременно с разработкой;
- ☐ ведение разработки немногочисленной хорошо управляемой командой профессионалов;
- ☐ грамотное руководство разработкой системы, четкое планирование и контроль выполнения работ.

47. Сущность структурного подхода к проектированию

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции (разбиении) на автоматизируемые функции: система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны. При разработке системы «снизу-вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов [10].

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов:

- принцип «разделяй и властвуй», который подразумевает решение сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;
- принцип иерархического упорядочивания, который обеспечивает организацию составных частей системы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне;
- принцип абстрагирования, который заключается в выделении существенных аспектов системы и отвлечения от несущественных;
- принцип формализации, который заключается в необходимости строгого методического подхода к решению проблемы;
- принцип непротиворечивости, который заключается в обоснованности и согласованности элементов;
- принцип структурирования данных, который заключается в том, что данные должны быть структурированы и иерархически организованы.

Первые два принципа считают основными.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными являются следующие:

1. SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы;
2. DFD (Data Flow Diagrams) – диаграммы потоков данных;
3. ERD (Entity-Relationship Diagrams) – диаграммы «сущность-связь».

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

48.Методология функционального проектирования SADT, состав модели

Методология SADT разработана Дугласом Россом. На ее основе разработана, в частности, известная методология IDEF0 (Icam DEFinition), которая является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой по инициативе ВВС США [6, 10].

Методология IDEF0, более известная как методология SADT, представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этой методологии основываются на следующих концепциях:

- ☐ графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описываются посредством интерфейсных дуг, выражающих «ограничения», которые в свою очередь определяют, когда и каким образом функции выполняются и управляются;

- ☐ строгость и точность. Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика.

Правила SADT включают:

- ☐ ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков);
- ☐ связность диаграмм (номера блоков);
- ☐ уникальность меток и наименований (отсутствие повторяющихся имен);
- ☐ синтаксические правила для графики (блоков и дуг);
- ☐ разделение входов и управлений (правило определения роли данных);
- ☐ отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Методология SADT может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы, которая удовлетворяет этим требованиям и реализует эти функции. Для уже существующих систем SADT может быть применена с целью анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.

Результатом применения методологии SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы – главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показана с левой стороны блока, а результаты выхода показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу. Функциональный блок и интерфейсные дуги в общем виде представлены на рис. 12, пример использования – на рис. 13.



Рис. 12 □ Функциональный блок и интерфейсные дуги



Рис. 13 □ Пример использования

Одной из наиболее важных особенностей методологии SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.

Построение SADT-модели начинается с представления всей системы в виде простейшей компоненты – одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также представляют полный набор внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления.

Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию. Кроме того, модель не может опустить какие-либо элементы, т.е., как уже отмечалось, родительский блок и его интерфейсы обеспечивают контекст. К нему нельзя ничего добавить, и из него не может быть ничего удалено.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые представлены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется родительской для более детальной диаграммы.

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

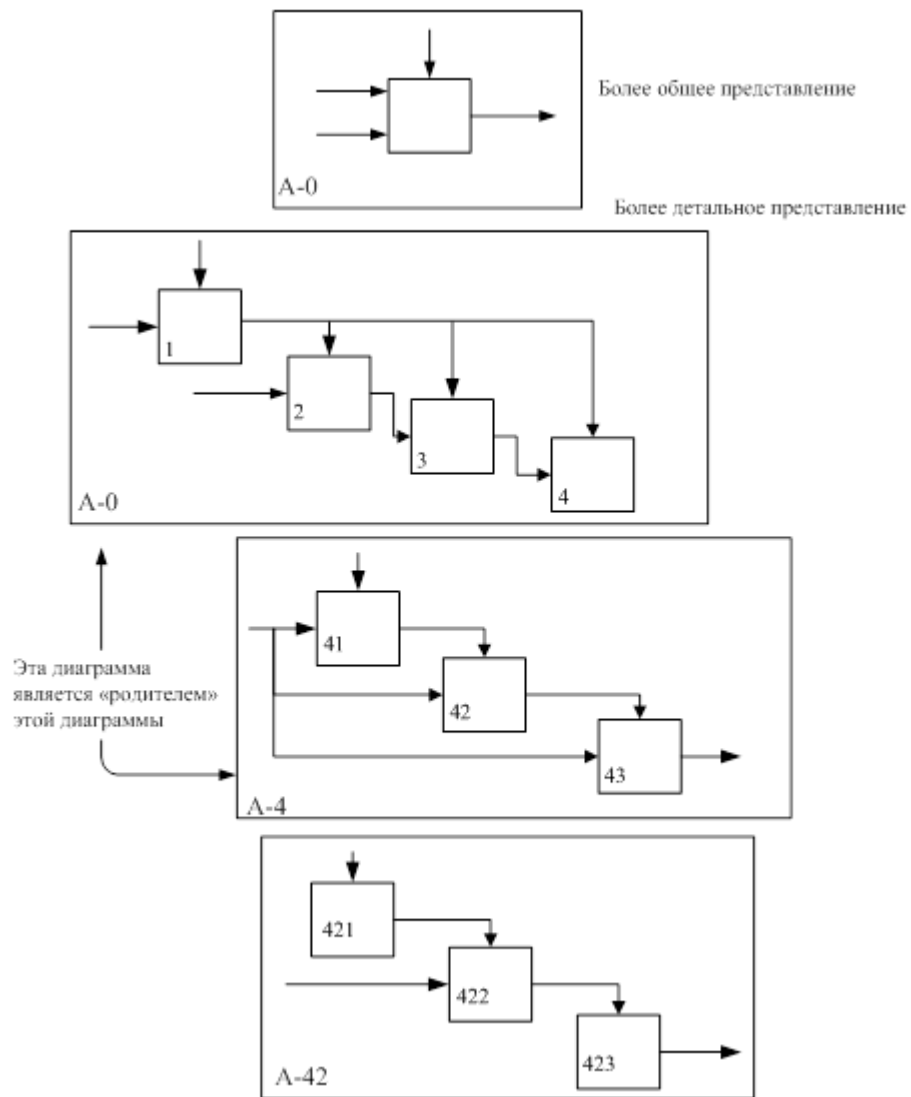


Рис. 14 □ Декомпозиция диаграмм SADT-модели

На SADT-диаграммах не указываются явно ни последовательность, ни время. Обратные связи, итерации, продолжающиеся процессы и перекрывающиеся (по времени) функции могут быть изображены с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д. (рис. 15).



Рис. 15 □ Пример обратной связи

Типы связей между функциями. Одним из важных моментов при проектировании ИС с помощью методологии SADT является точная согласованность типов связей между функциями. Различают, по крайней мере, семь типов связывания:

Таблица 41. – Типы связей

Тип связи	Относительная значимость
Случайная	0
Логическая	1
Временная	2
Процедурная	3
Коммуникационная	4
Последовательная	5
Функциональная	6

Тип случайной связности (0): наименее желательный. Случайная связность возникает, когда конкретная связь между функциями мала или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют малую связь друг с другом. Крайний вариант этого случая показан на рис. 16.

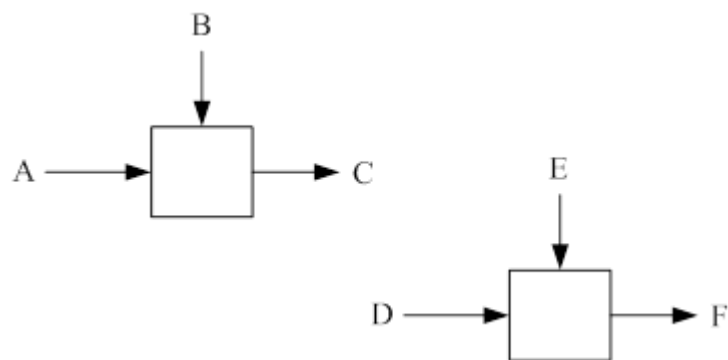


Рис. 16 □ Случайная связность

Тип логической связности (1). Логическое связывание происходит тогда, когда данные и функции собираются вместе вследствие того, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

Тип временной связности (2). Связанные по времени элементы возникают вследствие того, что они представляют функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

Тип процедурной связности (3). Процедурно-связанные элементы появляются сгруппированными вместе вследствие того, что они выполняются в течение одной и той же части цикла или процесса. Пример процедурно-связанной диаграммы приведен на рис.17.

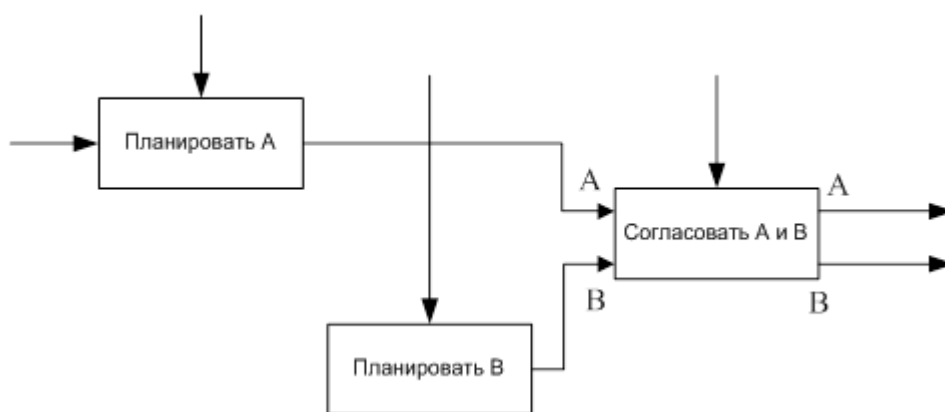


Рис. 17 □ Процедурная связность

Тип коммуникационной связности (4). Диаграммы демонстрируют коммуникационные связи, когда блоки группируются вследствие того, что они используют одни и те же входные данные и/или производят одни и те же выходные данные (рис. 18).

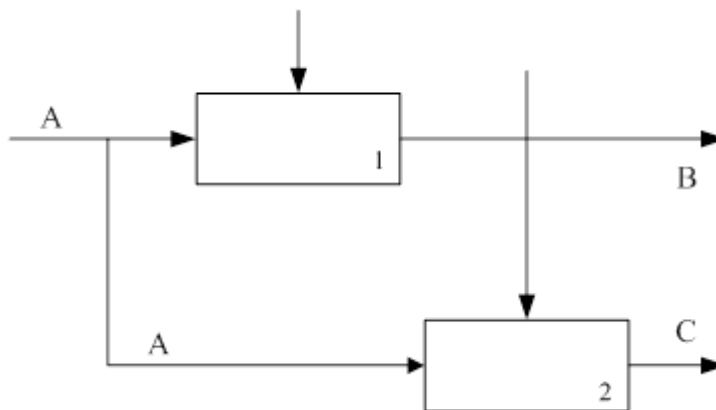


Рис. 18 □ Коммуникационная связность

Тип последовательной связности (5). На диаграммах, имеющих последовательные связи, выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем на рассмотренных выше уровнях связей, поскольку моделируются причинно-следственные зависимости (рис. 19).

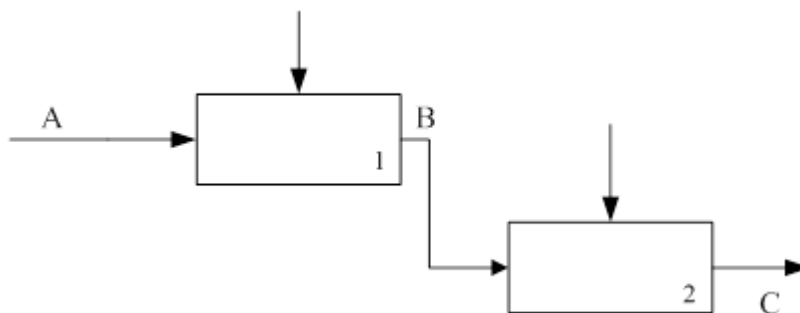


Рис. 19 □ Последовательная связность

Тип функциональной связности (6). Диаграмма отражает полную функциональную связность, при наличии полной зависимости одной функции от другой. Диаграмма, которая является чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связности. Одним из способов определения функционально-связанных диаграмм является рассмотрение двух блоков, связанных через управляющие дуги, как показано на рис. 20.

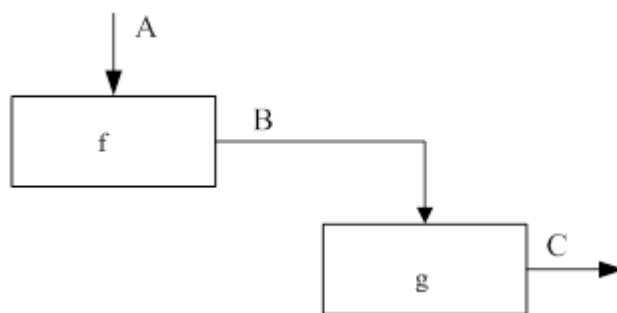


Рис. 20 □ Функциональная связность

Важно отметить, что уровни 4-6 устанавливают типы связностей, которые разработчики считают важнейшими для получения диаграмм хорошего качества.

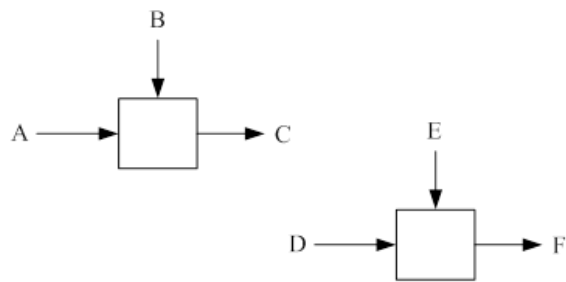
49. Типы связей между функциями в методологии SADT

Методология IDEF0, более известная как методология SADT, представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Одним из важных моментов при проектировании ИС с помощью методологии SADT является точная согласованность типов связей между функциями. Различают, по крайней мере, семь типов связывания:

Тип связи	Относительная значимость
Случайная	0
Логическая	1
Временная	2
Процедурная	3
Коммуникационная	4
Последовательная	5
Функциональная	6

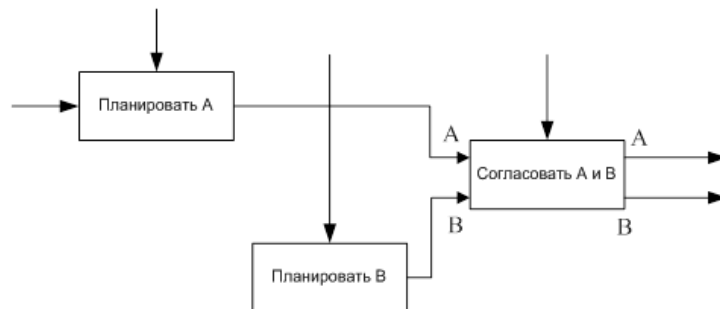
Тип случайной связности (0): наименее желательный. Случайная связность возникает, когда конкретная связь между функциями мала или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют малую связь друг с другом



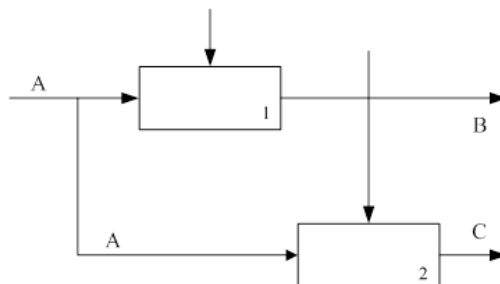
Тип логической связности (1). Логическое связывание происходит тогда, когда данные и функции собираются вместе вследствие того, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

Тип временной связности (2). Связанные по времени элементы возникают вследствие того, что они представляют функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

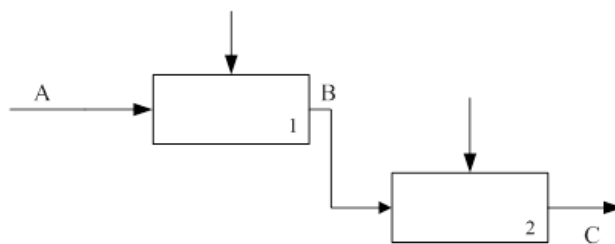
Тип процедурной связности (3). Процедурно-связанные элементы появляются сгруппированными вместе вследствие того, что они выполняются в течение одной и той же части цикла или процесса.



