МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

*Е. И. ЧИГАРИНА*

# ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ ИНТЕГРИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.04.01 Информатика и вычислительная техника

САМАРА
Издательство Самарского университета
2025

УДК 004.65(075)
ББК А635.1я7
Ч-586

Рецензенты: д-р техн. наук, доц. С. В. С м и р н о в,
канд. техн. наук, доц. А. А. Л о б а н к о в

***Чигарина, Елена Ивановна***

Ч-586 **Проектирование баз данных интегрированных информационных систем:** учебное пособие / *Е. И. Чигарина*. – Самара: Издательство Самарского университета, 2025. – 68 с.: ил.

**ISBN 978-5-7883-2175-2**

В данном учебном пособии подробно рассмотрены этапы проектирования баз данных, выполнена ориентация на разработку систем баз данных, интегрированных в различные автоматизированные системы, рассмотрены информационные системы оперативной и аналитической обработки данных, вопросы проектирования баз данных и хранилищ данных.

Предназначено для студентов – магистров, обучающихся по очной форме по направлению подготовки 09.04.01 Информатика и вычислительная техника по курсу «Проектирование баз данных интегрированных информационных систем», и подготовлено на кафедре информационных систем и технологий Самарского аэрокосмического университета.

УДК 004.65(075)
ББК А635.1я7

ISBN 978-5-7883-2175-2
© Самарский университет, 2025

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

Одной из самых распространенных сфер применения вычислительной техники является создание различных автоматизированных систем. Среди них САПР (системы автоматизированного проектирования), АСУ (автоматизированные системы управления), АИС (автоматизированные информационные системы), АСНИ (автоматизированные системы научных исследований) и другие. Главное отличие этих системах – вид автоматизируемой человеческой деятельности. В САПР осуществляется автоматизация проектирования, в АСУ – управления (причем системы автоматизированного управления технологическими процессами называются АСУ ТП, управления производством – АСУП), в АИС – хранения и поиска данных, в АСНИ – научно-экспериментальных исследований. В отличие от автоматических систем, в автоматизированных системах часть функций выполняет человек и чаще всего это функции принятия решения.

Информационная система – автоматизированная система, предназначенная для хранения и обработки данных. Различают информационные системы, предназначенные для оперативной обработки данных и для аналитической обработки. Причем последние обычно проектируются после создания первых. Каждая из информационных систем использует базы данных и, возможно, хранилища данных.

Каждая из автоматизированных систем согласно государственным стандартам включает несколько основных видов обеспечений, в том числе техническое обеспечение, программное, *информационное,* организационное и правовое. Целью рассмотрения данного пособия является информационное обеспечение автоматизированных систем. Информационная система может быть интегрирована в любую автоматизированную систему. Информационные системы называют иначе системами баз данных. Вопросы, связанные с проектированием баз данных интегрированных информационных систем рассмотрены в данном учебном пособии.

# ГЛАВА 1. ЭТАП КОНЦЕПТУАЛЬНОГО ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

## 1.1. Уровни представления данных. Основные этапы проектирования баз данных. Определение требований

**Данные** – представление информации в формальном виде, пригодном для передачи, интерпретации или обработки (ГОСТ 20546-2021).

**База данных** – совокупность данных, организованная в соответствии с концептуальной структурой, в которой описаны характеристики этих данных и взаимосвязи между сущностями для одной или нескольких областей применения или обработки (ГОСТ 20546-2021).

В зависимости *от степени абстрагирования данных* выделяют три уровня представления данных: концептуальный уровень, логический и физический.

*Концептуальный уровень* – это семантический (смысловой) уровень представления данных в виде абстрактных понятий, учитывающих особенности предметной области. На данном уровне вводятся, например такие абстрактные понятия как сущность и связь, агрегация и обобщение. Названия этих понятий для разных моделей концептуального уровня представления данных отличаются, но в целом они определяют особенности объектов предметной области и взаимосвязи между ними.

*Логический уровень* – уровень представления данных в виде структуры данных, к которым относятся иерархические, сетевые, реляционные структуры и другие виды структур, используемых при организации данных в вычислительных системах.

*Физический уровень* – уровень представления данных, учитывающий способ организации данных на машинном носителе в виде совокупности файлов различной структуры.

На каждом уровне представления данных имеются различные модели представления данных. Фактически синонимом понятия модели данных является понятие схемы базы данных. *Схема базы данных* – описание базы данных. По аналогии с уровнями представления данных различают три типа схем баз данных в зависимости от уровня абстракции трехуровневой архитектуры.

Схема создается в процессе проектирования базы данных. Совокупность информации, хранящаяся в базе данных в определенный момент времени, называется состоянием. Таким образом, одной и той же схеме данных может соответствовать несколько состояний базы данных. Схема базы данных иногда называется содержанием базы данных, а ее состояние детализацией. Схема базы данных показывает логическую организацию всей базы данных в целом, а *подсхема* – описание части базы данных, описание представления о данных отдельного пользователя или приложения.

**Этапы проектирования баз данных информационных систем**:

1. Анализ предметной области и определение требований к базе данных.
2. Построение концептуальной модели (схемы) базы данных (этап концептуального проектирования).
3. Выбор структуры данных.
4. Построение логической модели базы данных (этап логического проектирования).

5. Выбор средства реализации базы данных.

6. Построение физической модели базы данных (этап физического проектирования).

Предметная область информационной системы – совокупность реальных процессов и объектов, представляющих интерес для её пользователей.

Первый этап проектирования базы данных имеет следующие особенности.

Процесс анализа предметной области является итерационным, требует многократного общения с заказчиком на разработку системы базы данных. При анализе необходимо уточнить цель создания системы, определить функции, реализуемые системой базы данных.

При анализе хранимых данных необходимо определить место хранения данных – память одного или нескольких компьютеров, или облако, доступ к данным будет иметь один или несколько пользователей, необходимо уточнить характеристики пользователей.

Хранить необходимо те данные, для которых в системе реализуется поиск, выборка данных, которые используются в документах (отчетах), формируемых из данных базы данных. Поэтому кроме описания предметной области нужно знать запросы и выходные документы (отчеты), формируемые при работе с базами данных.

При определении требований к базе данных выполняется оценка качественных и количественных критериев. К качественным критериям относятся возможность восстановления базы данных после сбоев, возможность расширения или изменения структуры базы данных, наличие средств защиты информации, обеспечение целостности данных и другие критерии. К количественным критериям относятся ограничения на объем данных, ограничения на время отклика на запросы пользователей, стоимость хранения данных, стоимость обновления и другие критерии.

### 1.2. Основные понятия модели сущность-связь. Класс принадлежности сущности

Наибольшее распространение для процесса концептуального проектирования получила концептуальная модель данных типа «сущность – связь» (модель Entity – Relation, ER – модель, модель Чена). Основными абстрактными понятиями этой модели являются понятия сущности, экземпляра сущности, атрибута и связи.

*Сущность* – реальный или мыслимый объект предметной области. Сущность характеризуется свойствами – *атрибутами*. Каждая сущность состоит из множества *экземпляров сущности*. Например, сущность «Студент» имеет экземпляры сущности – сведения о конкретных студентах. Фамилия и номер зачётной книжки студента – это характеристики или атрибуты сущности «Студент».

Между сущностями устанавливаются связи. *Связь* – соответствие или отображение между элементами двух или более множеств (между экземплярами сущностей).

Различают следующие виды или типы связей:

*1:1* – связь один к одному между сущностями. Связь «один к одному» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует один экземпляр другой сущности,

*1:M (*или *M:1*) – связь один ко многим (или многие к одному). Связь «один ко многим» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует несколько экземпляров другой сущности,

*M:N* – связь многие ко многим. Связь «многие ко многим» между сущностями устанавливается, если каждому экземпляру одной сущности соответствует несколько экземпляров другой сущности и наоборот.

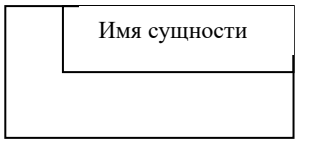
*Класс принадлежности сущности* в связи с другой сущностью является *обязательным*, если каждому экземпляру сущности соответствует хотя бы один экземпляр связанной сущности.

*Класс принадлежности сущности* в связи с другой сущностью является *необязательным*, если имеются экземпляры сущности, для которых отсутствует соответствующий экземпляр связанной сущности

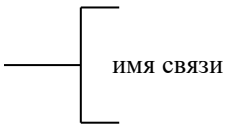
## 1.3. Графические примитивы модели сущность-связь. Диаграммы Бахмана, диаграммы Хау

Для описания или представления модели данных кроме введенных абстрактных понятий необходимо использование некоторых графических примитивов, позволяющих эту модель представить. Существует множество вариантов графического представления модели сущность – связь. Наиболее известными являются диаграммы Бахмана и Чена.

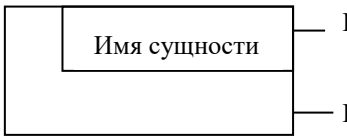
На диаграммах Бахмана для описания модели сущность-связь используются следующие графические примитивы:




- представление сущности

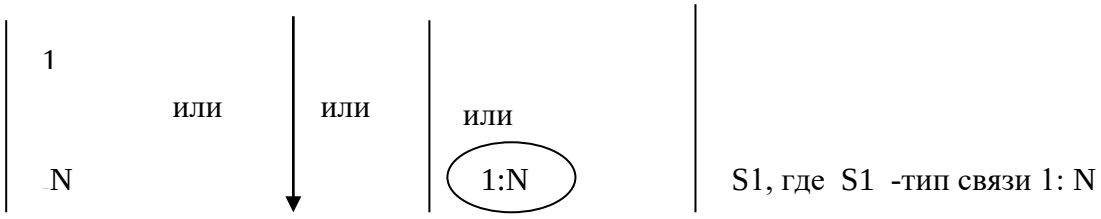



- представление связи




- представление атрибутов сущности

Допустимые варианты представления мощности связи -



Для графического представления класса принадлежности сущности на линии связи указывается либо линия (класс принадлежности сущности в связи обязательный), либо окружность (класс принадлежности необязательный).

Например, дано описание следующей предметной области: в городе имеется несколько предприятий, на каждом из которых работают сотрудники. Сотрудники характеризуются должностью, фамилией, именем, отчеством (фио), окладом, возрастом, адресом. Предприятие характеризуется названием, числом сотрудников, фио директора. Каждый сотрудник может работать на одном предприятии, на одном предприятии работают несколько сотрудников. Описание модели средствами диаграммы Бахмана представлено на рисунке 1.1.

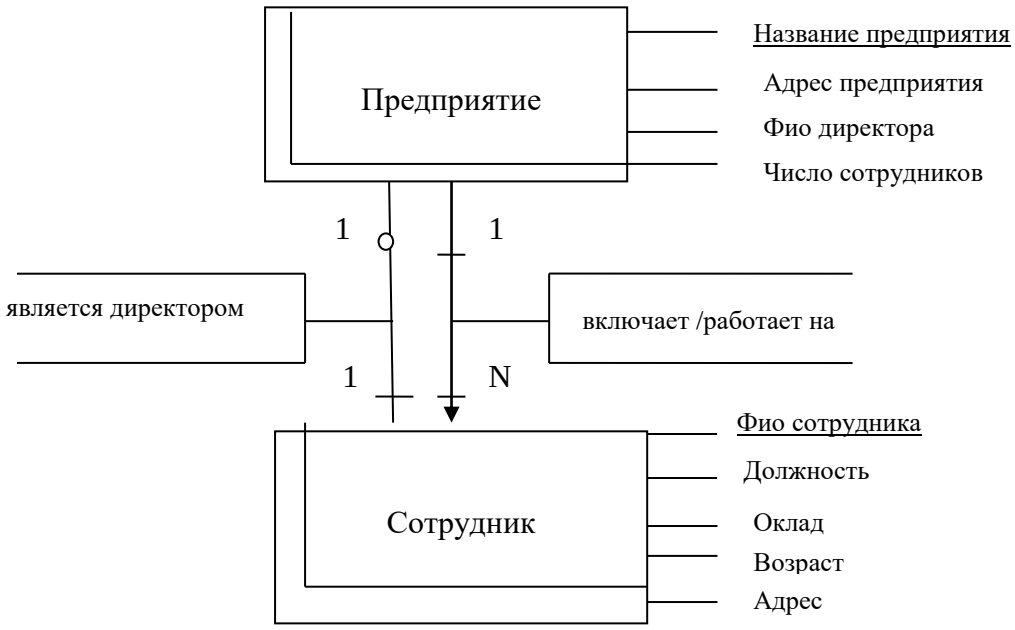


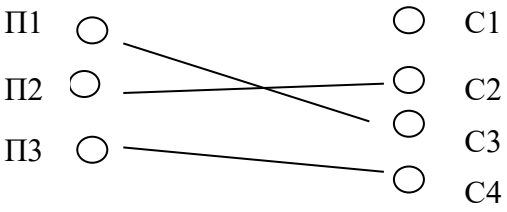

Рисунок 1.1 – Пример диаграммы Бахмана модели сущность-связь

На диаграмме Бахмана подчеркнутые атрибуты имеют неповторяющиеся значения, то есть однозначно определяют каждый экземпляр сущности (понятие ключевого атрибута). При описании модели имеет значение направление связи между сущностями. В связи с этим при проведении связи вводят понятие «родительской» и «дочерней сущности». Для связи «один к одному» с именем «является директором» родительской является сущность «Сотрудник», а дочерней «Предприятие», тогда с точки зрения логики модели связь читается как «сотрудник является директором предприятия». Имя связи на модели может читаться не только от родительской сущности по направлению к дочерней сущности, но и наоборот. Так на рисунке 1.1 связь «один ко многим» от родительской сущности «Предприятие» имеет имя «включает», а прочтение этой же связи от дочерней сущности «Сотрудник» выглядит как «работает на». Полностью эту связь можно назвать как «Сотрудник работает на Предприятии». Не каждый сотрудник является директором предприятия, поэтому класс принадлежности сущности Сотрудник в связи «является директором» с сущностью Предприятие является необязательным.

Альтернативой диаграмм Бахмана, используемых при построении модели сущность-связь являются диаграммы Чена, которые исторически появились первыми. С точки зрения за-

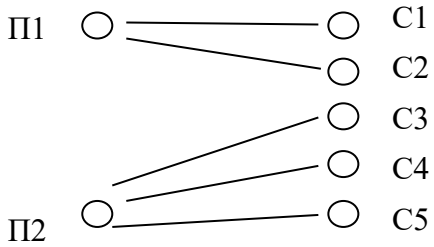
нимаемого пространства при описании модели сущность-связь диаграмма Чена менее предпочтительна и более громоздка, тем не менее, также как и диаграмма Бахмана описывает логику взаимосвязей сущностей предметной области [9].

Диаграммы ХАУ наглядно показывают разницу в классах принадлежности сущности. Например, для связи «является директором» диаграмма Хау выглядит следующим образом.




Слева показаны экземпляры сущности Предприятие, справа экземпляры сущности Сотрудник. Как видно, есть сотрудники, не являющиеся директорами, но каждое предприятие имеет директора. Мощность связи здесь один к одному.

Диаграмма Хау для связи «включает/работает на» выглядит следующим образом.




Как видно из этой диаграммы каждый сотрудник работает на одном предприятии, на каждом предприятии работают несколько сотрудников. Класс принадлежности и Предприятия и Сотрудника обязательные, так как нет предприятия, где не работает ни одного сотрудника, и нет сотрудника, который нигде не работает.

Ту или иную форму графического представления модели сущность-связь выбирает проектировщик базы данных, но обязательно на любой диаграмме модели должны присутствовать сущности, атрибуты, при этом ключевые атрибуты выделены особо, при указании связи приводится класс принадлежности сущности, имя и мощность связи. В настоящее время кроме приведенных диаграмм для описания моделей сущность-связь используются автоматизированные графические инструменты, например средство draw.io, имеющее возможность построения диаграмм сущность-связь.

### 1.4. Пример проектирования концептуальной схемы базы данных. Рекомендации для этапа концептуального проектирования баз данных

Пример описания предметной области – на предприятии для проведения испытаний новых изделий создаются комиссии из сотрудников разных подразделений. Подразделения характеризуются номером и названием подразделения. Каждый сотрудник может входить в не-

сколько комиссий, одна комиссия может проводить несколько испытаний. Каждая комиссия имеет своего председателя, индекс комиссии, состоящий из набора букв и цифр. Сотрудник характеризуется ФИО, табельным номером, может иметь паспортные данные. Сотрудник может быть совместителем или штатным сотрудником. Для совместителя устанавливается процент ставки, для штатного сотрудника устанавливается оклад. Новые изделия могут быть разных видов. Над одним видом изделия проводятся испытания разных видов. Испытание изделия (имеет номер изделия) оценивается комиссией по пятибалльной системе, для него планируется дата проведения, причем фактическая дата не всегда совпадает с плановой датой. Каждое испытание проводится одной комиссией.

При построении концептуальной схемы базы данных на первом шаге выделяются важные, существенные для хранения объекты предметной области (сущности). Для рассматриваемого примера это – Подразделение, Сотрудник, Комиссия и Испытание. Эти сущности можно назвать *основными*. Продолжая анализ этих сущностей, можно заметить у них ряд особенностей. Например, испытание может проводиться для разных видов изделия и сами испытания могут быть разного вида. В данном случае можно ввести *справочные* сущности Вид изделия и Вид испытания. Экземпляр сущности Сотрудник может быть либо штатным сотрудником, либо совместителем, причем с разным количеством атрибутов. Поэтому можно выделить *категориальные* сущности Штатный сотрудник и Совместитель. Кроме этого у сотрудника указана такая характеристика, как паспортные данные, которая может и не указываться для сотрудника. Паспортные данные в свою очередь имеют множество атрибутов, включая номер паспорта, серию, место выдачи и другие параметры. Поэтому можно выделить *дополнительную* сущность Паспортные данные.

На следующем шаге для каждой сущности определяется состав атрибутов. Например, для сущности Подразделение атрибуты – Номер подразделения и Название подразделения предприятия.

Далее из выделенных атрибутов выделяются ключевые атрибуты, если это возможно на данном этапе. Причем ключевые атрибуты могут быть атомарными (состоять из одного атрибута) или составными (состоять из множества атрибутов). В общем случае, ключевой атрибут – это множество атрибутов сущности, однозначно определяющее каждый экземпляр сущности. Ключевым атрибутом сущности Подразделение является Номер подразделения. Ключевым атрибутом Паспортные данные является составной атрибут, состоящий из номера паспорта, серии и атрибута «кем выдан». Из множества атрибутов сущности Испытание выделить ключевой на данном этапе проектирования сложно, так как значения всех атрибутов этой сущности могут повторяться, поэтому для этой сущности выделение ключевого атрибута можно отложить на более поздний этап проектирования базы данных.

На следующем шаге определяются связи между сущностями, указывается мощность связей, класс принадлежности сущностей и имена связей.

В результате концептуального проектирования получена семантическая модель, представленная на рисунке 1.2.

**Рекомендации для этапа концептуального проектирования баз данных.**

**1. Выделить основные сущности** – то есть существенные, необходимые для хранения понятия, сведения предметной области. На модели имена сущностей записываются в единственном числе именительного падежа. *Уточнить наличие справочных, категориальных и дополнительных сущностей.*

1) Если есть понятия предметной области, которые при использовании в качестве атрибута основной сущности имеют значения многократно повторяющиеся для разных экземпляров сущности, то рекомендуется эти понятия вынести в отдельные сущности как *справочные сущности*, устранив многократное повторение одного и того же значения атрибута для разных экземпляров сущностей. Например – сущности: «Вид испытания», «Вид изделия» являются справочными сущностями.

2) Если есть понятие предметной области, которое уточняется разным составом атрибутов, то рекомендуется ввести категории основной сущности, выделив *категориальные* сущности. Например, сущности – Штатный сотрудник, Совместитель. Если имеются характеристики основной сущности (логически связанные между собой, например, паспортные данные или адрес), у которых большое число атрибутов, при этом по логике они являются дополнительными (иногда и не обязательными для хранения), то их можно вынести в отдельную сущность. Это делает модель более наглядной. Такую сущность можно назвать *дополнительной* и она будет связана с основной сущностью связью один к одному. Например, сущность «Паспортные данные» является дополнительной сущностью.

**2. Определить атрибуты сущностей.** Указать их имена на модели.

**3. Определить ключевые атрибуты.** Для каждой полученной сущности (основной, справочной, категориальной, дополнительной) анализируется наличие множества атрибутов, однозначно определяющих экземпляры сущности (ключевой атрибут). Причем на концептуальном уровне не всегда возможно реализовать определение ключевых атрибутов. При концептуальном проектировании, если отсутствуют ключевые атрибуты выделенных сущностей, процесс добавления «суррогатных» ключей (атрибутов, не входящих в перечень свойств сущности, а являющихся фактически номером, id экземпляра сущности, который не повторяется) рекомендуется отложить на более поздний этап проектирования, так как с учетом структуры данных введение суррогатного ключа может и не понадобиться.