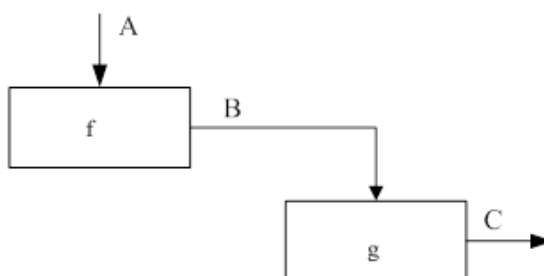
Тип коммуникационной связности (4). Диаграммы демонстрируют коммуникационные связи, когда блоки группируются вследствие того, что они используют одни и те же входные данные и/или производят одни и те же выходные данные



Тип последовательной связности (5). На диаграммах, имеющих последовательные связи, выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем на рассмотренных выше уровнях связей, поскольку моделируются причинно-следственные зависимости.



Тип функциональной связности (6). Диаграмма отражает полную функциональную связность, при наличии полной зависимости одной функции от другой. Диаграмма, которая является чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связности.



Важно отметить, что уровни 4-6 устанавливают типы связностей, которые разработчики считают важными для получения диаграмм хорошего качества.

50. Моделирование потоков данных (методология Гейна-Сарсона) основные компоненты.

Диаграммы потоков данных являются основным средством моделирования функциональных требований проектируемой системы. Эти требования разбиваются на функциональные компоненты (процессы) и представляются в виде сети, связанной потоками данных. Главная цель таких средств – продемонстрировать, как каждый процесс преобразует потоки входных данных в выходные, а также выявить отношения между этими процессами.

Для изображения DFD можно использовать нотации Гейна-Сарсона, Йордана – Де Марко и другие

Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации. Основными компонентами являются:

- внешние сущности;
- системы/подсистемы;
- процессы;
- накопители данных;
- потоки данных.

Внешняя сущность - это материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, заказчики, персонал, поставщики, клиенты, склад. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой ИС. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой ИС, если это необходимо, или, наоборот, часть процессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

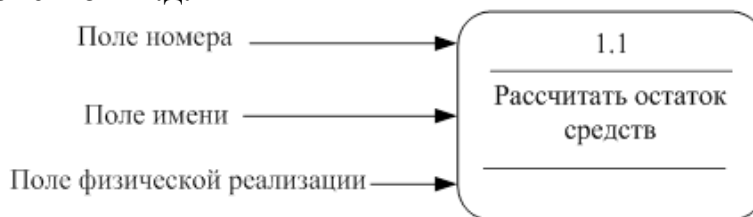
Внешняя сущность обозначается квадратом и располагается как бы «над» диаграммой.

При построении модели сложной ИС внешняя сущность может быть представлена в самом общем виде на так называемой контекстной диаграмме либо может быть декомпозирована на ряд *подсистем*.



Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно реализованное логическое устройство и т.д.



Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

D1	Получаемые счета
----	------------------

Накопитель данных идентифицируется буквой «D» и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика. Накопитель данных в общем случае является прообразом таблицы будущей БД и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой,

Атрибутами потока данных могут быть:

- имена и синонимы потоков данных в соответствии с узлами изменения имени;
- единицы измерения потоков;

- диапазон значений для непрерывного потока;
- список значений и их смысл для дискретного потока;
- список номеров диаграмм, в которых встречается поток;
- комментарии.

DFD предназначены для:

- ☐ определения основных функций, потоков и хранилищ данных;
- ☐ отображения потоков данных и функционального состава системы;
- ☐ определения связей между функциями;
- ☐ формирования системных требований.

Главная цель построения иерархического множества DFD заключается в том, чтобы сделать требования ясными и понятными на каждом уровне детализации, а также разбить эти требования на части с точно определенными отношениями между ними.

классические ошибки, возникающие при построении DFD:

1. «черная дыра» ☐ процесс имеет только входные потоки данных;
2. «серая дыра» ☐ генерирует выходной поток при недостаточных входных данных;
3. «белая дыра» ☐ наличие только выходных потоков;
4. прямая связь двух внешних сущностей или хранилищ
5. преобразование внешней сущности в хранилище данных наоборот.

51. Построение иерархии диаграмм потоков данных

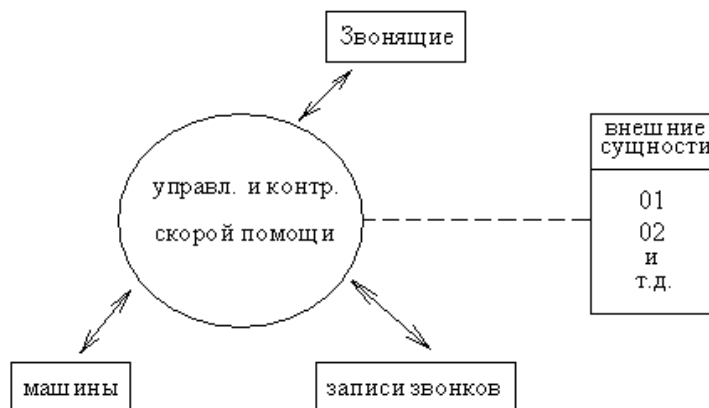
В соответствии с методологией Гейна-Сарсона модель проектируемой или реально существующей системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становятся элементарными и детализировать их далее невозможно.

Построение диаграммы потоков данных можно условно разбить на несколько этапов:

1. построение контекстной диаграммы;
2. декомпозиция (детализация) процессов;
3. декомпозиция данных.

Первым шагом при построении иерархии DFD является построение контекстной диаграммы. Контекстная диаграмма – это диаграмма верхнего уровня модели, которая отражает интерфейс системы с внешним миром.

Обычно при проектировании относительно простых ИС строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы



Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы.

Признаками сложности (в смысле контекста) могут быть:

- ☐ наличие большого количества внешних сущностей (десять и более);
- ☐ распределенная природа системы;

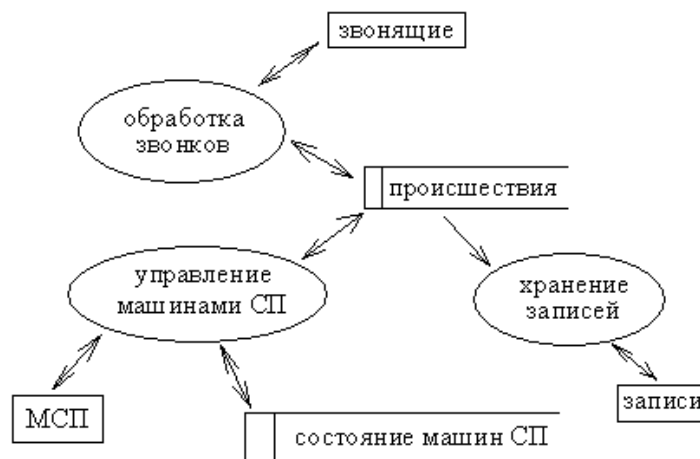
□ многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных ИС строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

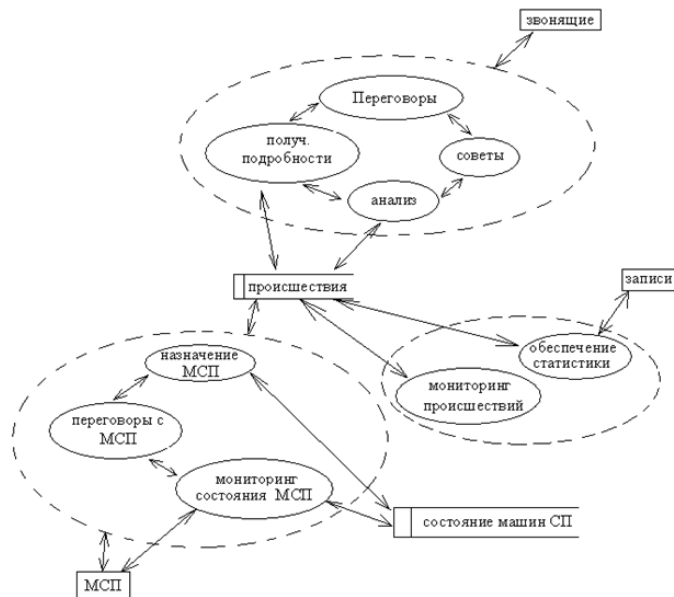
Иерархия диаграмм определяет взаимодействие основных функциональных подсистем проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС.

После построения контекстной диаграммы полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстной диаграмме, выполняется ее детализация при помощи DFD уровня подсистемы



Каждый процесс на DFD подсистемы, в свою очередь, может быть детализирован при помощи DFD еще более низкого уровня или спецификации.



При детализации должны выполняться следующие правила:

1. правило балансировки, которое означает, что при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников/приемников данных может иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеет информационную связь детализируемая подсистема или процесс на родительской диаграмме;
2. правило нумерации, которое означает, что при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т.д.

52. Понятие и методы задания спецификаций процессов

Спецификация (описание логики процесса) должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

Спецификация является конечной точкой иерархии DFD. Решение о завершении детализации процесса и использовании спецификации принимается аналитиком исходя из следующих критериев:

- ☐ наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- ☐ возможности описания преобразования данных процессом в виде последовательного алгоритма;
- ☐ выполнения процессом единственной логической функции преобразования входной информации в выходную;
- ☐ возможности описания логики процесса при помощи миниспецификации небольшого объема (не более 20-30 строк).

Спецификации содержат номер и/или имя процесса, списки входных и выходных данных и тело процесса.

Методы задания спецификации процесса:

- ☐ текстовое описание;
- ☐ структурированный естественный язык – является комбинацией строгости языка программирования и читабельности естественного языка и состоит из подмножества слов, организованных в определенные логические структуры, арифметических выражений и диаграмм. Применяется, если детали спецификации известны не полностью, обеспечивает быстрое проектирование, прост в использовании, легок для понимания, но не приспособлен к автоматической кодогенерации из-за наличия неоднозначностей.
- ☐ таблицы и деревья решений позволяют управлять сложными комбинациями условий и действий, обеспечивают визуальное представление и легко понимаются конечным пользователем, но не обладают процедурными возможностями.
- ☐ визуальные языки – базируются на основных идеях структурного программирования и позволяют определять потоки управления с помощью специальных иерархически организованных схем. Визуальные языки поддерживают автоматическую кодогенерацию, но вызывают затруднения при модификации спецификации в случае изменения каких-либо деталей.
- ☐ языки программирования.

53. Декомпозиция данных, расширения реального времени и типы управляющих потоков, словарь данных

54. Моделирование потоков данных, основные ошибки моделирования.

Диаграммы потоков данных (DFD □ Data Flow Diagrams) являются основным средством моделирования функциональных требований проектируемой системы. Эти требования разбиваются на функциональные компоненты (процессы) и представляются в виде сети, связанной потоками данных. Главная цель таких средств – продемонстрировать, как каждый процесс преобразует потоки входных данных в выходные, а также выявить отношения между этими процессами.

Для изображения DFD можно использовать нотации Гейна-Сарсона, Йордана – Де Марко и другие [6, 10].

В соответствии с методологией Гейна-Сарсона модель проектируемой или реально существующей системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становятся элементарными и детализировать их далее невозможно.

Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации. Таким образом, основными компонентами диаграмм потоков данных Гейна-Сарсона являются:

- внешние сущности;
- системы/подсистемы;
- процессы;
- накопители данных;
- потоки данных.

Внешняя сущность □ это материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, заказчики, персонал, поставщики, клиенты, склад. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой ИС. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой ИС, если это необходимо, или, наоборот, часть процессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

Внешняя сущность обозначается квадратом (рис. 21) и располагается как бы «над» диаграммой.



Рис. 21. – Внешняя сущность

При построении модели сложной ИС внешняя сущность может быть представлена в самом общем виде на так называемой контекстной диаграмме либо может быть декомпозирована на ряд подсистем.

Подсистема (или система) на контекстной диаграмме изображается следующим образом (рис. 22):

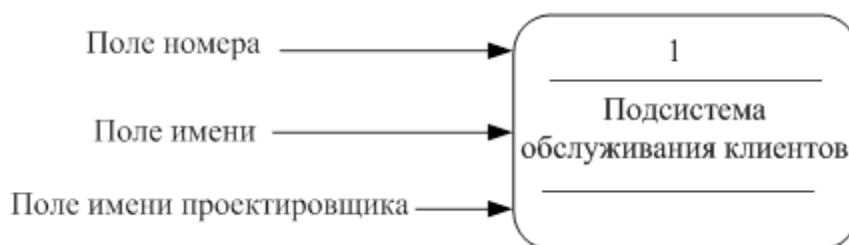


Рис. 22. – Подсистема

Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации (отдел), выполняющее обработку входных документов и выпуск отчетов, программа, аппаратно реализованное логическое устройство и т.д.

Процесс на диаграмме потоков данных изображается, как показано на рис. 23.

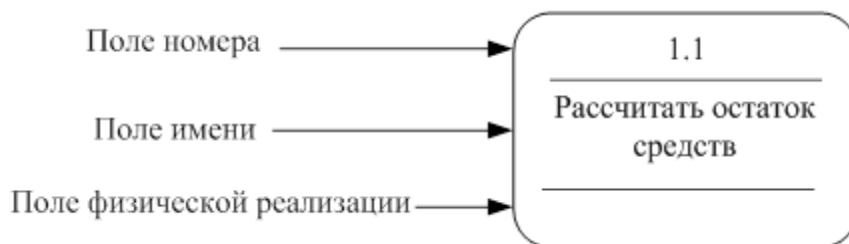


Рис. 23 – Процесс

Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например: «Ввести сведения о клиентах»; «Выдать информацию о текущих расходах»; «Проверить кредитоспособность клиента».

Использование таких глаголов, как «обработать», «модернизировать» или «отредактировать» означает, как правило, недостаточно глубокое понимание данного процесса и требует дальнейшего анализа.