**4. Определить связи между сущностями.** *Для каждой связи, при этом необходимо указываются имена связей, определить мощность (степень, вид) связей и класс принадлежности сущностей.* Имена связей для типа связей n:m и 1:1 рекомендуется указывать на модели в обе стороны (например, имя связи входит/состоит можно указать для связи между сущностями Сотрудник и Комиссия). Допускается применение предлогов в именах связей (входит в /состоит из). Это позволяет точнее описать модель.

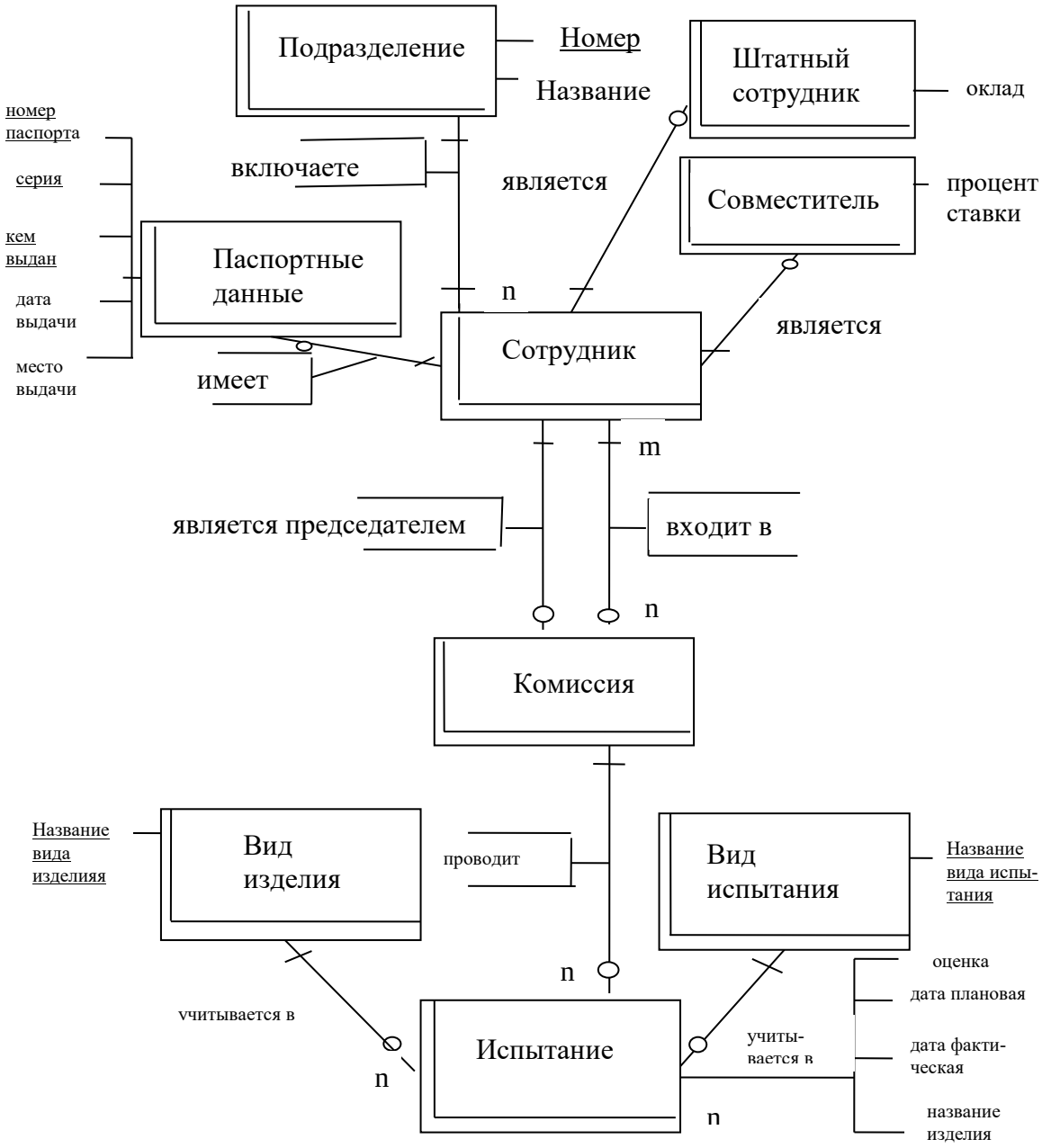



Рисунок 1.2 – Концептуальная модель базы данных «Испытания изделий»

# ГЛАВА 2. ЭТАП ВЫБОРА СТРУКТУРЫ ДАННЫХ

## 2.1. Модели логического уровня представления данных

Структура данных – это способ организации и хранения данных в памяти компьютера. На логическом уровне данные представляются в виде структур, к которым относятся древовидные структуры, графы, таблицы, объекты. Соответственно к моделям данных логического уровня представления данных относятся иерархическая модель, сетевая, реляционная, объектно-ориентированная и другие.

В случае применения *иерархической модели данных* предполагается, что предметная область может быть разбита на отдельные части, которые можно именовать и между данными устанавливаются иерархические отношения. Иерархическая модель реализуется в виде дерева, в узлах которого могут находиться сущности, экземпляры сущностей, атрибуты и значения атрибутов. Древовидная структура обязательно имеет корень дерева, узлы-предки и узлы-потомки. Узлы дерева, не уточняемые на более низких уровнях иерархии, называются концевыми узлами дерева. На рисунке 2.1 приведен пример иерархической модели данных.


Рисунок 2.1 – Пример иерархической модели данных

Реализация связей типа «многие ко многим» на иерархических структурах приводит к необходимости дублирования информации. Например, для реализации такой связи между S и A, показанной на рисунке 2.2, надо хранить два варианта связи, иначе при поиске данных может быть выбран только один из вариантов, а не все.


Рисунок 2.2 – Дублирование информации при реализации связей «многие ко многим» на иерархической модели

Достоинством иерархической модели является тот факт, что существуют хорошо развитые алгоритмические средства реализации древовидных структур, а также существенным фактором является наглядность и ясность иерархических структур. К недостаткам иерархической модели следует отнести избыточность при реализации связей типа «многие ко многим» и проблемы при манипулировании данными. Так, удалив узел-предок, удаляются все узлы-потомки, а это не всегда удобно. Кроме этого невозможно хранить порожденный узел без исходного уз-

ла, то есть нужен пустой исходный узел. Для ликвидации этих недостатков была предложена сетевая модель данных.

Примерами СУБД, реализующих иерархическую модель данных, являются СУБД семейства IMS и System.

*Сетевая модель* логического уровня представляется в виде графа, возможно имеющего циклы и петли. В узлах графа могут находиться сущности, экземпляры сущностей, атрибуты, значения атрибутов. Ребрами графа являются связи между объектами графа. Модель типа "сущность – связь" лучше всего трансформируется именно на сетевую модель данных. Наиболее известная сетевая модель данных – это модель, созданная рабочей группой по базам данных (КОДАСИЛ). Эта модель состоит из множества сущностей, которые могут быть владельцами или членами наборов данных.

Примером СУБД, реализующей сетевую модель данных КОДАСИЛ, является СУБД db_VISTA. Эта СУБД предназначена для создания и использования баз данных сложной структуры в рамках различных программных систем, реализованных на языке "С".

Отличительная черта сетевых моделей – высокая общность, любую модель можно представить в виде сетевой модели, в этом её главное достоинство. К недостатку модели следует отнести сложность реализации такой модели, отсутствие простых и легко реализуемых алгоритмов работы с сетевыми структурами. С целью преодоления названных недостатков была разработана реляционная модель данных самая распространённая в настоящее время.

В *реляционной модели данных* предметная область разбивается на множества, которые могут быть связаны между собой по элементам, то есть могут быть выделены упорядоченные наборы элементов, объединенные в одно множество. Предполагается, что правила или алгоритмы известны и по ним могут быть выделены эти наборы. Множество получившихся наборов называется *отношением между элементами* (релейшен). Эта модель в настоящее время строго и точно формализуема. Отношение удобно представлять в виде двумерной таблицы, которая привычна и наглядна для человека. *Реляционная база данных* представляет собой совокупность взаимосвязанных отношений. Каждое отношение в ЭВМ представляется в виде файла. На рисунке 2.3 приведено соответствие понятий, используемых в реляционных моделях и присутствующих на различных уровнях описания данных.

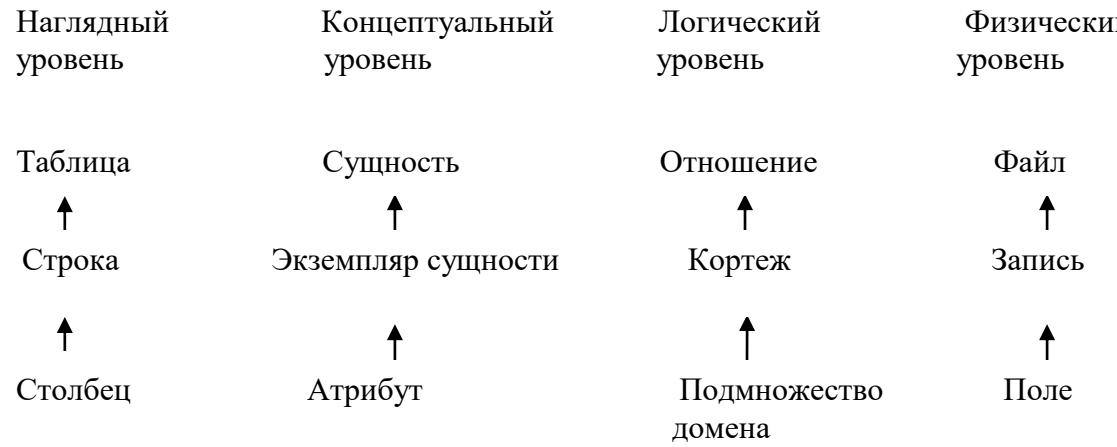



Рисунок 2.3 – Соответствие понятий реляционной модели на различных уровнях представления данных

Таблицы не учитывают необходимости согласования трех способов манипулирования данными: упорядочение, группировку по значению индексов, доступ по дереву параметров. В таблице данные жестко связаны между собой – упорядочиваются данные по одному параметру и, следовательно, доступ по нему облегчается, а по другому параметру при этом усложняется. Но главное достоинство реляционной модели состоит в наличии строгой, формализованной теории реляционной алгебры и реляционного исчисления, лежащих в основе языков манипулирования данными в реляционных баз данных.

Примерами СУБД, поддерживающими реляционную модель данных, являются PostgreeSQL, Oracle, MS SQL Server, FoxPro, Clipper, Paradox и многие другие.

При реализации баз данных используются и объектно-ориентированные модели данных. Следуя практике многих объектно-ориентированных баз данных (ООБД), предлагается выделить два уровня моделирования объектов: нижний (структурный) и верхний (поведенческий). На структурном уровне поддерживаются сложные объекты, их идентификация и разновидности связи "isa". ООБД – это набор элементов данных, связанных отношениями "входит в класс" или "является атрибутом". Таким образом, ООБД может рассматриваться как ориентированный граф и фактически соответствует семантической иерархической модели концептуального уровня представления данных. Важным аспектом является четкое разделение схемы базы данных и самой базы данных. В качестве первичных концепций схемного уровня ООБД (объектно-ориентированных баз данных) выступают типы и классы. Предполагается, что для точного определения ООБД требуется уровень метасхемы, содержимое которой должно определять виды объектов и связей, допустимых на схемном уровне базы данных. Мета-схема должна играть для ООБД такую же роль, какую играет структурная часть реляционной модели данных для схем реляционных баз данных.

Примерами СУБД, реализующими объектную модель данных можно назвать СУБД O2, ORION, GemStone, Iris. СУБД ORACLE иногда называют объектно-реляционной.

### 2.2. Рекомендации по выбору структуры данных

1. При наличии на концептуальной модели связей многие ко многим не использовать иерархическую структуру данных.

2. При выборе учитывать навыки проектировщика и опыт по работе с определенными структурами данных.

3. С точки зрения наглядности, наличия развитой теории реляционной алгебры и реляционного исчисления кортежей, математической обоснованности алгоритмов по ведению и манипулированию данных наиболее предпочтительна реляционная модель.

# ГЛАВА 3. ЭТАП ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

## 3.1. Этапы логического проектирования реляционных баз данных

*Отношение* – множество кортежей (упорядоченных наборов данных), являющихся подмножеством декартового произведения доменов (множество однотипных данных).

*Схема отношения* – это перечень имён атрибутов отношения. Предполагается, что студенты знакомы с теоретическими основами реляционных баз данных, которые изучают в курсе «Базы данных» [1, 2, 3, 9].

Можно выделить следующие этапы логического проектирования реляционных баз данных, представляющих собой совокупность взаимосвязанных отношений.

1. На основе концептуальной схемы базы данных по правилам Джексона построить схемы предварительных отношений (ненормализованных отношений).

2. Выполнить анализ схем предварительных отношений на нормальные формы и, при необходимости, выполнить нормализацию отношений.

3. С помощью CASE- средства проектирования базы данных построить логическую схему базы данных, например, используя методологии IDEF1X или IE.

## 3.2. Правила Джексона перехода от концептуальной модели к схемам предварительных отношений базы данных. Рекомендации по определению состава ключей при переходе к схемам отношений по правилам Джексона. Описание схем отношений

1. Если мощность бинарной связи 1:1 и класс принадлежности обеих сущностей обязательный, то необходимо одно отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

*Дополнение*: можно оставить два отношения как у исходных сущностей, если количество атрибутов в одной из сущностей большое и по логике является дополнительной сущностью, кроме этого данные дополнительной сущности редко участвует в выборках данных и необязательно присутствуют в основной сущности.

2. Если мощность бинарной связи 1:1 и класс принадлежности одной сущности обязательный (родительская сущность), а другой необязательный (дочерняя сущность – сущность *к которой* проводится связь), то необходимо два отношения, при этом ключ сущности с обязательным классом принадлежности добавляется в качестве дополнительного атрибута в отношение, соответствующее сущности с необязательным классом принадлежности. Ключи сущностей служат первичными ключами отношений.

*Дополнение:* если на концептуальном уровне первичный ключ сущности с классом принадлежности необязательным не был определен, то ключ сущности с обязательным классом принадлежности, переходящий в дочернюю сущность может использоваться в качестве первичного ключа или дополнить первичный ключ дочерней сущности.

3. Если мощность связи 1:1 и класс принадлежности обеих сущностей необязательный, то необходимо три отношения – по одному на каждую сущность и отношение на связь, в которое добавляются ключи исходных сущностей.

*Дополнение:* Возможно использование двух отношений по одному на каждую сущность, но при этом важно выделить из двух сущностей родительскую (наиболее важную с точки зрения логики хранения данных) и дочернюю. Тогда в дочернюю сущность перейдет ключ родительской сущности, не дополняя первичный ключ дочерней сущности, с признаком необязательности заполнения.

4. Если мощность бинарной связи 1:n и класс принадлежности n-связной сущности обязательный, необходимо два отношения по одному на каждую сущность, при этом ключ односвязной сущности добавляется в качестве дополнительного атрибута в n-связную сущность.

*Дополнение:* переходящий ключ может дополнить ключ n-связной сущности, либо не входить в первичный ключ дочерней сущности, а являться дополнительным атрибутом отношения, соответствующего дочерней сущности (но обязательным для заполнения). Это действие зависит от того, может ли дочерняя сущность однозначно определяться своим ключом.

5. Если мощность бинарной связи 1:n и класс принадлежности n-связной сущности не обязательный, необходимо три отношения по одному на каждую сущность и отношение на связь, в которое переходят ключи обеих сущностей.

*Дополнение:* возможно использование двух отношений, но при этом ключ односвязной сущности переходит в отношение, соответствующее дочерней сущности с признаком необязательности заполнения.

6. Если мощность связи n:m, то в независимости от класса принадлежности сущностей необходимо три отношения по одному на каждую сущность и отношение на связь, в которое переходят ключи обеих сущностей.

*Дополнение:* отношение на связь при необходимости может иметь дополнительные атрибуты, уточняющие связь. Кроме этого ключ этого отношения может состоять:

1) из ключей исходных отношений и плюс дополнительные атрибуты;

2) из одного из ключей исходных сущностей и плюс дополнительные атрибуты;

3) из собственных атрибутов, а переходящие ключи исходных отношений не входят в первичный ключ отношения на связь.

Эти действия определяются особенностями предметной области.

7. В случае n- сторонней связи между сущностями необходимо n+1 отношение по одному на каждую сущность и отношение на связь, в которое переходят ключи всех сущностей.

**Рекомендации к определению состава ключей при построении схем отношений по правилам Джексона:**

1. При определении схем отношений по правилам Джексона необходимо для каждого отношения определить состав ключей, начиная с множества возможных (потенциальных) ключей. Ключ – это множество атрибутов, однозначно определяющих каждый экземпляр сущности, не повторяющийся, уникальный.

2. Из множества возможных ключей выбирается один ключ, который называется первичным ключом (на схеме отношения подчеркивается сплошной линией). Первичным обычно становится ключ, который с точки зрения предметной области более важен для определения

сущности или более эффективен с точки зрения хранения. Например, возможными ключами для сущности Студент могут быть – «номер студенческого», «Номер группы, ФИО, дата рождения», «номер и серия паспорта». В качестве первичного ключа выбирается номер студенческого, так как он лучше характеризует именно студента, а не личность, и более экономичен по хранению по сравнению с составным ключом «Номер группы, ФИО, дата рождения».

3. Далее определяются все альтернативные ключи. Альтернативный ключ – это возможный (или иначе потенциальный ключ), не ставший первичным (на схеме отношений подчеркиваются пунктирной или волнистой линией).

4. В качестве первичного ключа может быть создан суррогатный ключ (номер записи). Суррогатному ключу дается имя – код и далее имя сущности (например, код студента). Суррогатный первичный ключ создается, если возможные (потенциальные) ключи являются составными (состоят из нескольких атрибутов) и от отношения с таким ключом имеются связи в базе данных или, если только вся совокупность атрибутов отношения не повторяется, а также, если первичный ключ является нечисловым типом и занимает большой объём памяти. Тем не менее, даже при создании суррогатного ключа необходимо **обязательно** определить все альтернативные ключи отношения.

С учетом применения перечисленных правил Джексона, схемы предварительных отношений для примера базы данных «Испытания изделий» имеют следующий вид.

Подразделение (Номер подразделения, название подразделения)

Сотрудник (Табельный номер, ФИО, номер подразделения)

Штатный сотрудник (Табельный номер, оклад)

Совместитель (Табельный номер, процент ставки)

Паспортные данные (Табельный номер, серия, номер паспорта, кем выдан, год рождения, место выдачи)

Вид изделия (Код вида изделия, название вида изделия)

Вид испытания (Код вида испытания, название вида испытания)

Комиссия (индекс комиссии, табельный номер председателя)

Сотрудник в комиссии (табельный номер, индекс комиссии)

Испытание (Номер изделия, код вида испытания, дата плановая, индекс комиссии, дата фактическая, оценка, код вида изделия)

Здесь выделены все атрибуты, которые могут использоваться в качестве ключевых атрибутов (возможные или потенциальные ключи). При этом сплошной линией подчеркнуты первичные ключи, а волнистой альтернативные ключи (потенциальные ключи, не ставшие первичными).

### 3.3. Анализ схем отношений базы данных на нормальные формы. Нормализация отношений

Функциональные зависимости отображают смысловые, семантические связи между атрибутами отношения и в теории проектирования реляционных баз данных играют важную роль. Существует множество видов функциональных зависимостей между атрибутами отношений [1].

*Атрибут B функционально зависит от атрибута A (*$A \rightarrow B$*)*, если для каждого значения атрибута *A* существует ровно одно, связанное с ним значение атрибута *B*. *A* и *B* могут быть составными атрибутами. Атрибут называется составным, если он включает множество атрибутов отношения.

Если $A = A_1A_2...A_к$ и $A \rightarrow B$, то есть B функционально зависит от всей группы атрибутов A, то говорят, что *B функционально полно зависит от A.*

Если $A = A_1A_2...A_к$ и $A_i...A_j \rightarrow$ B и $i, j < k$, то *между A и B имеется частичная функциональная зависимость.*

Если $A \leftrightarrow B$, то *A и B взаимозависимые атрибуты.*

Взаимозависимость также может быть полной и частичной.

Если для атрибутов *A, B и C* выполняется условие $A \rightarrow$ B и $B \rightarrow C$, но обратной связи нет, то говорят, что *C транзитно зависит от A*.

*Атрибут B многозначно зависит от A,* $A \rightarrow\rightarrow B$, если каждому значению *A* соответствует множество значений *B*, никак не связанных с другими атрибутами отношения.

С точки зрения понятия функциональной зависимости между атрибутами к*лючом отношения* называется непустое подмножество атрибутов, от которого функционально полно зависит все множество атрибутов отношения. Аналогом введенного понятия является понятие потенциального ключа. В отношении может быть множество потенциальных ключей, среди которых выбирается один первичный. В последствие, называя ключ, будем понимать потенциальный ключ отношения.