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных изображается, как показано на рис. 24.

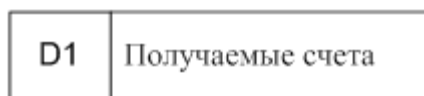


Рис. 24. – Накопитель данных

Накопитель данных идентифицируется буквой «D» и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика.

Накопитель данных в общем случае является прообразом таблицы будущей БД и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рис. 25).



Рис. 25. – Поток данных

Атрибутами потока данных могут быть:

- ☐ имена и синонимы потоков данных в соответствии с узлами изменения имени;
- ☐ единицы измерения потоков;
- ☐ диапазон значений для непрерывного потока;
- ☐ список значений и их смысл для дискретного потока;
- ☐ список номеров диаграмм, в которых встречается поток;
- ☐ комментарии.

Построение диаграммы потоков данных можно условно разбить на несколько этапов [11]:

1. построение контекстной диаграммы;
2. декомпозиция (детализация) процессов;
3. декомпозиция данных.

Первым шагом при построении иерархии DFD является построение контекстной диаграммы. Контекстная диаграмма – это диаграмма верхнего уровня модели, которая отражает интерфейс системы с внешним миром.

Обычно при проектировании относительно простых ИС строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы, например, как на рис. 26:

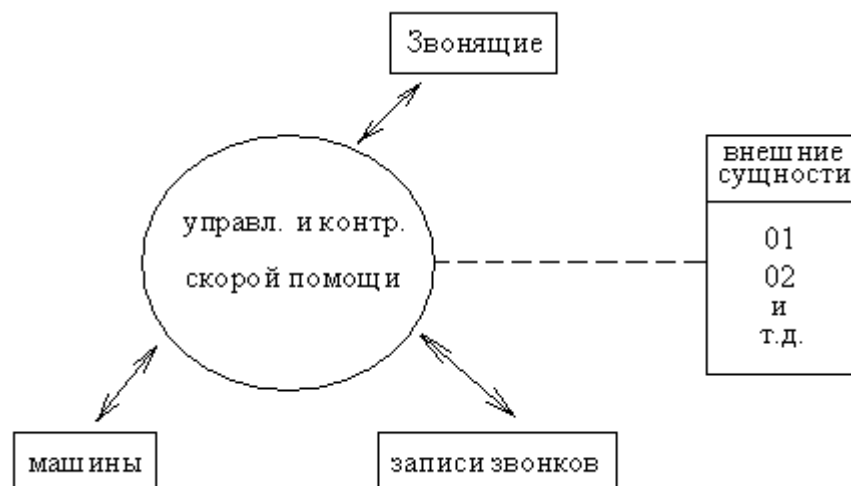


Рис. 26. – Пример контекстной диаграммы

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы.

Признаками сложности (в смысле контекста) могут быть:

- ☐ наличие большого количества внешних сущностей (десять и более);
- ☐ распределенная природа системы;
- ☐ многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных ИС строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

Иерархия диаграмм определяет взаимодействие основных функциональных подсистем проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС.

Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры ИС на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в разработке которых участвуют разные организации и коллективы разработчиков.

После построения контекстной диаграммы полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстной диаграмме, выполняется ее детализация при помощи DFD уровня подсистемы (рис. 27).

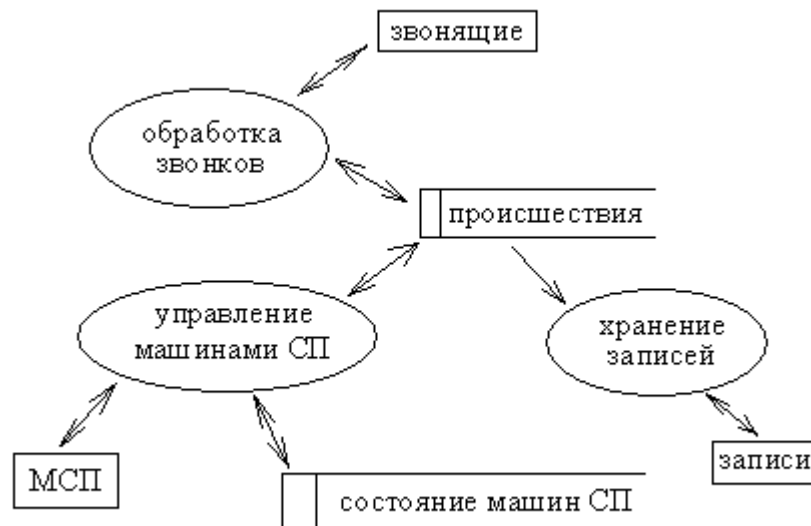


Рис. 27. – Пример детализации основного процесса

Каждый процесс на DFD подсистемы, в свою очередь, может быть детализирован при помощи DFD еще более низкого уровня (рис. 28) или спецификации.

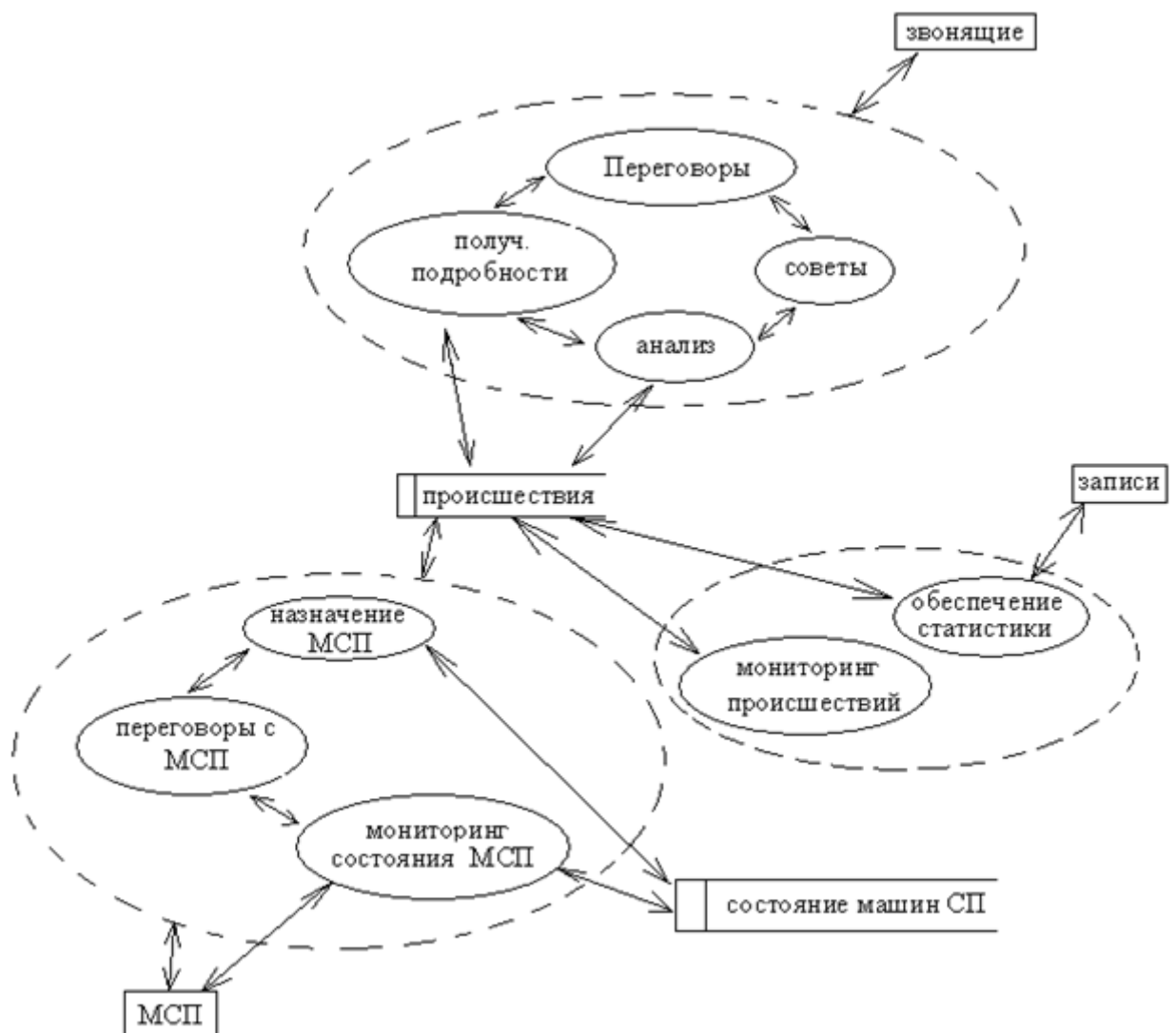


Рис. 28. – Пример дальнейшей детализации основного процесса

При детализации должны выполняться следующие правила:

1. правило балансировки, которое означает, что при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников/приемников данных может иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеет информационную связь детализируемая подсистема или процесс на родительской диаграмме;

2. правило нумерации, которое означает, что при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т.д.

Спецификация (описание логики процесса) должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу.

Спецификация является конечной точкой иерархии DFD. Решение о завершении детализации процесса и использовании спецификации принимается аналитиком исходя из следующих критериев:

- ☐ наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);
- ☐ возможности описания преобразования данных процессом в виде последовательного алгоритма;
- ☐ выполнения процессом единственной логической функции преобразования входной информации в выходную;
- ☐ возможности описания логики процесса при помощи миниспецификации небольшого объема (не более 20-30 строк).

Итак, спецификация процесса используется для описания функционирования процесса в случае отсутствия необходимости детализировать его с помощью DFD.

Фактически миниспецификация представляет собой алгоритмы описания задач, выполняемых процессами. Множество всех спецификаций является полной спецификацией системы. Спецификации содержат номер и/или имя процесса, списки входных и выходных данных и тело процесса.

Методы задания спецификации процесса:

- ☐ текстовое описание;
- ☐ структурированный естественный язык – является комбинацией строгости языка программирования и читабельности естественного языка и состоит из подмножества слов, организованных в определенные логические структуры, арифметических выражений и диаграмм. Применяется, если детали спецификации известны не полностью, обеспечивает быстрое проектирование, прост в использовании, легок для понимания, но не приспособлен к автоматической кодогенерации из-за наличия неоднозначностей.

☐ таблицы и деревья решений позволяют управлять сложными комбинациями условий и действий, обеспечивают визуальное представление и легко понимаются конечным пользователем, но не обладают процедурными возможностями.

☐ визуальные языки – базируются на основных идеях структурного программирования и позволяют определять потоки управления с помощью специальных иерархически организованных схем. Визуальные языки поддерживают автоматическую кодогенерацию, но вызывают затруднения при модификации спецификации в случае изменения каких-либо деталей.

☐ языки программирования.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных текущего уровня, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений.

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные недетализированные объекты следует детализировать, вернувшись на предыдущие шаги разработки. В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть записаны.

Таким образом DFD предназначены для:

- ☐ определения основных функций, потоков и хранилищ данных;
- ☐ отображения потоков данных и функционального состава системы;
- ☐ определения связей между функциями;
- ☐ формирования системных требований.

Главная цель построения иерархического множества DFD заключается в том, чтобы сделать требования ясными и понятными на каждом уровне детализации, а также разбить эти требования на части с точно определенными отношениями между ними. Для достижения этого целесообразно использовать следующие рекомендации:

- ☐ размещать на каждой диаграмме от 3 до 7 процессов;
- ☐ не загромождать диаграммы не существенными на данном уровне деталями;

□ осуществлять параллельно декомпозицию процессов и декомпозицию данных;

□ выбирать ясные, отражающие суть дела, имена процессов и потоков.

Для обеспечения декомпозиции данных в DFD могут быть добавлены следующие типы объектов:

1. групповой узел предназначен для расщепления или объединения потоков;



Рис. 29. – Групповой узел

2. узел-предок позволяет увязать входящие и выходящие потоки между несколькими уровнями декомпозиции процессов;



Рис. 30. –Узел-предок

3. неиспользуемый узел применяется в ситуации, когда декомпозиция данных производится в групповом узле и при этом требуются не все элементы входящего потока

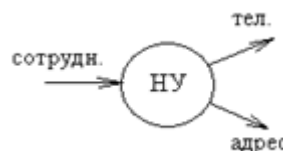


Рис. 31. – Неиспользуемый узел

4. узел изменения имени позволяет неоднозначно именовать потоки, при том, что их содержимое эквивалентно.



Рис. 32. – Узел изменения имени

5. Текст в свободном формате в любом месте диаграммы.

Также на DFD используют атрибуты расширения реального времени. Они обозначаются пунктирными линиями и отражают процесс управления во

времени. Так в качестве расширений могут использоваться управляющий поток, управляющий процесс и управляющее хранилище.

Выделяют 3 группы управляющих потоков:

- Т-поток (tigger flow), который может вызвать выполнение процесса одной короткой операцией;

- А-поток (activator flow), может изменять выполнение отдельного процесса, обеспечивает непрерывность выполнения процесса, пока включен.

- Е/D-поток (enable-disable flow), может переключать выполнение отдельного процесса, течение по линии Е вызывает выполнение процесса, которое продолжается до тех пор, пока не начинается течение по линии D.

В заключение отметим классические ошибки, возникающие при построении DFD:

1. «черная дыра» □ процесс имеет только входные потоки данных;

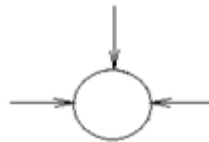


Рис. 33. – «черная дыра»

2. «серая дыра» □ генерирует выходной поток при недостаточных входных данных;

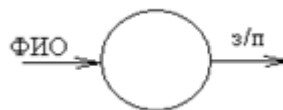


Рис. 34. – «серая дыра»

3. «белая дыра» □ наличие только выходных потоков;

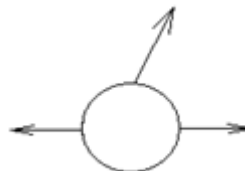


Рис. 35. – «белая дыра»

4. прямая связь двух внешних сущностей или хранилищ (рис. 36);



Рис. 36. – Связь внешних сущностей

5. преобразование внешней сущности в хранилище данных (рис. 37) и наоборот.



Рис. 37. – Преобразование внешней сущности в хранилище

55. Моделирование данных: ER – диаграммы (нотации Чена, Мартина и др)

Цель моделирования данных состоит в обеспечении разработчика ИС концептуальной схемой БД в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую СУБД.

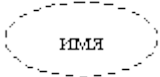
Наиболее распространенным средством моделирования данных являются диаграммы «сущность-связь». С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных БД, но также могут быть применены и для создания постреляционных БД.

Нотация ERD была впервые введена П. Ченом и получила дальнейшее развитие в работах Мартина, Баркера, Т.Рэмея [6, 10].

Нотация Чена приведена в табл. 42:

Таблица 42

Элемент диаграммы	Описание
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи
	связь
	идентифицирующая связь
	атрибут
	первичный ключ
	внешний ключ (понятие внешнего ключа вводится в реляционной модели данных)
	многозначный атрибут

	получаемый (наследуемый) атрибут в иерархических связях
---	---

Связь соединяется с ассоциируемыми сущностями линиями. Возле каждой сущности на линии, соединяющей ее со связью, цифрами указывается класс принадлежности. Пример приведен на рис. 38:

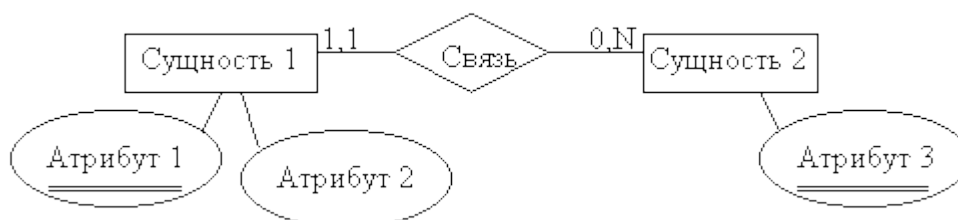
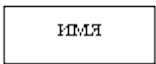
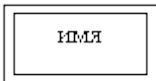
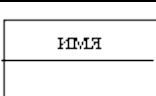


Рис. 38

Нотация Мартина приведена в табл. 43:

Таблица 43

Элемент диаграммы	Описание
	независимая сущность
	зависимая сущность
	родительская сущность в иерархической связи

Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Ключевые атрибуты подчеркиваются. Связи изображаются линиями, соединяющими сущности, вид линии в месте соединения с сущностью определяет кардинальность связи:

Таблица 44

Обозначение	Кардинальность
	нет
	1,1
	0,1
	M,N

56. Моделирование данных CASE-методом Баркера.

Наиболее распространенными методами для построения ERD являются метод Баркера и метод IDEF1. Метод Баркера основан на нотации, предложенной автором, и используется в CASE -средстве Oracle Designer.

CASE-средство Oracle Designer фирмы Oracle является интегрированным CASE-средством, обеспечивающим в совокупности со средствами разработки приложений Oracle Developer и Oracle Application Server поддержку полного ЖЦ ПО для систем, использующих СУБД Oracle.

Oracle Designer представляет собой семейство методов и поддерживающих их программных продуктов. Базовый метод Oracle Designer (CASE*Method Баркера) – структурный метод проектирования систем, охватывающий полностью все стадии ЖЦ ПО.

Основные шаги метода это: выделение сущностей, идентификация связей, идентификация атрибутов [10]. Первый шаг моделирования – это сбор информации и выделение сущностей.

Сущность (рис. 40) – реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

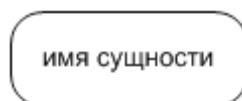


Рис. 40. – Графическое изображение сущности

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать:

- уникальным именем, и к одному и тому же имени должна всегда применяться одна и та же интерпретация;
- одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
- одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- любым количеством связей с другими сущностями модели.

Например, для компании по торговле автомобилями можно выделить четыре сущности (автомашина, продавец, покупатель, контракт), которые изображаются на диаграмме следующим образом (рис. 41).



Рис. 41. – Сущности для компании по торговле автомобилями

Следующим шагом моделирования является идентификация связей.

Связь – это поименованная ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Например, связь продавца с контрактом может быть выражена следующим образом:

□ продавец может получить вознаграждение за 1 или более контрактов;

□ контракт должен быть инициирован ровно одним продавцом.

Степень связи и ее обязательность графически изображаются следующим образом (рис. 42).



Рис. 42. – Графическое изображение связей

Таким образом, приведенные выше предложения, описывающие связь продавца с контрактом, графически будут выражены следующим образом (рис. 43).

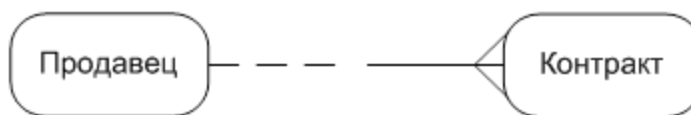


Рис. 43. – Связь между сущностями

Описав также связи остальных сущностей, получим следующую схему (рис. 44).

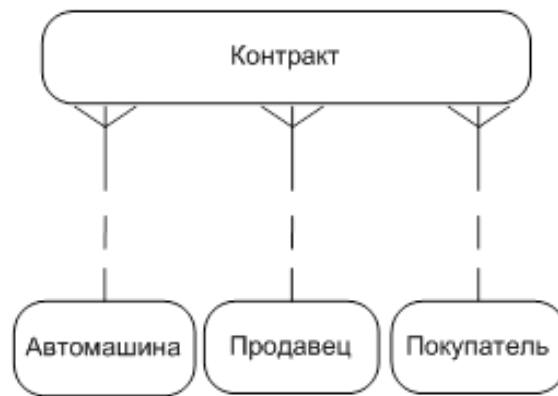


Рис. 44. – Добавление связей на ERD

Последним шагом моделирования является идентификация атрибутов.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных со множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.).

Экземпляр атрибута — это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым значением атрибута. В ER-модели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Атрибут может быть либо обязательным, либо необязательным. Обязательность означает, что атрибут не может принимать неопределенных значений (null values) и обозначается символом «звездочка», необязательный атрибут обозначается кружком (o). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор — это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя.

Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются наверху списка и выделяются знаком «#».

Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности — это один или несколько атрибутов, чьи значения

однозначно определяют каждый экземпляр сущности. При существовании нескольких возможных ключей один из них обозначается в качестве первичного ключа, а остальные □ как альтернативные ключи.

С учетом имеющейся информации дополним построенную ранее диаграмму (рис. 45).



Рис. 45. – ERD для компании по торговле автомобилями

Помимо перечисленных основных конструкций модель данных может содержать ряд дополнительных:

1. Подтипы и супертипы: одна сущность является обобщающим понятием для группы подобных сущностей (рис. 46).

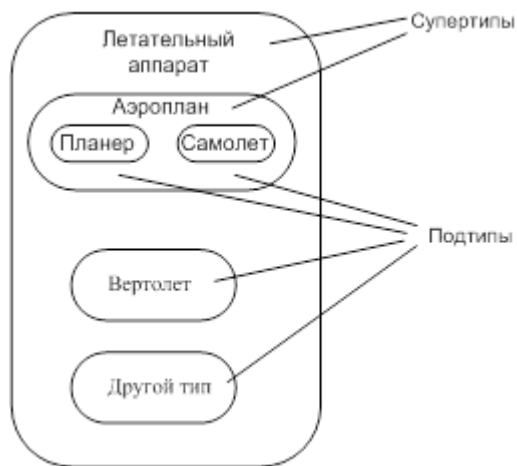


Рис. 46. – Подтипы и супертипы

2. Взаимно исключающие связи: каждый экземпляр сущности участвует только в одной связи из группы взаимно исключающих связей (рис. 47).

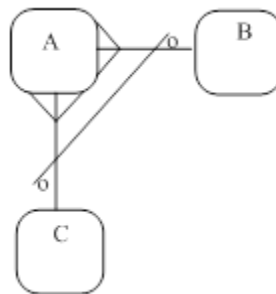


Рис. 47. – Взаимно исключающие связи

3. Рекурсивные связи: сущность может быть связана сама с собой (рис. 48).

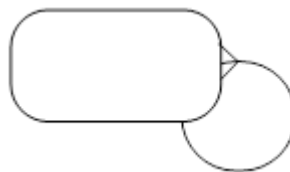


Рис. 48. – Рекурсивная связь

4. Неперемещаемые связи: экземпляр сущности не может быть перенесен из одного экземпляра связи в другой (рис. 49).



Рис. 49. – Неперемещаемая связь

57. Методология IDEF1X

Методология IDEF1, разработанная Т. Рэмей, также основана на подходе П. Чена и позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. На основе совершенствования методологии IDEF1 была создана ее новая версия – методология IDEF1X, которая разработана с учетом таких требований, как простота изучения и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF) [10].

Сущность в методологии IDEF1X является независимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (рис. 50).

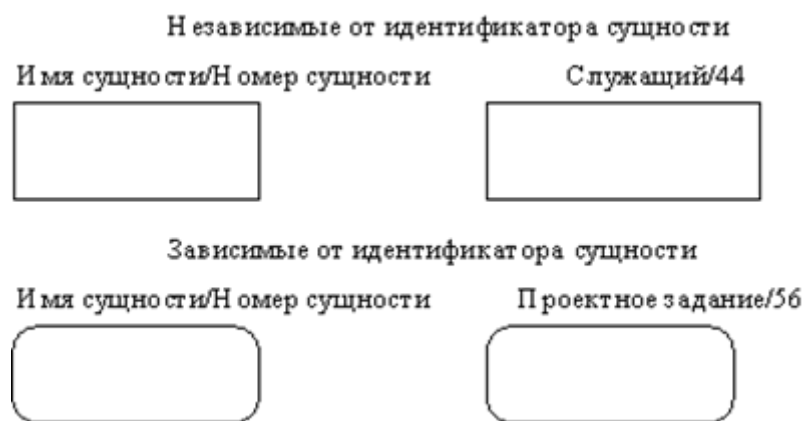


Рис. 50. – Сущности

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой (/) и помещаемые над блоком.

Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X могут быть выражены следующие мощности связей:

- ☐ каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- ☐ каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- ☐ каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- ☐ каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае – неидентифицирующей.

Связь изображается линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. Мощность связи обозначается, как показано на рис. 51.

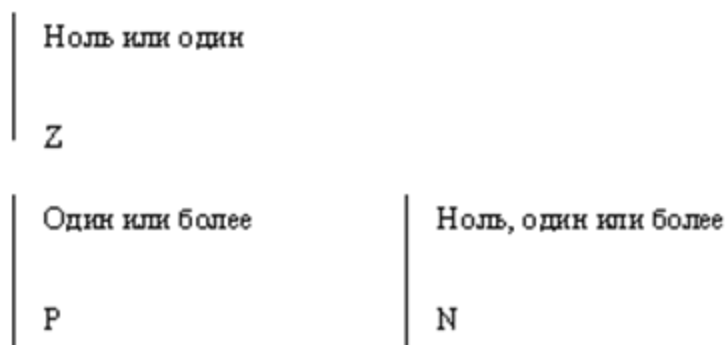


Рис. 51. – Мощность связи

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (рис. 52). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора сущностью (это определяется ее связями с другими сущностями).



Рис. 52. – Идентифицирующая связь

Пунктирная линия изображает неидентифицирующую связь (рис. 53). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

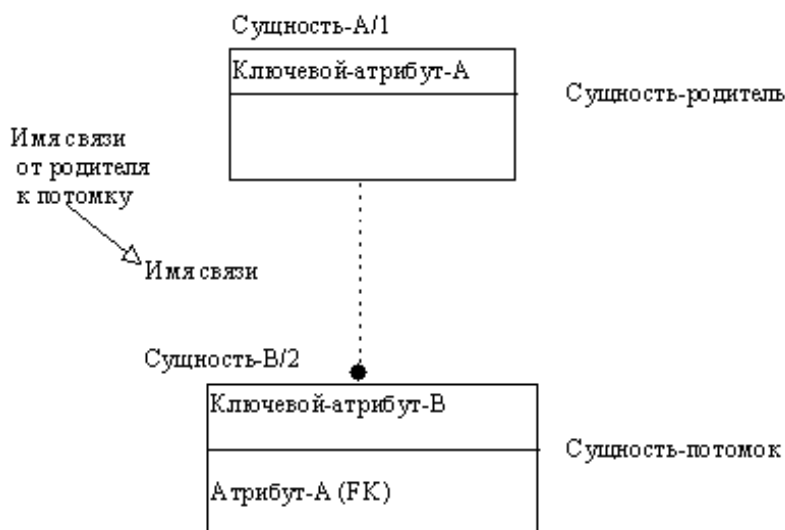


Рис. 53. – Неидентифицирующая связь

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой (рис. 54).

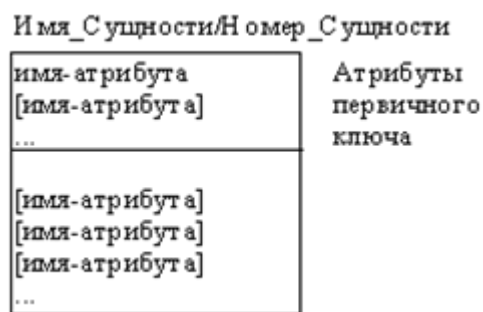


Рис. 54. – Атрибуты и первичные ключи

Сущности могут иметь также внешние ключи, которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK (Foreign Key) в скобках (рис. 55).

Пример внешнего ключа -
неключевого атрибута

Брокер/12	
номер брокера	
номер отдела (FK)	

Пример внешнего ключа -
атрибута первичного ключа

Заявка-на-покупку/2	
номер-заказа (FK)	
номер-товара	

Рис. 55. – Примеры внешних ключей

Подводя итог, отметим, что в результате развития структурного подхода и CASE-технологий были созданы и успешно применяются при проектировании ИС следующие методологии:

- IDEF0 (Integrated Definition Function Modeling) – методология функционального моделирования. Используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, преобразуемые этими функциями. Более известна как методологии SADT.

- DFD (Data Flow Diagram) – методология моделирования потоков данных. Применяется для описания обмена данными между рабочими процессами.

- IDEF1 применяется для построения информационной модели, отображающей структуру и содержание информационных потоков, необходимых для поддержки функций системы.

- IDEF2 позволяет построить динамическую модель меняющихся во времени поведения функций, информации и ресурсов системы.

- IDEF3 – методология моделирования потоков работ. Является более детальной по отношению к IDEF0 и DFD. Позволяет рассмотреть конкретный процесс с учетом последовательности выполняемых операций. С помощью IDEF3 описываются сценарий и последовательность операций для каждого процесса.

- IDEF1X (IDEF1 Extended) – расширенная методология описания данных. Применяется для построения БД. Относится к типу диаграмм «сущность-связь», как правило, используется для моделирования РБД, имеющих отношение к рассматриваемой ИС.

- IDEF4 – объектно-ориентированная методология. Отражает взаимодействие объектов. Позволяет наглядно отображать структуру объектов и заложенные принципы их взаимодействия. Удобна для создания программных продуктов на объектно-ориентированных языках.

Этот список можно продолжить, но другие методологии более специфичны и не рассматриваются в данном курсе.

58. Архитектурные решения ИС

Проектирование и разработка ИС может базироваться на разных архитектурных решениях. Кратко рассмотрим классификацию возможных архитектур ИС.

1. Архитектурное решение, основанное на использовании выделенных файл-серверов. При работе на файл-серверных архитектурах сохраняется автономность прикладного (и большей части системного) программного обеспечения, работающего на каждом компьютере сети. Фактически, компоненты ИС, выполняемые на разных компьютерах, взаимодействуют только за счет наличия общего хранилища файлов, которое хранится на файл-сервере. В классическом случае на каждом компьютере дублируются не только прикладные программы, но и средства управления базами данных. Файл-сервер представляет собой разделяемое всеми компьютерами комплекса расширение дисковой памяти. Таким образом, в файл-серверной архитектуре мы имеем «толстого» клиента и очень «тонкий» сервер в том смысле, что почти вся работа выполняется на стороне клиента, а от сервера требуется только достаточная емкость дисковой памяти.