*Нормализация* – это процесс разбиения или декомпозиции исходного отношения на несколько отношений с целью устранения нежелательных функциональных зависимостей, приводящих к возникновению избыточности хранения информации и аномалиям добавления, удаления, обновления.

Аппарат нормализации отношений разработан Коддом. В нем определены различные *нормальные формы отношений*. Каждая нормальная форма ограничивает типы допустимых функциональных зависимостей отношений. Первоначально Кодд выделил три нормальные формы: 1, 2, 3НФ, а сегодня определены и другие нормальные формы отношений, ограничивающие некоторые функциональные зависимости между атрибутами отношения.

Следует отметить, что нормализация или иначе декомпозиция должна быть обратимой, то есть выполняться без потерь информации. Вопрос о том, происходит ли утрата информации при декомпозиции, тесно связан с концепцией функциональной зависимости [9]. Зависимость между декомпозицией без потерь ФЗ подтверждается теоремой Хеза.

*Теорема Хеза*: Пусть R(A,B,C) является отношением с атрибутами A,B,C. Если оно удовлетворяет зависимости A → B, то R равно соединению его проекций (A,B) и (A,C).

*1НФ* – Отношение находится *в первой нормальной форме*, если значения всех атрибутов атомарные, то есть значение атрибута не является множеством, диапазоном или повторяющейся группой.

*2НФ* – Отношение находится *во второй нормальной форме*, если оно находится в первой нормальной форме, и отсутствуют частичные функциональные зависимости атрибутов не входящих в ключ от ключа (зависимости атрибутов не входящих в ключ от части ключа). Иначе можно сказать, если каждый атрибут не входящий в ключ, функционально полно зависит от

ключа. Анализируя определение второй нормальной формы, можно сделать вывод, что отношение находится во второй нормальной форме, если в нем отсутствуют составные ключи.

*Правило нормализации при приведении отношения ко второй нормальной форме:* Выполняется дважды операция проекции над исходным отношением. В результате первой операции получается отношение, в которое входит часть ключа и *все* атрибуты, от нее зависящие. В результате второй операции проекции получается отношение, в которое входит составной ключ и все оставшиеся атрибуты исходного отношения, не вошедшие в отношение, полученное после применения первой операции проекции.

*3НФ* – Отношение находится *в третьей нормальной форме*, если оно находится во второй нормальной форме, и отсутствуют транзитивные зависимости атрибутов, не входящих в ключ от ключа. Иными словами, если выполняется совокупность условий:

$$A \rightarrow B;\ B \rightarrow C;\ C \not\rightarrow A;\ B \not\rightarrow A;\ C \not\rightarrow B,$$

тогда в отношении существует транзитивная зависимость атрибутов не входящих в ключ от ключа A и отношение не находится в третьей нормальной форме. Если хотя бы одно из условий не выполняется, то такой зависимости между атрибутами A, B, C нет. Причем атрибуты A, B, C могут быть составными.

*Правило нормализации отношения для приведения к третьей нормальной форме*: Дважды выполняется операция проекции над исходным отношением. В результате выполнения первой операции проекции в отношение включается атрибуты B и C, не являющиеся ключами. В результате выполнения второй операции проекции над исходным отношением, в отношение включается ключевой атрибут A, B и все оставшиеся атрибуты исходного отношения, не попавшие в отношение, полученное в результате выполнения первой операции проекции. Следует отметить, что процесс нормализации не является обязательным, так как в случае требований к базе данных высокой скорости выполнения запросов, игнорируя эффективность хранения данных, приводит к денормализации отношений. Поэтому вопросы, связанные с необходимостью нормализации отношений решает проектировщик базы данных с учетом требований к базе данных

### 3.4. Построение логической модели базы данных, используя CASE-средства по методологиям IDEF1X или IE

Методология IDEF1X была разработана для армии США и широко используется в государственных учреждениях США, финансовых и промышленных корпорациях при разработке реляционных баз данных. Эта методология входит в общую методологию проектирования автоматизированных систем IDEF, в основе которой лежит структурный системный анализ. В настоящее время методология IDEF1X, реализующая двухуровневую архитектуру представления данных, используется для построения логических моделей реляционных баз данных. В нотации этой методологии используется диаграмма модели сущность-связь. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области.

В IDEF1X различают зависимые и независимые сущности. Особенности сущности определяется её связью с другими сущностями. *Независимая сущность* может находиться на модели вне связи с другими сущностям и не требует дополнительных атрибутов, относящихся к

другим сущностям. Независимая сущность обозначается на модели графическим примитивом в виде прямоугольника, внутрь которого, включаются имена атрибутов и специально выделена область первичных ключевых атрибутов [9].

Зависимая сущность обязательно связана с одной или несколькими сущностями. Выделяют несколько видов зависимых сущностей: характеристическую зависимую сущность, ассоциативную и категориальную зависимую сущности. *Характеристическая зависимая сущность* – это зависимая дочерняя сущность, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской сущности. *Ассоциативная зависимая сущность* – сущность, связанная с несколькими родительскими сущностями. Такая сущность содержит информацию о связях сущностей. *Категориальная зависимая сущность* – дочерняя сущность в иерархии наследования.

В нотации IDEF1X различают связи идентифицирующие, неидентифицирующие и категориальные. *Идентифицирующая связь* устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Экземпляр зависимой сущности определяется только через отношение к родительской сущности. Обозначается идентифицирующая связь сплошной линией с черной точкой на конце. *Неидентифицирующая связь* устанавливается между двумя независимыми сущностями. Обозначается неидентифицирующая связь пунктирной линией с черной точкой на конце. *Категориальная связь* используется для реализации механизма наследования между сущностями. Для обозначения категориальной связи имеется специальный значок, называемый дискриминатором.

При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов, которая уже учитывает реализацию модели на реляционной структуре данных. В дочерней сущности новые атрибуты помечаются как внешний ключ – (FK). В последствие, при физической генерации схемы базы данных, внешний ключ, мигрирующий в область первичных ключей, получит признак NO NULL. Для неидентифицирующей связи возможны два варианта включения внешнего ключа в дочернюю сущность - с признаком NULLS или NO NULL.

Важным понятием в IDEF1X является понятие ключа и различные типы ключей. Каждый экземпляр сущности должен быть уникален и отличаться от других атрибутов.

*Первичный ключ (Primary key)* – это атрибут или группа атрибутов, однозначно идентифицирующих экземпляр сущности. Атрибуты первичного ключа на диаграмме не требуют специального обозначения – это те атрибуты, которые находятся в списке атрибутов выше горизонтальной линии. Выбор первичного ключа может оказаться непростой задачей для проектировщика, решение которой может повлиять на эффективность будущей информационной системы. В одной сущности могут оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются *потенциальными ключами (candidate key).*

Ключи могут быть *сложными (составными)*, то есть содержащими несколько атрибутов. Сложные первичные ключи не требуют специального обозначения – это список атрибутов выше горизонтальной линии.

*Альтернативный ключ (Alternate Key)* – это потенциальный ключ, не ставший первичным. При работе информационной системы часто бывает необходимо обеспечить доступ к нескольким экземплярам сущности, объединенным каким-либо одним признаком. Для повышения производительности в этом случае используются неуникальные индексы. Атрибуты, участвующие в неуникальных индексах, по которым будет осуществляться выборка данных, называются *инверсионными входами (Inversion Entry)***.** *Инверсионные входы* – это атрибут или группа атрибутов, которые не определяют экземпляр сущности уникальным образом, но часто используются для обращения к экземплярам сущности.

*Внешние ключи (Foreign Key)* создают автоматически, когда связь соединяет сущности. Связь образует ссылку на атрибуты первичного ключа в дочерней сущности и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибут внешнего ключа обозначается символом (FK) после своего имени.

Кроме нотации IDEF1X распространенной в настоящее время является и нотация IE. На рисунке 3.1 приведено соответствие графических примитивов связей в обеих нотациях.

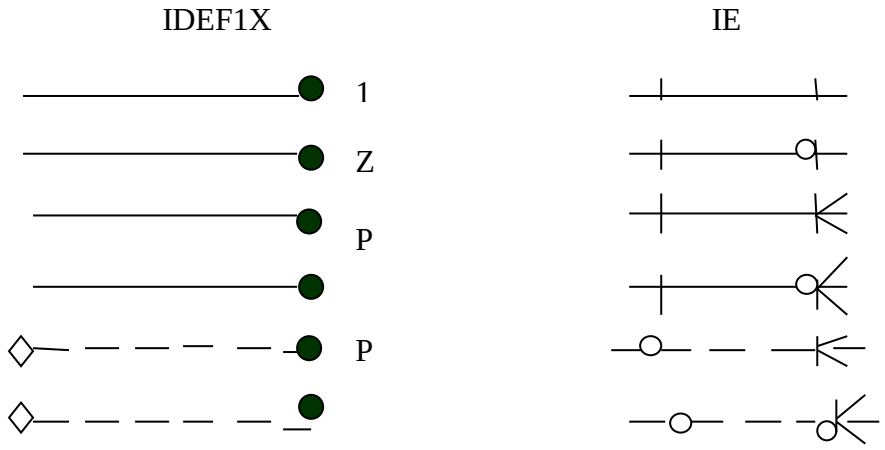



Рисунок 3.1 – Соответствие графических примитивов связей нотаций IDEF1X и IE для моделей «сущность-связь»

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности систем баз данных. Опыт проектирования таких систем показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Для автоматизации этой технологии в настоящее время используются программно-технологические средства специального класса – CASE-средства, реализующие CASE-технологию создания и сопровождения информационных систем. Под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения информационных систем, включая анализ и формулировку требований, проектирование приложений и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы.

Примером CASE-средства проектирования баз данных является программный продукт Erwin DataModeler, который позволяет проектировать схему реляционной базы данных, используя методологии IDEF1X и IE с последующей генерацией на физическом уровне средствами одной из современных СУБД (Oracle, Informix, DB/2, Ingres, Progress, SQL Server, SQLBase, Sybase и др.).

Соответствие концептуальной модели и схем отношений после применения правил Джексона и нормализации отношений представлено на рисунках 3.2, 3.3, 3.4, 3.5. Здесь использованы следующие обозначения: S – сущности, К – ключевые атрибуты, О – отношения, А – атрибуты, Р – родительское отношение, Д – дочернее отношение.