2. «Клиент-серверное» архитектурное решение. Под клиент-серверным приложением понимают ИС, основанную на использовании серверов баз данных. Сервером баз данных называют программное обеспечение, привязанное к соответствующей базе (базам) данных, существующее независимо от наличия пользовательских (клиентских) процессов и выполняемое (хотя и не обязательно) на выделенной аппаратуре. Интерфейс между клиентской частью приложения и клиентской частью сервера баз данных, как правило, основан на использовании языка SQL. Клиентская часть сервера баз данных, используя средства сетевого доступа, обращается к серверу баз данных, передавая ему текст оператора языка SQL.

Отметим, что обычно компании, производящие развитые серверы баз данных, стремятся к тому, чтобы обеспечить возможность использования своих продуктов не только в стандартных на сегодняшний день TCP/IP-ориентированных сетях, но в сетях, основанных на других протоколах (например, SNA или IPX/SPX). Поэтому при организации сетевых взаимодействий между клиентской и серверной частями, СУБД часто используются не стандартные средства высокого уровня (например, вызовов удаленных процедур), а собственные функционально подобные средства, менее зависящие от особенностей сетевых транспортных протоколов. В связи с этим, несмотря на титанические усилия по стандартизации языка SQL, нет такой его реализации, в которой стандартные средства языка не были бы расширены. Необдуманное использование расширений языка приводит к полной зависимости приложения от конкретного производителя сервера баз данных.

В клиент-серверной организации клиенты могут являться достаточно «тонкими», а сервер должен быть «толстым» настолько, чтобы быть в состоянии удовлетворить потребности всех клиентов.

С другой стороны, разработчики и пользователи ИС, основанных на архитектуре «клиент-сервер», часто бывают неудовлетворенны постоянно существующими сетевыми накладными расходами, которые следуют из потребности обращаться от клиента к серверу с каждым очередным запросом. На практике распространена ситуация, когда для эффективной работы отдельной клиентской составляющей информационной системы в действительности требуется только небольшая часть общей БД. Это приводит к идее поддержки локального КЭШа общей БД на стороне каждого клиента. В таком случае, клиенты становятся более «толстыми» при том, что сервер «тоньше» не делается.

3. Архитектура КИС, базирующаяся на технологии Internet (Intranet-приложения). Возникновение и внедрение в широкую практику высокоуровневых служб Internet (e-mail, ftp, telnet, Gopher, WWW и т.д.) естественным образом повлияли на технологию создания КИС, породив направление, известное теперь под названием Intranet. По сути дела, информационная Intranet-система – это корпоративная система, в которой используются методы и средства Internet. Такая система может быть локальной, изолированной от остального мира Internet, или опираться на виртуальную корпоративную подсеть Internet. В последнем случае особенно важны средства защиты информации от несанкционированного доступа.

4. Архитектура ИС, основанная на концепции «склада данных» (DataWarehouse) – интегрированной информационной среды, включающей разнородные информационные ресурсы.

5. Архитектура, предназначенная для построения глобальных распределенных информационных приложений с интеграцией информационно-вычислительных компонентов на основе объектно-ориентированного подхода.

Остановимся более подробно на организации доступа прикладной клиентской программы к серверу базы данных в рамках «клиент-серверной» архитектуры.

Как правило, любой поставщик СУБД предоставляет вместе со своей системой внешнюю или встроенную утилиту, которая позволяет вводить операторы SQL в режиме командной строки и выдает на консоль результаты их выполнения. Недостатки такого режима работы – это необходимость очень хорошо знать SQL, помнить схему БД, невозможность удобного просмотра результатов выполнения запросов. Поэтому подобные утилиты стали инструментами администраторов баз данных, а для создания пользовательских приложений используются универсальные и специализированные языки программирования. Приложения, написанные таким образом, позволяют пользователю сосредоточиться на решении собственных задач, а не на структуре данных в базе.

Кроме этого, SQL позволяет только манипулировать данными, но в нем отсутствуют средства создания экранного интерфейса, что необходимо для пользовательских приложений. Для создания этого интерфейса служат универсальные языки программирования третьего поколения (C, C++ и т.д.)

или объектно-ориентированные языки четвертого поколения (xBase, Informix 4Gl, Progress, Jam и т.д.). Все они содержат необходимые операторы ввода/вывода на экран, управляющие операторы (циклы, ветвления и т.д.), допускают определение структур, соответствующих записям таблиц обрабатываемой БД. В текст программ включаются операторы языка SQL, во время исполнения передающиеся серверу БД, который собственно и производит манипулирование данными. Результаты, полученные при выполнении сервером SQL-запросов, возвращаются прикладной программе, которая заполняет строками этих результатов заранее определенные структуры. Дальнейшая работа клиентской программы (отображение, корректировка записей) ведется с этими извлеченными из БД структурами [6].

59. Организация доступа программ к серверу БД (использование специализированных библиотек и встраиваемого SQL, интерфейс CLI)

Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку доступа и набор драйверов для различных операционных систем. Схема взаимодействия клиентского приложения с сервером базы данных в этом случае приведена на рис. 56.

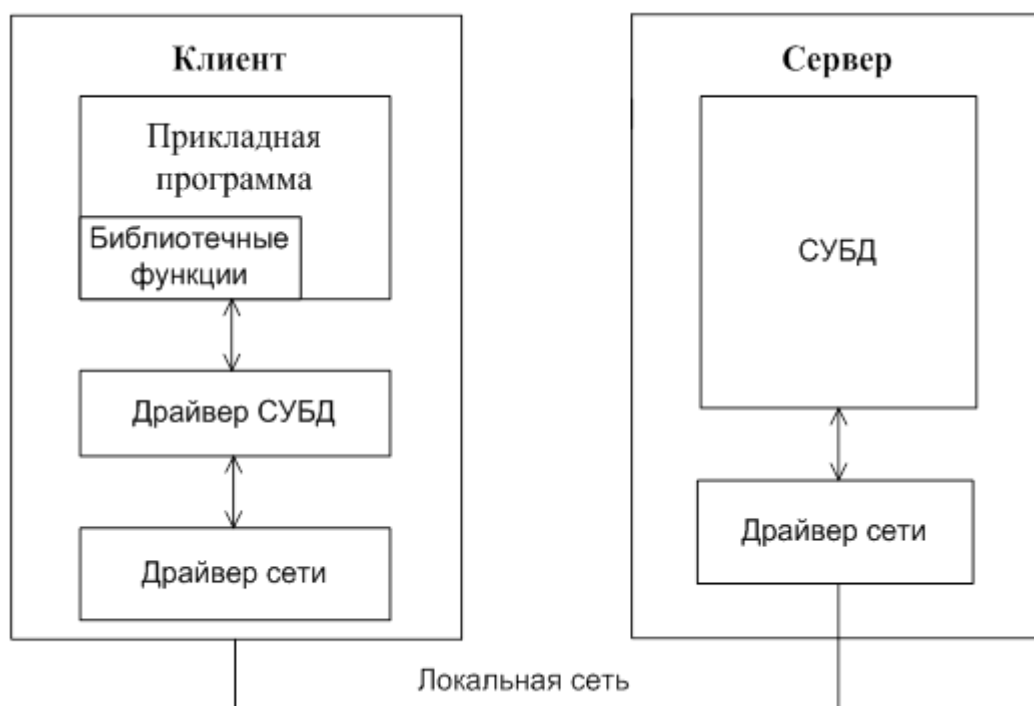


Рис. 56. – Схема взаимодействия клиентского приложения с сервером базы данных

Библиотека доступа – это, как правило, объектный файл, исходный код которого создан на универсальном языке типа С. Эта библиотека содержит набор функций, позволяющих пользовательскому приложению соединиться с базой данных, передавать запросы серверу и получать ответные данные. Типичный минимальный набор функций такой библиотеки (имена функций зависят от используемой библиотеки) для работы с БД на сервере приведен ниже:

- `DB_connect (char *имя_базы_данных, char *имя_пользователя, char *пароль)` – устанавливает соединение с БД, возвращает указатель на структуру `db`, описывающую характеристики этого соединения или ошибку, если подключиться не удалось;
- `DB_exec (db, char *запрос)` – выполняет запрос к БД, определяемой структурой `db`. Применяется для любых запросов кроме `SELECT`. Возвращает код выполнения запроса (0 – успешно, код ошибки – неудачно);

□ DB_select (db, char *запрос) – выполнить запрос на извлечение данных (SELECT). Возвращает структуру result, содержащую результаты выполнения запроса (реляционное отношение);

□ DB_fetch (result) – извлечь следующую запись из структуры result;

□ DB_close (db) – закрыть соединение с базой данных;

Обычно в библиотеке присутствуют также функции, позволяющие определить характеристики структуры result (число, порядок и имена столбцов, число строк, номер текущей строки), передвигаться по этой структуре не только вперед, но и назад (DB_next, DB_prev) и т.д.

Программа, использующая библиотеку связи с БД, может иметь следующий вид:

```
#include <dblib.h> /* Файл, содержащий описание функций библиотеки */
/* Организация интерфейса с пользователем, запрос его имени и пароля */
/* Присвоение значений переменным: dbname – имя базы данных */
/* username – имя пользователя */
/* password – пароль */
db=DB_connect (dbname,username,password); /* Установление соединения */
if (db == NULL) {
    error_message(); /* Выдача сообщения об ошибке на монитор пользователя */
    exit(1); /* Завершение работы */
}
/* Ожидание запроса пользователя. Формирование строки s_query – запроса */
/* на выборку данных */
result=DB_select(db,s_query); /* Пересылка запроса на сервер */
if (result==NULL) {
    error_message(); /* Ошибка выполнения запроса. Выдача сообщения */
    exit(2); /* Завершение работы */
}
/* Вывод результатов запроса на монитор пользователя. Ожидание следующего */
/* запроса. Подготовка строки u_query="UPDATE ... SET ...", содержащей */
/* запрос на изменение данных. */
res=DB_exec(db,u_query); /* Пересылка запроса на сервер */
if (res != 0) {
    error_message(); /* Ошибка выполнения запроса. Выдача сообщения */
    exit(2); /* Завершение работы */
}
DB_close(db); /* Завершение работы */
```

Данная программа, обеспечивающая взаимодействие пользователя с СУБД, компилируется совместно с библиотекой доступа. Библиотечные вызовы преобразуются драйвером БД в сетевые вызовы и передаются сетевым программным обеспечением на сервер БД.

На сервере происходит обратный процесс преобразования: сетевые пакеты, функции библиотеки, SQL-запросы, обрабатываются, их результаты передаются клиенту.

Такой способ создания приложений чрезвычайно гибок, позволяет реализовать практически любое приложение, но в то же время имеет явные недостатки:

- разработка клиентской программы возможна только для той операционной системы и на том языке программирования, который поддерживается библиотекой;
- необходим драйвер БД, который определяет допустимые типы сетевых интерфейсов;
- большой объем кодирования и декодирования данных;
- нестандартизованные библиотечные функции.

В результате получается приложение, которое привязано как к сетевой среде, так и к программно-аппаратной платформе и используемой БД.

Некоторой модификацией данного способа является использование «встроенного» языка SQL. В этом случае в текст программы на языке третьего поколения включаются не вызовы библиотек, а непосредственно предложения SQL, которые предваряются ключевым выражением «EXEC SQL». Перед компиляцией в машинный код такая программа обрабатывается препроцессором, который транслирует смесь операторов собственного языка СУБД и операторов SQL в чистый исходный код. Затем коды SQL замещаются вызовами соответствующих процедур из библиотек исполняемых модулей, служащих для поддержки конкретной СУБД.

Такой подход позволил несколько снизить степень привязанности к СУБД, например, при переключении прикладной программы на работу с другим сервером базы данных достаточно было заново обработать ее исходный текст новым препроцессором и перекомпилировать [6].

Большим достижением явилось появление в 1994 г. в стандарте SQL интерфейса уровня вызова – CLI (Call Level Interface), в котором стандартизован общий набор рабочих процедур, обеспечивающий совместимость со всеми основными серверами баз данных. Ключевой элемент CLI – специальная библиотека для компьютера-клиента, в которой хранятся вызовы процедур и большинство часто используемых сетевых компонентов для организации связи с сервером. Это ПО поставляется разработчиком средств SQL, не является универсальным и поддерживает разнообразные транспортные протоколы.

Использование программных вызовов позволяет свести к минимуму операции на компьютере-клиенте. В общем случае клиент формирует оператор языка SQL в виде строки и пересылает ее на сервер посредством процедуры исполнения (execute). Когда же сервер в качестве ответа возвращает несколько строк данных, клиент считывает результат с помощью серии вызовов процедуры выборки данных. Далее информация из столбцов полученной таблицы может быть связана с соответствующими переменными приложения. Вызов специальной процедуры позволяет клиенту определить считанное число строк, столбцов и типы данных в каждом столбце.

Интерфейс CLI построен таким образом, что перед передачей запроса серверу клиент не должен заботиться о типе оператора SQL, будь то выборка, обновление, удаление или вставка [6].

60. Интерфейсы ODBC и JDBC.

ODBC – открытый интерфейс к БД на платформе MS Windows

Важный шаг к созданию переносимых приложений обработки данных сделала фирма Microsoft, опубликовавшая в 1992 году программный интерфейс ODBC (Open Database Connectivity – открытый интерфейс доступа к базам данных). ODBC основан на спецификации CLI и предназначен для унификации программного взаимодействия клиентских приложений, работающих под управлением операционной системы Windows, с СУБД, делает его независимым от поставщика СУБД и программно-аппаратной платформы. Структурная схема доступа к данным с использованием ODBC приведена на рис. 57.

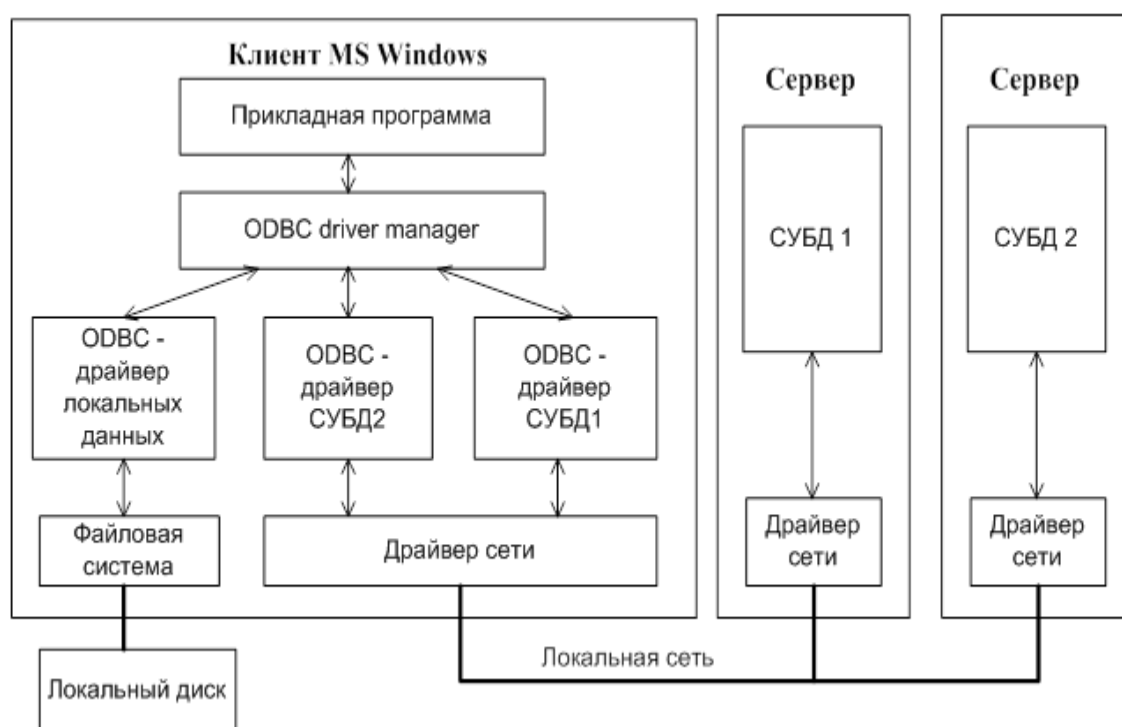


Рис. 57. – Структурная схема доступа к данным с использованием ODBC

За реализацию особенностей доступа к каждой отдельной СУБД отвечает специальный ODBC-драйвер. Пользовательское приложение этих особенностей не видит, т.к. взаимодействует с универсальным программным слоем более высокого уровня. Таким образом, приложение становится в значительной степени независимым от СУБД. Однако этот способ также не лишен недостатков:

- ☐ приложения становятся привязанными к платформе MS Windows;
- ☐ увеличивается время обработки запросов (как следствие введения дополнительного программного слоя);
- ☐ необходимо предварительная инсталляция ODBC-драйвера и настройка ODBC (указание драйвера, сетевого пути к серверу, базе данных и

т.д.) на каждом рабочем месте. Параметры настройки являются статическими, т.е. приложение их самостоятельно изменять не может [6].

JDBC (Java DataBase Connectivity) – это интерфейс прикладного программирования (API) для выполнения SQL-запросов к базам данных из программ, написанных на языке Java или, другими словами, это платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.

Структура подключения к БД с помощью JDBC (рис. 58) во многом подобна ODBC, JDBC также построен на основе спецификации CLI, однако имеет ряд преимуществ.

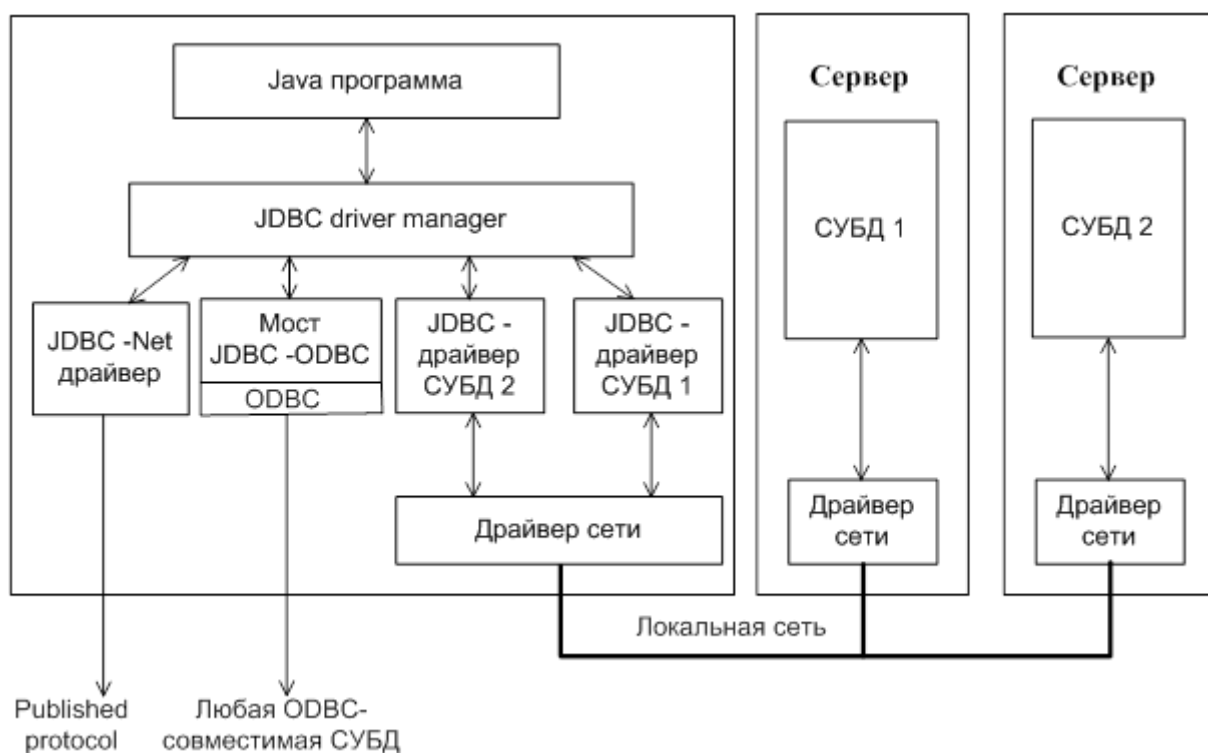


Рис. 58. – Структурная схема доступа к данным с использованием JDBC

Во-первых, приложение загружает JDBC-драйвер динамически, а, следовательно, администрирование клиентов упрощается, более того, появляется возможность переключаться на работу с другой СУБД без перенастройки клиентского рабочего места.

Во-вторых, JDBC, как и Java в целом, не привязан к конкретной аппаратной платформе, и, как следствие, проблемы с переносимостью приложений практически снимаются.

В-третьих, использование Java-приложений и связанной с ними идеологии «тонких клиентов» снижает требования к оборудованию клиентских рабочих мест [6].

При создании JDBC были установлены следующие приоритеты:

- передача запроса к БД в виде строки. Таким образом, могут использоваться конструкции SQL, специфичные для данной базы данных и/или ее JDBC-драйвера.

- универсальный метод работы с базами данных. JDBC предполагает конкретные требования к базовому SQL. Базовым требованием JDBC является удовлетворение входного уровня ANSI-стандарта SQL-92 (SQL-2). Такие требования обусловлены наличием определенного понятийного поля, то есть минимального набора предопределенных сущностей, характерных для реляционных СУБД.

- сохранение по возможности строгой статической типизации, что должно обеспечить большую степень защиты от ошибок на уровне компиляции в байт-код. Однако это требование не является абсолютным, так как SQL DML является по своей природе динамическим. Например, такие сущности, как наборы строк результирующих множеств, содержат динамически определяемое число и типы столбцов. Несколько утрируя данную позицию создателей спецификации, можно сказать, что «динамичности» предостаточно у самого SQL.

- увеличение числа методов интерфейсов, с одновременной «атомизацией» функциональности этих методов. Определяя работу по принципу «одно действие – один метод», подход должен обеспечить лучшую читаемость и прозрачность логики кода. Другими словами, лучше использовать набор простых методов, чем многоцелевые методы с большими наборами параметров-флагов.

25. ОБЪЕКТНО-РЕЛЯЦИОННЫЕ СУБД

Этот способ объединения возможностей реляционного и объектно-ориентированного подхода к управлению данными предложил известный американский ученый Майкл Стоунбрейкер. Согласно его воззрениям реляционную СУБД нужно просто дополнить средствами доступа к сложным данным. При этом ядро СУБД не требует переработки и сохраняет все присущие реляционным системам достоинства. Объектные расширения реализуются в виде надстроек, которые динамически подключаются к ядру. На основе этой идеи под руководством М. Стоунбрейкера в университете Беркли (Калифорния, США) была разработана СУБД Postgres, которая имеет следующие ключевые возможности.

1. Типы, операторы и методы доступа, определяемые пользователем. Для решения проблем с примером о хранении земельных участков, можно определить новый тип данных «участок», необходимые операции над ним (например, вычисление площади), а также метод доступа, поскольку с помощью В-дерева нельзя выполнить двумерный поиск в задаче о перекрывающихся многоугольниках. Здесь целесообразно использовать дерево более высокой размерности (R-дерево) или другие методы.

2. Поддержка сложных объектов, представляющих собой наборы других объектов.

3. Перегрузка операторов манипулирования данными.

4. Создание функций, определяемых пользователем.

5. Динамическое (т.е. без прерывания работы СУБД) добавление новых типов данных, операторов, функций и методов доступа. Описание всех этих возможностей создается на языке C и компилируется в объектный файл, который может динамически загружаться сервером СУБД.

6. Наследование данных и функций. Например, от типа «участок» мы можем породить потомков «обычный участок» (сумма_налога, поступление_платежей) и «участок освобожденный от налога» (сумма_налога, причина_освобождения).

7. Использование массивов как значений полей кортежей. Это необходимо, например, для хранения ставки налога, изменяющейся в зависимости от времени года.

Кроме того, СУБД Postgres обладает свойствами, которые позволяют назвать ее темпоральной СУБД. При любом обновлении записи создается ее новая копия, а предыдущий вариант продолжает существовать вечно. Даже после удаления записи все накопленные варианты сохраняются в БД. Можно извлечь из БД любой вариант записи, если указать момент или интервал времени, когда этот вариант был текущим. Достижение этих свойств позволило пересмотреть схемы журнализации и отката транзакций.

Все вышеописанные функции развиваются и в коммерческой СУБД Informix. Тем не менее, проект Postgres продолжается до сих пор, уже международной группой независимых разработчиков. К возможностям СУБД была добавлена поддержка SQL, поэтому несколько было изменено

название – PostgreSQL, оптимизатор запросов на основе генетических алгоритмов и многое другое. При этом PostgreSQL остается свободно распространяемой системой, причем бесплатно можно получить как исходный код, так и бинарные файлы, собранные для той или иной платформы (поддерживаются практически все разновидности ОС Unix).

26. ЭТАПЫ РАЗРАБОТКИ БАЗЫ ДАННЫХ И ОСНОВНЫЕ ПОДХОДЫ К ПРОЕКТИРОВАНИЮ

Существует два подхода к проектированию реляционных БД.

Первый подход заключается в том, что на этапе концептуального проектирования создается не концептуальная модель данных, а непосредственно реляционная схема БД, состоящая из определений реляционных таблиц, в дальнейшем подвергающихся нормализации.

Второй подход основан на механическом или автоматизированном преобразовании функциональной модели, созданной ранее, в нормализованную реляционную модель. Этот подход чаще всего используется при проектировании больших, сложных схем БД, необходимых для КИС (корпоративные информационные системы).

Целью разработки любой БД является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты: реляционная модель данных – удобный способ представления данных предметной области и язык SQL – универсальный способ манипулирования такими данными.

При разработке любой БД происходит постепенный переход от анализа предметной области к конкретной реализации БД средствами выбранной СУБД. Выделяют следующие этапы этого процесса перехода:

1. анализ предметной области;
2. модель предметной области;
3. логическая модель данных;
4. физическая модель данных;
5. собственно БД и приложения.

Предметная область – это часть реального мира, данные о которой мы хотим отразить в БД. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как важные понятия и данные, так и малозначащие или вообще ничего не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия "накладная" и "счет-фактура" являются важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей – это для учета товаров неважно. Однако с точки зрения отдела кадров данные о наличии детей являются важными. Таким образом, важность данных зависит от выбора предметной области.

Модель предметной области. Модель предметной области – это отражение наших знаний о предметной области. Знания могут быть как в

виде неформальных суждений в голове эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке БД являются описания предметной области, выполненные при помощи специализированных графических нотаций. Имеется большое количество графических методик описания предметной области. Из наиболее известных можно назвать следующие: метод структурного анализа (SADT) и основанные на нем IDEF0, IDEF1 диаграммы, диаграммы потоков данных Гейна-Сарсона, метод объектно-ориентированного анализа (UML) и др. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений и БД.

Логическая модель данных. На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий: «сотрудник», «отдел», «проект», «зарплата». Примеры взаимосвязей между понятиями: «сотрудник числится ровно в одном отделе», «сотрудник может выполнять несколько проектов», «над одним проектом может работать несколько сотрудников». Примеры ограничений: «возраст сотрудника не менее 16 и не более 60 лет», «дата поставки не может быть меньше текущей даты», «в поле могут быть только перечисленные в списке значения».

Логическая модель данных является начальным прототипом будущей БД. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД. Более того, логическая модель данных необязательно должна быть выражена средствами именно реляционной модели данных. Основным средством разработки логической модели данных являются различные варианты ER-диаграмм (Entity-Relationship). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако, т.к. мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных.

Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения. Например, модель предметной области складского учета содержит понятия "склад", "накладная", "товар". При разработке соответствующей реляционной модели эти термины обязательно должны быть использованы, но различными способами реализации много – можно создать одно отношение, в котором будут присутствовать в

качестве атрибутов "склад", "накладная", "товар", а можно создать три отдельных отношения, по одному на каждое понятие.

При разработке логической модели данных должны решаться вопросы: хорошо ли спроектированы отношения? Правильно ли они отражают модель предметной области, а, следовательно, и саму предметную область?

Физическая модель данных. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Понятия, выбранные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преобразуются в типы данных, принятые в конкретной реляционной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом опять-таки решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных, принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным.

При разработке физической модели данных возникают вопросы: хорошо ли спроектированы таблицы? Правильно ли выбраны индексы? Сколько программного кода в виде триггеров и хранимых процедур необходимо разработать для поддержания целостности данных?

Собственно БД и приложения. Результатом предыдущих этапов является собственно сама БД на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с БД. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы.

Решения, принятые на предыдущем уровне – уровне физического проектирования, снова определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД.

Таким образом, решения, принятые на каждом этапе моделирования и разработки БД, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений на ранних этапах моделирования.

27. КРИТЕРИИ ОЦЕНКИ КАЧЕСТВА ЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ

Рассмотрим некоторые принципы построения хороших логических моделей данных. Хороших в том смысле, что решения, принятые в процессе логического проектирования приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе БД.

Для того чтобы оценить качество принимаемых решений на уровне логической модели данных, необходимо сформулировать некоторые критерии качества в терминах физической модели и конкретной реализации и посмотреть, как различные решения, принятые в процессе логического моделирования, влияют на качество физической модели и на скорость работы БД.

Конечно, таких критериев может быть очень много и выбор их в достаточной степени произволен. Рассмотрим некоторые из таких критериев.

1. Адекватность БД предметной области. Адекватность означает, что должны выполняться следующие условия:

- состояние БД в каждый момент времени должно соответствовать состоянию предметной области;
- изменение состояния предметной области должно приводить к соответствующему изменению состояния БД;
- ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться БД.

2. Легкость разработки и сопровождения БД. Практически любая БД, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.

Хранимые процедуры – это процедуры и функции, хранящиеся непосредственно в БД в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с БД. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур – реализация бизнес-процессов предметной области.