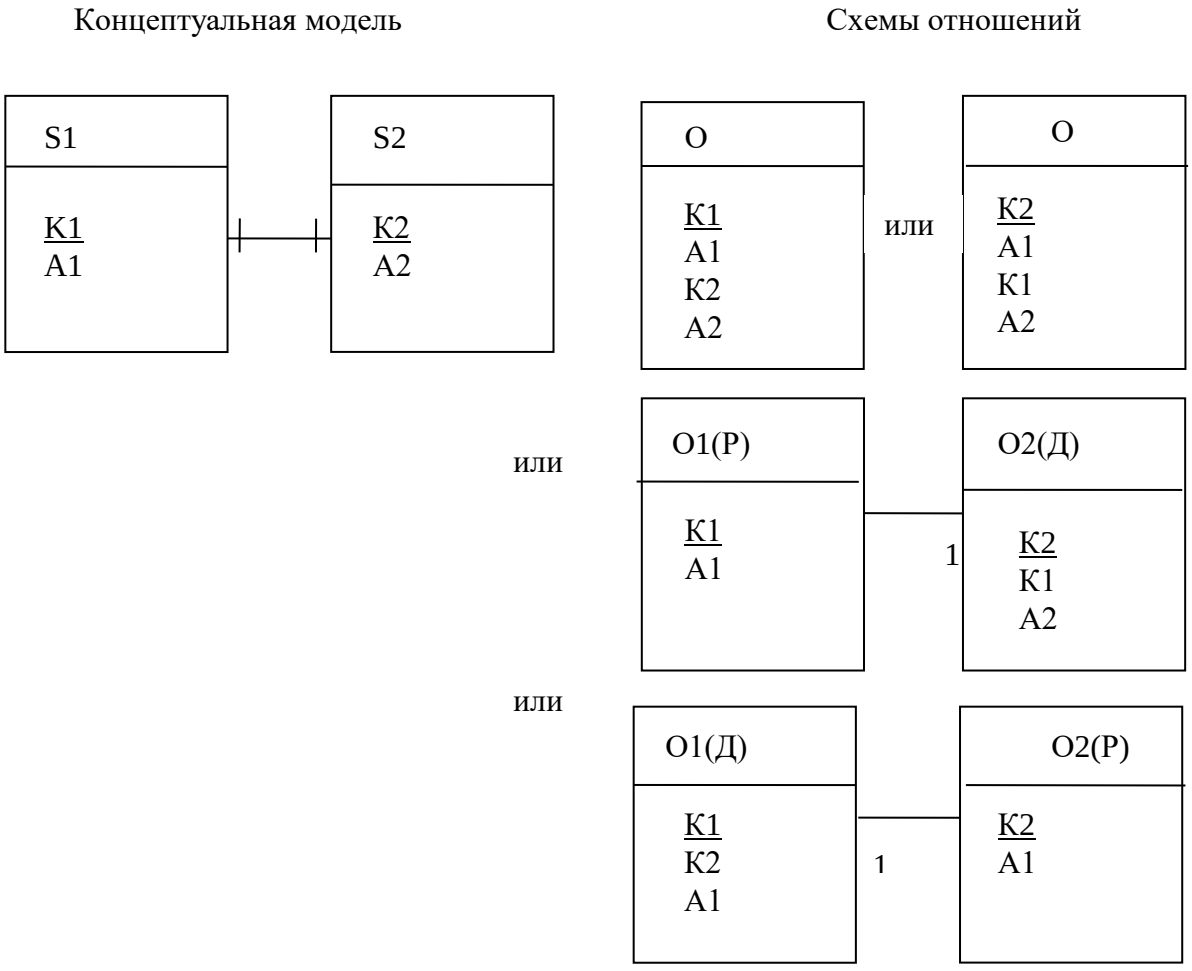



Рисунок 3.2 – Соответствие концептуальной модели и схем отношений после применения правил Джексона и нормализации отношений для связи 1:1 с обязательными классами принадлежности

**Рекомендации по переходу от схем отношений к нотации IDEF1X**

1. Сущности-отношения соответствуют отношениям на схеме отношений базы данных. Создаются сущности-отношения с указанием соответствующих имен.

2. Для каждой сущности-отношения указывается перечень атрибутов, кроме тех, что переходят по правилам Джексона (являются внешними ключами).

3. Делается анализ того, куда выполнен переход атрибутов – в область основных атрибутов, или в первичный ключ. Если в первичный ключ дочерней сущности, то проводится идентифицирующая связь. Если переход выполнен в область основных атрибутов, то проводится неидентифицирующая связь.

4. Если имеется несколько связей 1:1 одной сущности с несколькими дочерними сущностями, при этом класс принадлежности дочерних сущностей необязательный, то проводится категориальная связь с дискриптором, который имеет свойство полной или неполной категории.

Методология **IE** – Information Engineering (информационное проектирование). Основоположники методологии Клайв Финклештейн и Джеймс Мартин. Нотация использовалась сначала преимущественно в промышленности. В настоящее время широко применяется в различных областях преимущественно в Европе. Нотация IE во многом похожа на IDEF1X, которая используется в США.

Нотация IE выделяет собственные типы иерархии категории (наследования) в зависимости от количество вхождений экземпляра супертипа в подтипы: эксклюзивную и неэксклюзивную. При этом как *эксклюзивная иерархия категорий*, так и *неэксклюзивна*я в IE считается полной.

*Эксклюзивная иерархия категорий предполагает, что экземпляр су*пертипа входит только в один экземпляра подтипа. Например, банковский счет может быть либо накопительным, либо до востребования (т.е. возможен лишь один вариант).

*Неэксклюзивная иерархия категорий предполагает, что э*кземпляр супертипа может входить более чем в один экземпляр подтипа. Например, банковский счет может быть одновременно накопительным и до востребования. На рисунке 3.6 приведено описание типов иерархий в разных нотациях.

На рисунке 3.7 приведено соответствие между условными обозначениями нотаций IDEF1X и IE для идентифицирующих связей.

На рисунке 3.8 приведено соответствие между условными обозначениями нотаций IDEF1X и IE для неидентифицирующих связей.

На рисунке 3.9 приведен пример описания логической схемы базы данных «Испытания изделий» в нотации IDEF1X средствами ERWin DataModeler.

На рисунке 3.10 приведено описание логической схемы базы данных «Испытания изделий» в нотации IE средствами ERWin DataModeler.

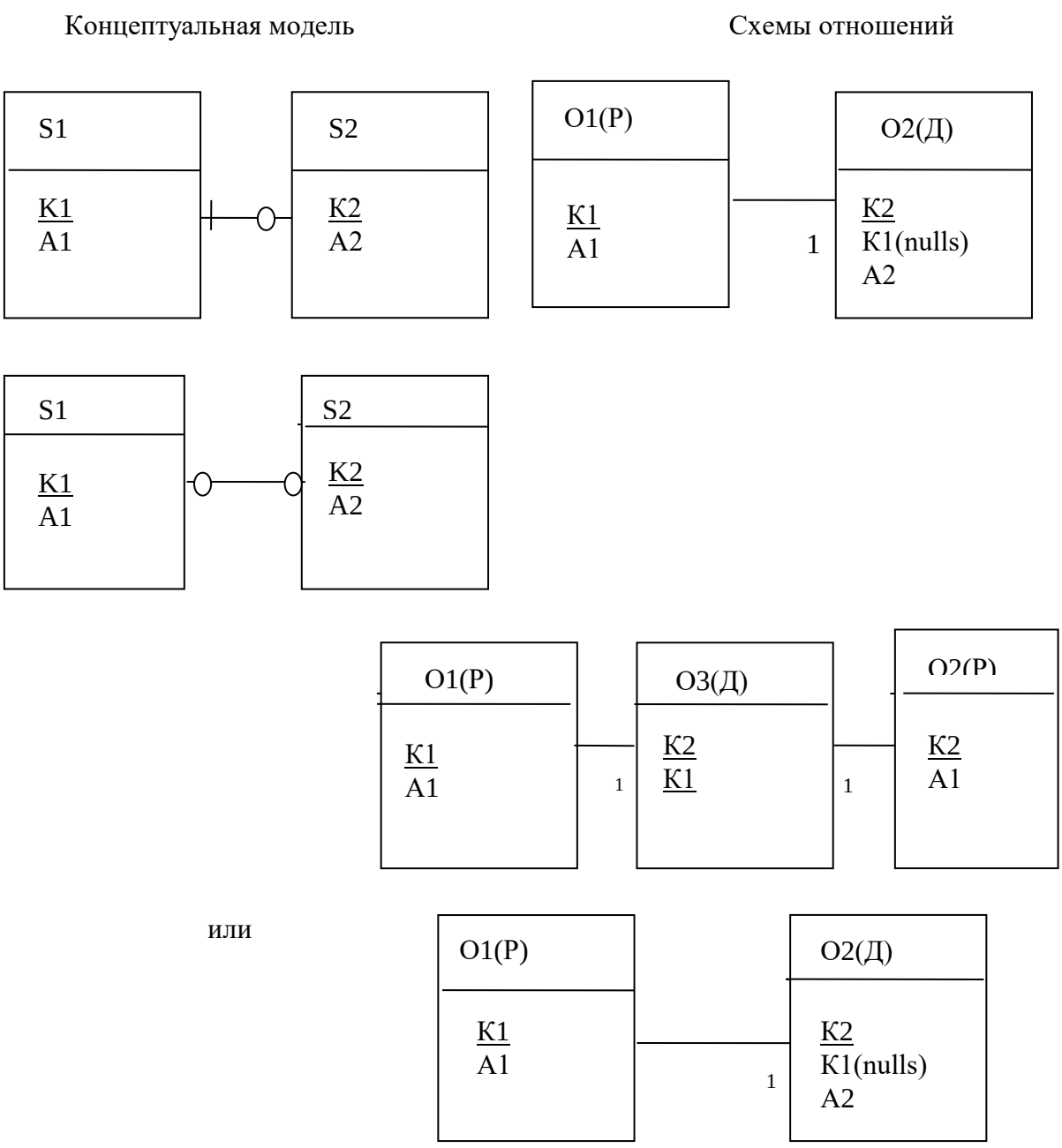



Рисунок 3.3 – Соответствие концептуальной модели и схем отношений после применения правил Джексона и нормализации отношений для связи 1:1 с необязательными классами принадлежности сущностей