Триггеры – это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы БД. В качестве таких событий выступают обычно операции вставки(INSERT), обновления(UPDATE) и удаления(DELETE) строк таблиц. Если в БД определен некоторый триггер, то он запускается автоматически всегда при возникновении события, с которым он связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее его запуск. Таким образом, основное назначение триггеров – автоматическая

поддержка целостности БД. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими ограничения предметной области или действия, которые должны произойти при наступлении некоторых событий.

Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия: проверяет, есть ли необходимое количество товара; при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе; при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.

3. Скорость выполнения операций изменения данных. На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять – это распределение атрибутов по различным отношениям. Можно использовать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо ответить на вопрос – влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций изменения данных.

Основными операциями, изменяющими состояние БД, являются операции вставки, обновления и удаления кортежей. В БД, требующих постоянных изменений (складской учет, продажа билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления.

Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки уменьшается при увеличении количества индексов у таблицы и мало зависит от числа строк в таблице.

Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы до нахождения нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но на самом деле неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам. При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы

объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость выполнения операций обновления и удаления также уменьшается при увеличении количества индексов у таблицы и мало зависит от числа строк в таблице.

Можно предположить, что чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но при одинаковых подходах к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что чем больше атрибутов имеют отношения, разработанные в ходе логического моделирования, тем медленнее будут выполняться операции обновления данных, за счет затраты времени на перестройку большего количества индексов.

4. Скорость выполнения операций выборки данных. Одно из назначений БД – предоставление информации пользователям. Информация извлекается из реляционной БД при помощи оператора SQL – SELECT. Одной из наиболее дорогостоящих операций (в смысле использования ресурсов) при выполнении оператора SELECT является операция соединения таблиц. Таким образом, чем больше взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, увеличение количества отношений приводит к замедлению выполнения операций выборки данных, особенно, если запросы заранее неизвестны.

28. ПРОЕКТИРОВАНИЕ БД НА ОСНОВЕ НОРМАЛИЗАЦИИ ОТНОШЕНИЙ: 1NF И АНАЛИЗ АНОМАЛИЙ 1NF

Рассмотрим классический подход, при котором процесс проектирования БД производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений.

Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Нормализация – это разбиение таблицы на две или более, обладающих лучшими свойствами (по отношению к начальному) при добавлении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта БД, в котором каждый факт появляется лишь в одном месте, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

Каждая таблица в реляционной БД удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не

может быть множества таких значений. Любая таблица, удовлетворяющая этому условию, называется нормализованной. Фактически, ненормализованные таблицы, т.е. таблицы, содержащие повторяющиеся группы, не допускаются в реляционной БД.

Всякая нормализованная таблица автоматически считается таблицей в первой нормальной форме (1NF). Таким образом, понятия «нормализованная» и «находящаяся в 1NF» означают одно и то же. Однако на практике термин «нормализованная» часто используется в более узком смысле – «полностью нормализованная», который означает, что в проекте не нарушаются никакие принципы нормализации.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

В качестве общих свойств для всех нормальных форм можно отметить:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих сохраняются.

Первая нормальная форма (1NF) – это обычное отношение. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, будем считать, что исходный набор отношений уже соответствует этому требованию. Напомним свойства отношений (это и есть свойства 1NF):

- в отношении нет одинаковых кортежей;
- кортежи не упорядочены;
- атрибуты не упорядочены и различаются по наименованию;
- все значения атрибутов атомарны.

Пример 2. Рассмотрим в качестве примера предметной области организацию, выполняющую некоторые проекты [8]. Модель предметной области опишем следующим неформальным текстом:

Сотрудники организации выполняют проекты.

Проекты состоят из нескольких заданий.

Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.

Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.

Над каждым заданием в проекте работает только один сотрудник.

Каждый сотрудник числится в одном отделе.

Каждый сотрудник имеет телефон, находящийся в отделе сотрудника.

Также допустим, что в ходе уточнения того, какие данные необходимо учитывать, выяснились следующие факты.

О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.

Каждый отдел имеет уникальный номер.

Каждый проект имеет уникальный номер и наименование.

Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

В ходе логического моделирования на первом шаге предложено хранить данные в одном отношении СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ (Н_СОТР, ФАМ, Н_ОТД, ТЕЛ, Н_ПРО, ПРОЕКТ, Н_ЗАДАН), имеющем следующие атрибуты:

Н_СОТР – табельный номер сотрудника;

ФАМ – фамилия сотрудника;

Н_ОТД – номер отдела, в котором числится сотрудник;

ТЕЛ – телефон сотрудника;

Н_ПРО – номер проекта, над которым работает сотрудник;

ПРОЕКТ – наименование проекта, над которым работает сотрудник;

Н_ЗАДАН – номер задания, над которым работает сотрудник.

Так как каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве потенциального ключа отношения необходимо взять пару атрибутов (Н_СОТР, Н_ПРО).

Пусть в текущий момент состояние предметной области отражается нижеследующими фактами в табл. 2.

Сотрудник Иванов, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 1 и во втором проекте "Климат" задание 1.

Сотрудник Петров, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 2.

Сотрудник Сидоров, работающий во 2 отделе, выполняет в первом проекте "Космос" задание 3 и во втором проекте "Климат" задание 2.

Таблица
2. СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ

Н_СОТР	ФАМ	Н_ОТД	ТЕЛ	Н_ПРО	ПРОЕКТ	Н_ЗАДАН
1	Иванов	1	11-22-33	1	Космос	1
1	Иванов	1	11-22-33	2	Климат	1
2	Петров	1	11-22-33	1	Космос	2
3	Сидоров	2	33-22-11	1	Космос	3
3	Сидоров	2	33-22-11	2	Климат	2

Аномалии

Даже одного взгляда на табл. 2 СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ достаточно, чтобы увидеть, что данные хранятся в ней с большой избыточностью. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные о сотрудниках, об отделах, о проектах и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние БД, возникает большое количество проблем.

Проблемы получили название аномалий обновления. Аномалия обновления – это либо неадекватность модели данных предметной области, либо некоторые дополнительные трудности в реализации ограничений предметной области средствами СУБД.

Различают следующие виды аномалий:

- аномалии вставки (INSERT);
- аномалии обновления (UPDATE);
- аномалии удаления (DELETE).

Рассмотрим примеры аномалий в отношении СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ.

Аномалии вставки (INSERT). В отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ нельзя вставить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11, null, null, null). Это сделать невозможно, т.к. атрибут Н_ПРО (номер проекта) входит в состав потенциального ключа, и, следовательно, не может содержать null-значений. Точно также нельзя вставить данные о проекте, над которым пока не работает ни один сотрудник.

Аномалии обновления (UPDATE). Фамилии сотрудников, наименования проектов, номера телефонов повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, или проект меняет наименование,

или меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где эти данные встречаются, иначе отношение станет некорректным (например, один и тот же проект в разных кортежах будет называться по-разному). Таким образом, обновление БД одним действием реализовать невозможно. Для поддержания отношения в целостном состоянии необходимо написать триггер, который при обновлении одной записи корректно исправлял бы данные и в других местах. Как следствие, увеличивается сложность разработки и БД, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Аномалии удаления (DELETE). При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове. Если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11. Если по проекту временно прекращены работы, то при удалении данных о работах по этому проекту будут удалены и данные о самом проекте. При этом если бы был сотрудник, который работал только над этим проектом, то будут потеряны и данные об этом сотруднике.

Причина всех аномалий – хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту). Таким образом, логическая модель данных неадекватна модели предметной области и БД, основанная на такой модели, будет работать неправильно.

29. ОПРЕДЕЛЕНИЕ ФУНКЦИОНАЛЬНОЙ ЗАВИСИМОСТИ, 2NF, АНОМАЛИИ

Функциональная зависимость

Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ находится в 1NF, но при этом логическая модель данных не адекватна модели предметной области. Таким образом, первой нормальной формы недостаточно для правильного моделирования данных и для устранения аномалий применяется метод нормализации отношений.

Нормализация основана на понятии функциональной зависимости атрибутов отношения.

Определение функциональной зависимости: Пусть R – отношение. Множество атрибутов Y функционально зависит от множества атрибутов X (X функционально определяет Y) тогда и только тогда, когда для любого состояния отношения R для любых кортежей $r_1, r_2 \in R$ из того, что $r_1 * X = r_2 * X$ следует что $r_1 * Y = r_2 * Y$ (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X , значения атрибутов Y также совпадают в любом состоянии отношения R). Символически функциональная зависимость обозначается $X \rightarrow Y$.

Множество атрибутов X называется детерминантом функциональной зависимости, а множество атрибутов Y называется зависимой частью.

Если атрибуты X составляют потенциальный ключ отношения R , то любой атрибут отношения R функционально зависит от X .

В отношении СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ можно привести следующие примеры функциональных зависимостей:

- зависимость атрибутов от ключа отношения:

$(Н_СОТР, Н_ПРО) \rightarrow ФАМ;$

$(Н_СОТР, Н_ПРО) \rightarrow Н_ОТД;$

$(Н_СОТР, Н_ПРО) \rightarrow ТЕЛ;$

$(Н_СОТР, Н_ПРО) \rightarrow ПРОЕКТ;$

$(Н_СОТР, Н_ПРО) \rightarrow Н_ЗАДАН;$

- зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника: $Н_СОТР \rightarrow ФАМ;$ $Н_СОТР \rightarrow Н_ОТД;$ $Н_СОТР \rightarrow ТЕЛ;$

- зависимость наименования проекта от номера проекта: $Н_ПРО \rightarrow ПРОЕКТ;$

- зависимость номера телефона от номера отдела: $Н_ОТД \rightarrow ТЕЛ.$

Таким образом, функциональная зависимость – семантическое понятие. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефон. Функциональная зависимость задает дополнительные ограничения на данные, которые могут храниться в отношениях. Для корректности БД (адекватности предметной области) необходимо при выполнении операций модификации БД проверять все ограничения, определенные функциональными зависимостями.

Функциональная зависимость атрибутов отношения напоминает понятие функциональной зависимости в математике. Но это не одно и то же.

Отличие от математического понятия отношения состоит в том, что, если рассматривать математическое понятие функции, то для фиксированного значения $x \in X$ соответствующее значение функции $y = f(x)$ всегда одно и то же. Например, если задана функция $y = x^2$, то для значения $x = 2$ соответствующее значение y всегда будет равно 4. В противоположность этому в отношениях значение зависимого атрибута может принимать различные значения в различных состояниях БД.

Например, атрибут ФАМ функционально зависит от атрибута Н_СОТР. Предположим, что сейчас сотрудница с табельным номером 1 имеет фамилию Иванова, т.е. при значении детерминанта равного 1, значение зависимого аргумента равно «Иванова». Но сотрудница может сменить фамилию, например на Сидорова. Теперь при том же значении детерминанта, равного 1, значение зависимого аргумента равно «Сидорова».

Таким образом, функциональная зависимость атрибутов утверждает лишь то, что для каждого конкретного состояния БД по значению одного

атрибута (детерминанта) можно однозначно определить значение другого атрибута (зависимой части), но конкретные значения зависимой части могут быть различны в различных состояниях БД.

Вторая нормальная форма

Отношение R находится во второй нормальной форме (2NF) тогда и только тогда, когда отношение находится в 1NF и нет неключевых атрибутов, зависящих от части сложного ключа. Неключевой атрибут – это атрибут, не входящий в состав никакого потенциального ключа.

Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2NF.

Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ не находится в 2NF, т.к. есть атрибуты, зависящие от части сложного ключа:

– зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

$H_СОТР \rightarrow ФАМ;$

$H_СОТР \rightarrow H_ОТД;$

$H_СОТР \rightarrow ТЕЛ;$

– зависимость наименования проекта от номера проекта:

$H_ПРО \rightarrow ПРОЕКТ.$

Для того чтобы устранить зависимость атрибутов от части сложного ключа, нужно произвести декомпозицию отношения на несколько отношений. При этом те атрибуты, которые зависят от части сложного ключа, выносятся в отдельное отношение вместе с детерминантом.

Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ необходимо разбить на три отношения – СОТРУДНИКИ_ОТДЕЛЫ, ПРОЕКТЫ, ЗАДАНИЯ:

1. Отношение СОТРУДНИКИ_ОТДЕЛЫ ($H_СОТР$, $ФАМ$, $H_ОТД$, $ТЕЛ$) приведено в табл. 3:

Таблица 3. – Отношение СОТРУДНИКИ_ОТДЕЛЫ

$H_СОТР$	$ФАМ$	$H_ОТД$	$ТЕЛ$
1	Иванов	1	11- 22- 33
2	Петров	1	11- 22- 33
3	Сидоров	2	33- 22- 11

В отношении присутствуют следующие функциональные зависимости:

– зависимость атрибутов, характеризующих сотрудника, от табельного номера сотрудника:

$H_COTR \rightarrow \Phi AM;$

$H_COTR \rightarrow H_OTD;$

$H_COTR \rightarrow T\Phi L;$

– зависимость номера телефона от номера отдела: $H_OTD \rightarrow T\Phi L.$

Отношение ПРОЕКТЫ ($H_ПРО, ПРОЕКТ$) приведено в табл. 4:

Таблица 4. – Отношение ПРОЕКТЫ

$H_ПРО$	ПРОЕКТ
1	Космос
2	Климат

Функциональная зависимость: $H_ПРО \rightarrow ПРОЕКТ.$

2. Отношение ЗАДАНИЯ ($H_COTR, H_ПРО, H_ЗАДАН$) приведено в табл. 5.

Таблица 5. – Отношение ЗАДАНИЯ

H_COTR	$H_ПРО$	$H_ЗАДАН$
1	1	1
1	2	1
2	1	2
3	1	3
3	2	2

Функциональная зависимость: $(H_COTR, H_ПРО) \rightarrow H_ЗАДАН.$

Анализ декомпозированных отношений на аномалии. Отношения, полученные в результате декомпозиции, находятся в 2NF. Действительно, отношения СОТРУДНИКИ_ОТДЕЛЫ и ПРОЕКТЫ имеют простые ключи, следовательно, автоматически находятся в 2NF, отношение ЗАДАНИЯ имеет сложный ключ, но единственный неключевой атрибут $H_ЗАДАН$ функционально зависит от всего ключа ($H_COTR, H_ПРО$).

Часть аномалий обновления устранена. Так, данные о сотрудниках и проектах теперь хранятся в различных отношениях, поэтому при появлении сотрудников, не участвующих ни в одном проекте просто добавляются кортежи в отношение СОТРУДНИКИ_ОТДЕЛЫ. Точно также, при появлении проекта, над которым не работает ни один сотрудник, вставляется кортеж в отношение ПРОЕКТЫ.

Фамилии сотрудников и наименования проектов теперь хранятся без избыточности. Если сотрудник сменит фамилию или проект сменит наименование, то такое обновление будет произведено в одном месте.

Если по проекту временно прекращены работы, но требуется, чтобы сам проект сохранился, то для этого проекта удаляются соответствующие кортежи в отношении ЗАДАНИЯ, а данные о самом проекте и данные о сотрудниках, участвовавших в проекте, остаются в отношениях ПРОЕКТЫ и СОТРУДНИКИ_ОТДЕЛЫ.

Тем не менее, часть аномалий разрешить не удалось.

Оставшиеся аномалии вставки (INSERT). В отношении СОТРУДНИКИ_ОТДЕЛЫ нельзя вставить кортеж (4, Пушников, 1, 33-22-11), т.к. при этом получится, что два сотрудника из 1-го отдела (Иванов и Пушников) имеют разные номера телефонов, а это противоречит модели предметной области. В этой ситуации можно предложить два решения, в зависимости от того, что реально произошло. Другой номер телефона может быть введен по двум причинам – по ошибке человека, вводящего данные о новом сотруднике, или потому что номер в отделе действительно изменился. В любом случае необходимо написать триггер, который при вставке записи о сотруднике проверяет, совпадает ли телефон с уже имеющимся телефоном у другого сотрудника этого же отдела. Если номера отличаются, то система должна задать вопрос, оставить ли старый номер в отделе или заменить его новым. Если нужно оставить старый номер (новый номер введен ошибочно), то кортеж с данными о новом сотруднике будет вставлен, но номер телефона будет у него тот, который уже есть в отделе (в данном случае, 11-22-33). Если же номер в отделе действительно изменился, то кортеж будет вставлен с новым номером, и одновременно будут изменены номера телефонов у всех сотрудников этого же отдела.

Оставшиеся аномалии обновления (UPDATE). Одни и те же номера телефонов повторяются во многих кортежах отношения. Поэтому, если в отделе меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречается, иначе отношение станет некорректным. Таким образом, обновление БД одним действием реализовать невозможно. Необходимо и здесь написать триггер, который при обновлении одной записи корректно исправляет номера телефонов в других местах.

Причина аномалии – избыточность данных, порожденная тем, что в одном отношении хранится разнородная информация (о сотрудниках и об отделах) за счет чего увеличивается сложность разработки, и БД будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Оставшиеся аномалии удаления (DELETE). При удалении некоторых данных по-прежнему может произойти потеря другой информации. Например, если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11.

В итоге, логическая модель данных снова неадекватна модели предметной области и БД, основанная на такой модели, будет работать неправильно.

Заметим, что при переходе к 2NF отношения стали почти адекватными предметной области. Остались трудности, связанные с необходимостью написания триггеров, поддерживающих целостность БД, и они связаны только с одним отношением СОТРУДНИКИ_ОТДЕЛЫ.

Про аномалии: в предыдущем вопросе.

30. 3NF И ОБЩИЙ АЛГОРИТМ НОРМАЛИЗАЦИИ ДО 3NF

Атрибуты называются взаимно независимыми, если ни один из них не является функционально зависимым от другого.

Отношение находится в третьей нормальной форме (3NF) тогда и только тогда, когда отношение находится в 2NF и все неключевые атрибуты взаимно независимы.

Отношение СОТРУДНИКИ_ОТДЕЛЫ не находится в 3NF, т.к. имеется функциональная зависимость неключевых атрибутов (зависимость номера телефона от номера отдела): Н_ОТД → ТЕЛ.

Для того чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом те неключевые атрибуты, которые являются зависимыми, выносятся в отдельное отношение.

Отношение СОТРУДНИКИ_ОТДЕЛЫ нужно декомпонировать на два отношения – СОТРУДНИКИ и ОТДЕЛЫ.

Отношение СОТРУДНИКИ (Н_СОТР, ФАМ, Н_ОТД) включает следующие функциональные зависимости: Н_СОТР → ФАМ, Н_СОТР → Н_ОТД, Н_СОТР → ТЕЛ.

Таблица 6. – Отношение СОТРУДНИКИ

Н_СОТР	ФАМ	Н_ОТД
1	Иванов	1
2	Петров	1
3	Сидоров	2

Отношение ОТДЕЛЫ (Н_ОТД, ТЕЛ) включает функциональную зависимость номера телефона от номера отдела: Н_ОТД → ТЕЛ.

Таблица 7. – Отношение ОТДЕЛЫ

Н_ОТД	ТЕЛ
1	11-22-33
2	33-22-11

Отметим, что атрибут Н_ОТД, не являвшийся ключевым в отношении СОТРУДНИКИ_ОТДЕЛЫ, становится потенциальным ключом в отношении ОТДЕЛЫ. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов.

Таким образом, все обнаруженные аномалии обновления устранены. Реляционная модель данных, состоящая из четырех отношений – СОТРУДНИКИ, ОТДЕЛЫ, ПРОЕКТЫ, ЗАДАНИЯ, находится в 3NF, является адекватной описанной модели предметной области и предполагает наличие только тех триггеров, которые поддерживают

ссылочную целостность. Такие триггеры являются стандартными и не требуют больших усилий в разработке.

Алгоритм нормализации (приведение к 3NF)

Шаг 1 (приведение к 1NF). На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1NF.

Шаг 2 (приведение к 2NF). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводится декомпозиция этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа, выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты.

Пусть исходное отношение: $R (K_1, K_2, A_1, \dots, A_n, B_1, \dots, B_m)$.

Ключ: (K_1, K_2) – сложный.

Функциональные зависимости:

$(K_1, K_2) \rightarrow (K_1, K_2, A_1, \dots, A_n, B_1, \dots, B_m)$ – зависимость всех атрибутов от ключа отношения.

$(K_1) \rightarrow (A_1, \dots, A_n)$ – зависимость некоторых атрибутов от части сложного ключа.

Декомпозированные отношения:

$R_1 (K_1, K_2, B_1, \dots, B_m)$ – остаток от исходного отношения. Ключ (K_1, K_2) .

$R_2 (K_1, A_1, \dots, A_n)$ – новое отношение, где помещаются атрибуты, вынесенные из исходного вместе с частью сложного ключа. Ключ (K_1) .

Шаг 3 (приведение к 3NF). Если в отношениях обнаружена зависимость одних неключевых атрибутов от других неключевых атрибутов, то проводится декомпозиция этих отношений следующим образом: те неключевые атрибуты, которые зависят от других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости.

Пусть исходное отношение: $R (K, A_1, \dots, A_n, B_1, \dots, B_m)$. Ключ: (K) .

Функциональные зависимости:

$(K) \rightarrow (A_1, \dots, A_n, B_1, \dots, B_m)$ – зависимость всех атрибутов от ключа отношения.

$(A_1, \dots, A_n) \rightarrow (B_1, \dots, B_m)$ – зависимость некоторых неключевых атрибутов от других неключевых атрибутов.

Декомпозированные отношения:

$R_1 (K, A_1, \dots, A_n)$ – остаток от исходного отношения. Ключ K .

$R_2(A_1, \dots, A_n, B_1, \dots, B_m)$ – новое отношение, где атрибуты, вынесены из исходного отношения вместе с детерминантом функциональной зависимости. Ключ (A_1, \dots, A_n) .

На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно сразу строят отношения в 3NF. Кроме того, основным средством разработки логических моделей данных являются различные варианты ER-диаграмм. Особенность диаграмм заключается в том, что они сразу позволяют создавать отношения в 3NF. Тем не менее, приведенный алгоритм важен по двум причинам.

Во-первых, этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений.

Во-вторых, модель предметной области никогда не бывает правильно разработана с первого шага. Эксперты предметной области могут забыть о чем-либо упомянуть, разработчик может неправильно понять эксперта, во время разработки могут измениться правила, принятые в предметной области, и т.д. Все это может привести к появлению новых зависимостей, которые отсутствовали в первоначальной модели предметной области. Тут как раз и необходимо использовать алгоритм нормализации хотя бы для того, чтобы убедиться, что отношения остались в 3NF и логическая модель не ухудшилась.

У слабо нормализованных отношений есть единственное преимущество – если к БД обращаться только с запросами на выборку данных, то они будут выполняться быстрее. Это связано с тем, что в слабо нормализованных отношениях не нужно выполнять операцию соединения отношений и на это не тратится время при выборке данных.

Таким образом, выбор степени нормализации отношений зависит от типа операций, которые планируется чаще выполнять над данными в базе.

31. OLTP И OLAP-СИСТЕМЫ

Сильно и слабо нормализованные модели данных послужили основанием для создания двух классов систем.

Сильно нормализованные модели данных хорошо подходят для так называемых OLTP-приложений (On-Line Transaction Processing – оперативная обработка транзакций). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, «снять сумму денег со счета А, добавить эту сумму на счет В». Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, они выполняются одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-

третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции (не должно быть ситуации, когда деньги сняты со счета А, но не поступили на счет В). Практически все запросы к БД в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора информации из различных справочников системы. Большая часть таких запросов, как правило, известна заранее, еще на этапе проектирования системы.

Таким образом, критическим для OLTP-приложений является скорость и надежность выполнения коротких операций изменения данных. Чем выше уровень нормализации данных в OLTP-приложении, тем они, как правило, быстрее и надежнее. Отступления от этого правила могут происходить тогда, когда уже на этапе разработки известны некоторые часто возникающие запросы, требующие соединения отношений, и от скорости выполнения которых существенно зависит работа приложений. В этом случае можно жертвовать нормализацией для ускорения выполнения подобных запросов.

Другим типом приложений являются так называемые OLAP-приложения (On-Line Analytical Processing – оперативная аналитическая обработка данных). Это обобщенный термин, характеризующий принципы построения систем поддержки принятия решений (Decision Support System – DSS), хранилищ данных (Data Warehouse), систем интеллектуального анализа данных (Data Mining). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа типа «что если...?». OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми из электронных таблиц или из других источников данных. Такие системы характеризуются следующими общими признаками:

- добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения);
- данные, добавленные в систему, обычно никогда не удаляются;
- перед загрузкой данные проходят различные процедуры «очистки», связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны;
- запросы к системе являются нерегламентированными и, как правило, достаточно сложными, причем очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса;
- скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж. Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы типа «у какого подразделения самые лучшие объемы продаж в текущем году?», или «каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?».

Физически гиперкуб может быть построен на основе специальной многомерной модели данных (MOLAP – Multidimensional OLAP) или средствами реляционной модели данных (ROLAP – Relational OLAP).

Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных (ROLAP), данные целесообразно хранить в виде слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Большая избыточность и связанные с ней проблемы здесь не страшны, т.к. обновление происходит только в момент загрузки новой порции данных и при этом происходит пересчет итогов.

32. КОРРЕКТНОСТЬ ПРОЦЕДУРЫ НОРМАЛИЗАЦИИ. ТЕОРЕМА ХЕЗА

Как было показано выше, алгоритм нормализации состоит в выявлении функциональных зависимостей предметной области и соответствующей декомпозиции отношений.

При декомпозиции из одного отношения получаем два или более отношений, каждое из которых содержит часть атрибутов исходного отношения. В полученных новых отношениях необходимо удалить дубликаты строк, если таковые возникли. Это означает, что декомпозиция отношения есть не что иное, как взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали (возможно, с повторениями) все атрибуты исходного отношения. Т.е., при декомпозиции не должны теряться атрибуты отношений и сами данные.

Данные можно считать не потерянными в том случае, если возможна обратная операция: по декомпозированным отношениям можно восстановить исходное отношение в точности в прежнем виде. Операцией, обратной операции проекции, является операция соединения отношений. Существует несколько видов операции соединения, но т.к. при восстановлении исходного отношения из проекций не должны появиться новые атрибуты, то используют естественное соединение.

Проекция $R[X]$ отношения R на множество атрибутов X называется собственной проекцией, если множество атрибутов X является собственным подмножеством множества атрибутов отношения R (т.е. множество атрибутов X не совпадает с множеством всех атрибутов отношения R).

Собственные проекции R_1 и R_2 отношения R называются декомпозицией без потерь, если отношение R точно восстанавливается из них при помощи естественного соединения для любого состояния отношения R : $R_1 \text{ join } R_2 = R$.

Рассмотрим пример, показывающий, что декомпозиция без потерь происходит не всегда. Пусть дано отношение R :

Таблица 8. – Отношение R

Номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	1000

Рассмотрим первый вариант декомпозиции отношения R на два отношения R_1 и R_2 :

Таблица 9. –

Отношение R_1

Номер	Зарплата
1	1000
2	1000

Таблица 10. – Отношение R_2

Фамилия	Зарплата
Иванов	1000
Петров	1000

Естественное соединение этих проекций, имеющих общий атрибут Зарплата, будет следующим (каждая строка одной проекции соединится с каждой строкой другой проекции):

Таблица 11. – Отношение $R_1 \text{ join } R_2 \neq R$

Номер	Фамилия	Зарплата
1	Иванов	1000
1	Петров	1000
2	Иванов	1000
2	Петров	1000

Данная декомпозиция не является декомпозицией без потерь, т.к. исходное отношение не восстанавливается в точном виде по его проекциям (серым цветом выделены лишние кортежи).

Рассмотрим другой вариант декомпозиции:

Таблица 12. – Отношение R_1

Номер	Фамилия
1	Иванов
2	Петров

Таблица 13. – Отношение R_2

Номер	Зарплата
1	1000
2	1000

По данным проекциям, имеющим общий атрибут Номер, исходное отношение восстанавливается в точном виде. Тем не менее, нельзя сказать, что данная декомпозиция является декомпозицией без потерь, т.к. рассмотрено только одно конкретное состояние отношения R , и нельзя сказать, будет ли и в других состояниях отношение R восстанавливаться точно. Например, предположим, что отношение R перешло в следующее состояние:

Таблица 14. – Отношение R в новом состоянии

Номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	1000
2	Сидоров	2000

Кажется, что этого не может быть, т.к. значения в атрибуте Номер повторяются. Но ведь ничего и не было сказано о ключе этого отношения. Сейчас проекции будут иметь вид:

Таблица 15. – Отношение R_1

Номер	Фамилия
1	Иванов
2	Петров
2	Сидоров

Таблица 16. – Отношение R_2

Номер	Зарплата
1	1000
2	1000
2	2000

Естественное соединение этих проекций снова будет содержать лишние кортежи:

Таблица 17. – Отношение $R_1 \text{ join } R_2 \neq R$

Номер	Фамилия	Зарплата
1	Иванов	1000
2	Петров	1000
2	Петров	2000
2	Сидоров	1000
2	Сидоров	2000

Таким образом, доказано, что без дополнительных ограничений на отношение R нельзя говорить о декомпозиции без потерь.

Необходимыми дополнительными ограничениями являются функциональные зависимости. Имеет место следующая теорема Хеза:

Пусть $R(A, B, C)$ является отношением, и A, B, C – атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость $A \rightarrow B$, то проекции $R_1 = R[A, B]$ и $R_2 = R[A, C]$ образуют декомпозицию без потерь.

Основной смысл теоремы Хеза заключается в доказательстве того, что не появятся новые кортежи, отсутствовавшие в исходном отношении.

Т.к. алгоритм нормализации (приведения отношений к 3NF) основан на имеющихся в отношениях функциональных зависимостях, то теорема Хеза показывает, что алгоритм нормализации является корректным, т.е. в ходе нормализации не происходит потери информации.

Нормализации до 3NF в большинстве случаев вполне достаточно, чтобы разрабатывать адекватные и работоспособные БД. Но для научного подхода к проектированию БД со сложной структурой зависимостей между данными используются нормальные формы более высоких порядков.