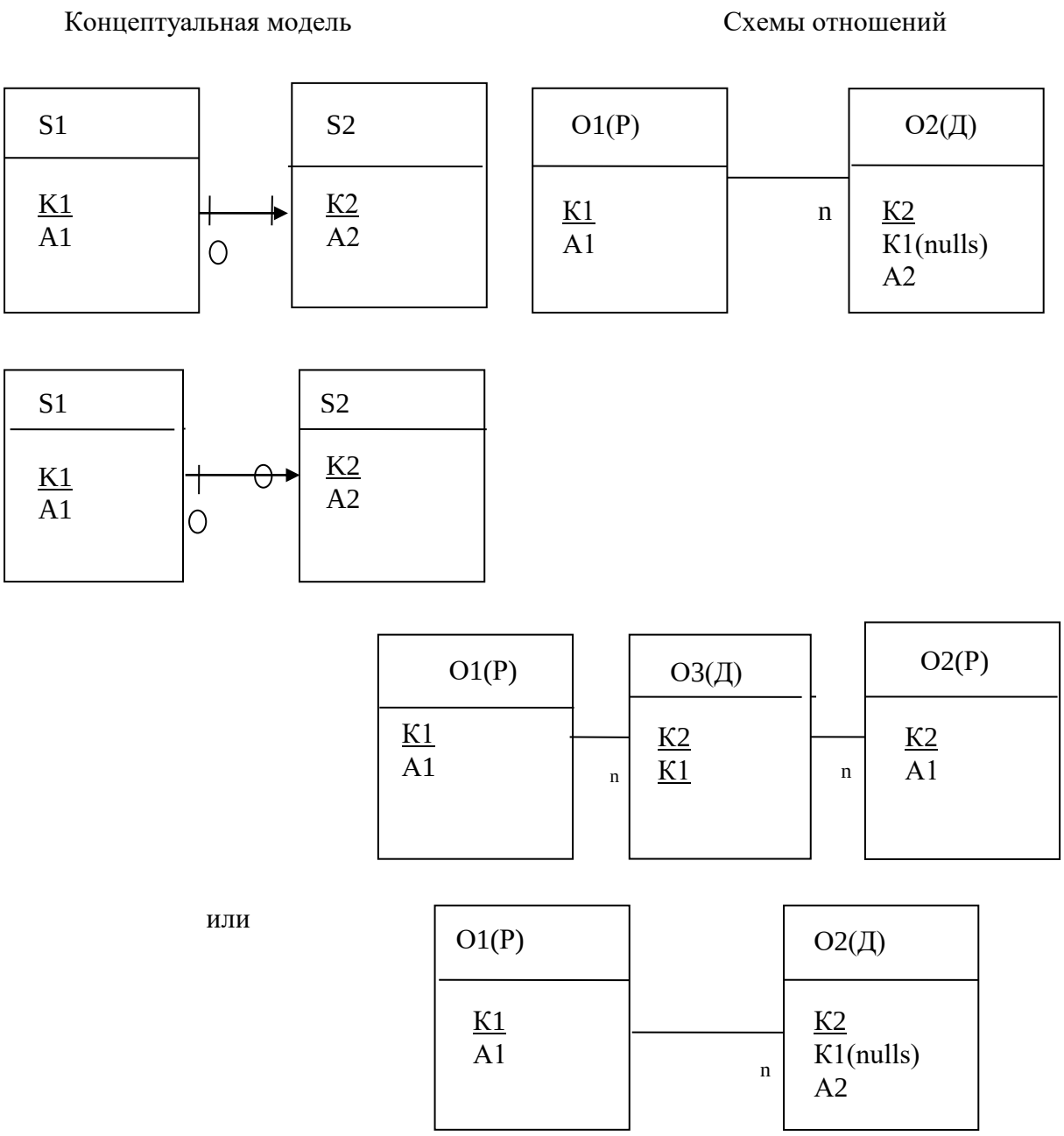



Рисунок 3.4 – Соответствие концептуальной модели и схем отношений после применения правил Джексона и нормализации отношений для связи 1:N с обязательными и необязательными классами принадлежности N – связной сущности

Концептуальная модель Схемы отношений

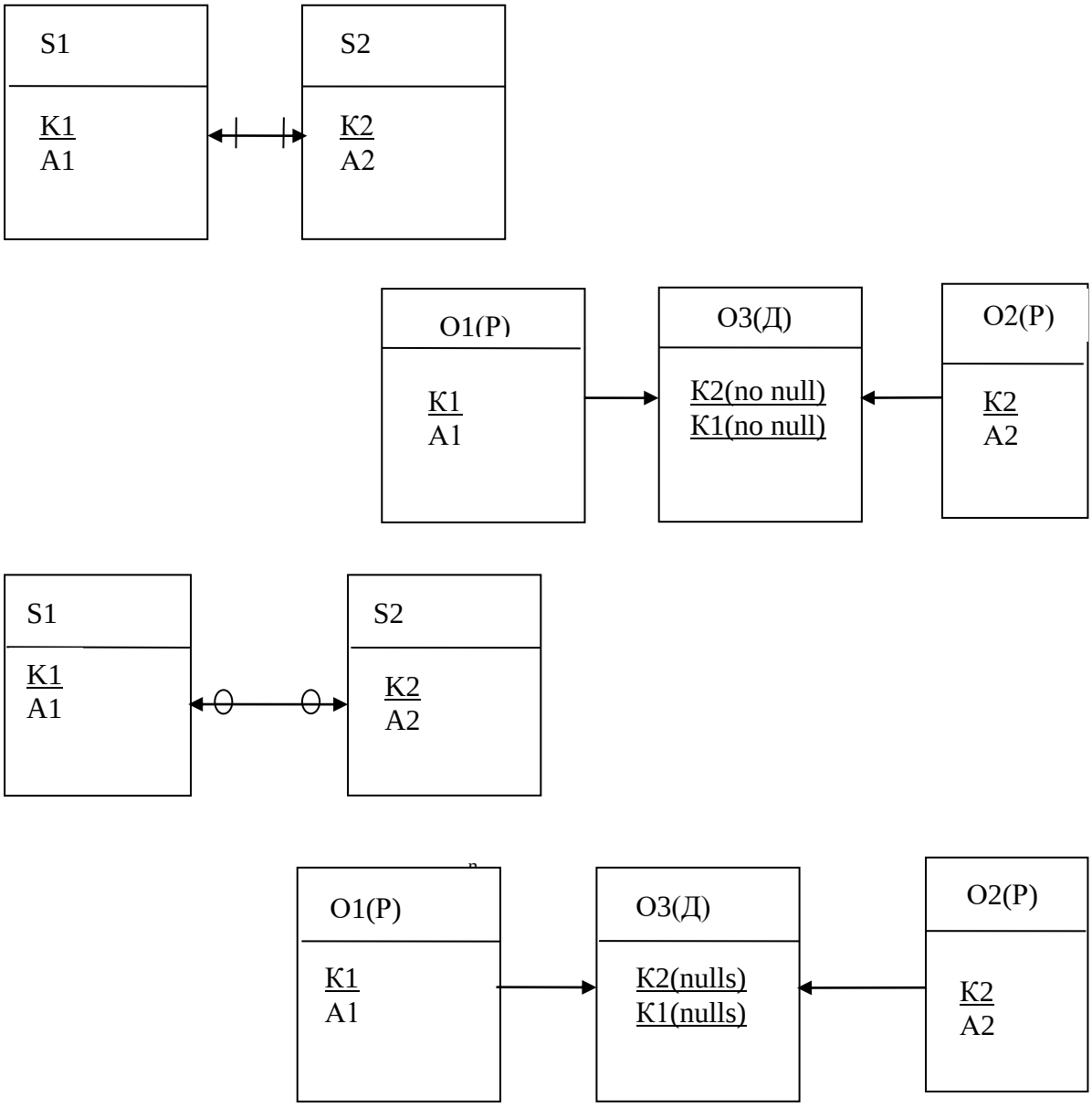



Рисунок 3.5 – Соответствие концептуальной модели и схем отношений после применения правил Джексона и нормализации отношений для связи M:N с обязательными и необязательными классами принадлежности сущностей

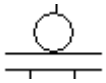
| Нотация | Тип иерархии категорий | Графическое обозначение | Описание |
|---|---|---|---|
| **IDEF1X** | Полная |  | Отображены все варианты сущностей-потомков |
| | Неполная |  | Отображены не все варианты сущностей-потомков |
| **IE** | Эксклюзивная |  | Одновременно существует лишь одна из сущностей-потомков |
| | Неэксклюзивная |  | Одновременно могут существовать все сущности-потомки |

Рисунок 3.6 – Типы иерархий категорий и их отображение в нотациях IDEF1X, IE

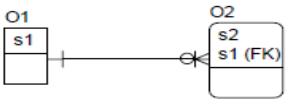
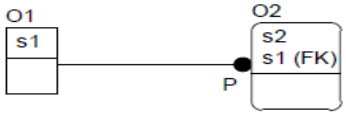
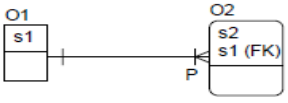
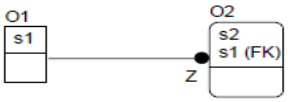
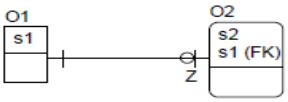
| Вид связи | Изображение в IDEF1X | Изображение в IE |
|---|---|---|
| Идентифицирующая связь Zero, One or Many |  |  |
| Идентифицирующая связь One or Many |  |  |
| Идентифицирующая связь Zero, One |  |  |

Рисунок 3.7 – Соответствие между условными обозначениями нотаций IDEF1X и IE для идентифицирующих связей

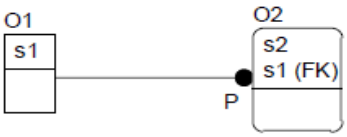
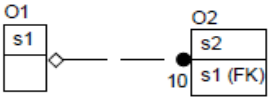
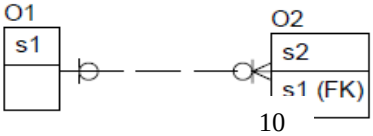
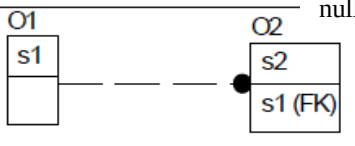
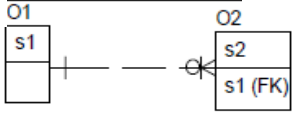
| Вид связи | Изображение в IDEF1X | Изображение в IE |
|---|---|---|
| Идентифицирующая связь Exactly |  |  |
| Неидентифицирующая связь Zero, One or Many, Nulls allowed |  |  |
| Неидентифицирующая связь Zero, One or Many, No nulls |  |  |

Рисунок 3.8 – Соответствие между условными обозначениями нотаций IDEF1X и IE для неидентифицирующих связей

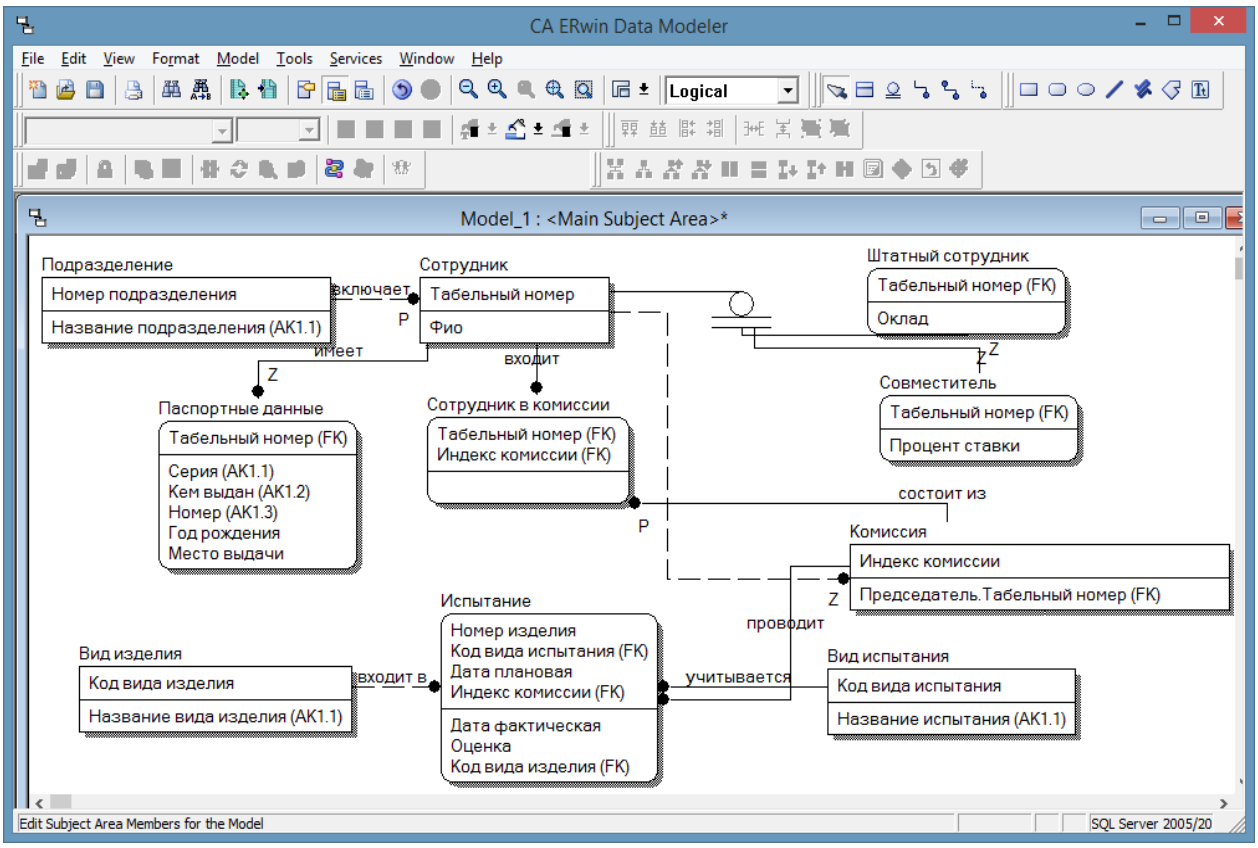



Рисунок 3.9 – Описание базы данных «Испытания изделий» в нотации IDEF1X

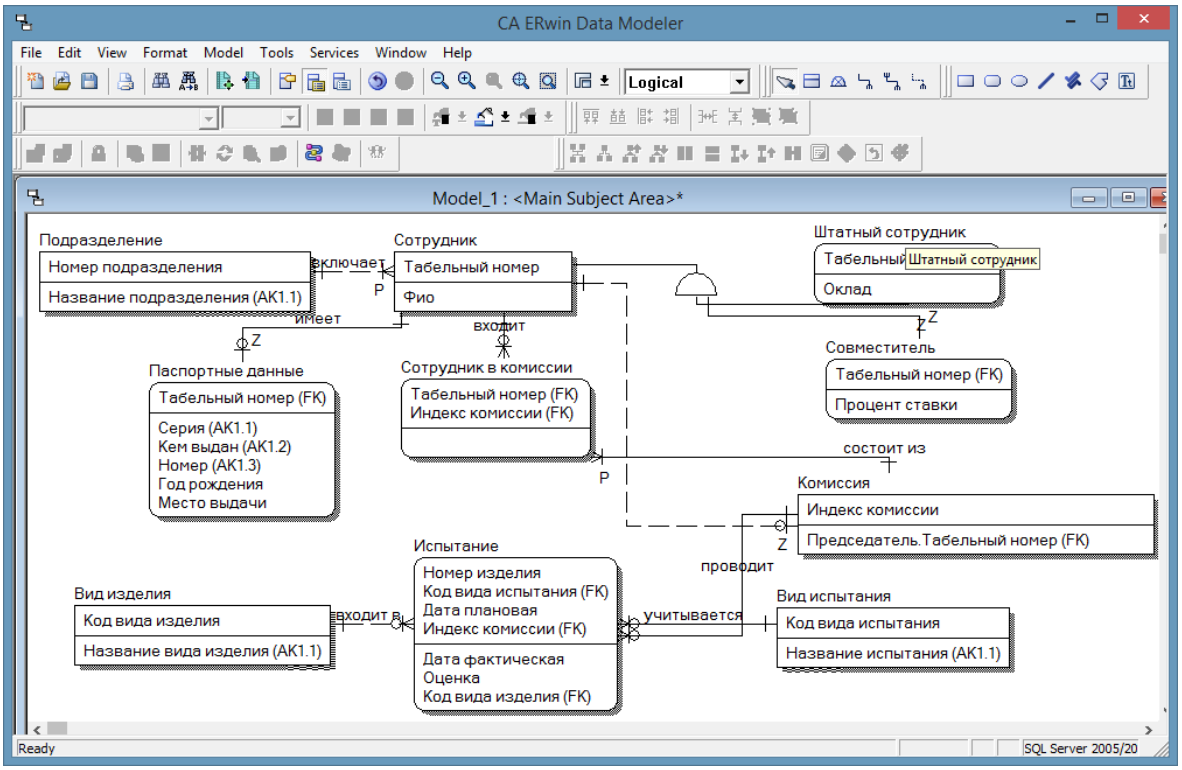



Рисунок 3.10 – Описание схемы базы данных «Испытания изделий» в нотации IE

# ГЛАВА 4. ЭТАПЫ ВЫБОРА СРЕДСТВА РЕАЛИЗАЦИИ БАЗЫ ДАННЫХ И ФИЗИЧЕСКОГО ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ

## 4.1. Рекомендации по выбору средства реализации реляционной базы данных

1. Анализ места хранения данных и количества пользователей, реализующих одновременно доступ к базе данных. Рекомендуется определить распределенная или централизованная база данных. Для локальных (централизированных) СУБД реляционного типа используются Access, Paradox, FoxPro и другие. Для реализации распределенных СУБД реляционного типа используются MS SQL Server, Oracle, SyBase, Postgree и другие.

2. Анализ навыков разработчика системы баз данных (например, для непрофессионального разработчика локальных баз данных можно использовать Access, для профессиональных FoxPro).

3. Анализ знаний разработчика языка манипулирования данными. Например, знания SQL (FoxPro) или QBE(Access).

4. Анализ объема хранимых данных и возможности СУБД при прочих равных условиях. Например, Oracle позволяет работать с базами данных большого объема по количеству реализуемых отношений и объему записей, MS SQL Server базу данных среднего объема.

5. Анализ возможностей диалекта языка SQL, требований к web – хранению.

## 4.2. Средства реализации нереляционных баз данных

База данных NoSQL – это база данных, имеющая структуру хранения данных, отличную от реляционной структуры. Она позволяет избежать соединений и легко масштабируется. Основная цель использования базы данных NoSQL – реализация хранилищ данных, динамических баз данных, баз данных большого объема. Базы данных NoSQL с точки зрения физической организации файлов данных можно разделить на базы данных «ключ-значение», графовые базы данных, колоночные базы данных, документально-ориентированные базы данных.

При физической организации базы данных «ключ-значение» используют пары ключ-значение для хранения и извлечения данных. Имя атрибута хранится в «ключе». Значения, соответствующие этому атрибуту, будут храниться в «значении». Ключ может *быть только строкой*, тогда как значение может хранить *строку, JSON, XML, большой двоичный объект.* Это даёт возможность обрабатывать большие объемы данных. База данных «ключ-значение» использует ассоциативный массив в качестве структуры данных, где каждый ключ связан с одним и только одним значением в коллекции. Для того, чтобы хранение записей осуществлялась в отсортированном виде, используется ключ сортировки. Ключ сортировки используется для указания зависимости от других ключей. *Хранилища ключей и значений не имеют языка запросов*. Они обеспечивают способ хранения, извлечения и обновления данных с помощью простых команд получения, размещения и удаления; путь для извлечения данных – это прямой запрос к объекту в памяти или на диске.

Базы данных графовой структуры создают и хранят данные и связи между ними. Каждый элемент с данными хранится в узле, и этот узел связан с другими данными-элементами.

Базы данных в виде графов позволяют определять связи между элементами данных и прямо или косвенно связывать одну часть с различными частями. Графовая структура данных в реляционной базе данных это совокупность взаимосвязанных графовых таблиц. Вершины и ребра графа представляются в виде новых типов таблиц: NODE и EDGE. В отличие от других моделей данных, в графовых базах данных не требуется вычислять связи с помощью внешних ключей или какими-то другими способами. Можно создавать сложные модели данных, используя только абстракции вершин и ребер.

Базы данных, ориентированные на столбцы, используют концепцию документа BigTable от Google**.** В базах данных NoSQL, ориентированных на столбцы, данные хранятся в ячейках, сгруппированных в столбцы данных, а не в виде строк данных. Столбцы логически сгруппированы в семейства столбцов. Семейства столбцов могут содержать практически неограниченное количество столбцов, которые можно создавать во время выполнения или при определении схемы. Чтение и запись выполняются с использованием столбцов, а не строк. Семейства столбцов – это группы похожих данных, доступ к которым обычно осуществляется вместе. Основными преимуществами хранения данных в столбцах по сравнению с реляционными СУБД являются быстрый поиск/доступ и агрегация данных. Реляционные базы данных хранят одну строку как непрерывную запись на диске. Разные строки хранятся в разных местах на диске, в то время как колоночные базы данных хранят все ячейки, соответствующие столбцу, как непрерывную запись на диске, что ускоряет поиск/доступ.

База данных, ориентированная на документы (документально – ориентированные базы данных), использует понятия коллекции и документов. В базах данных документ реализует формат JSON и является самостоятельным и свободным от схемы. В базах данных документов вся информация для данного документа или объекта хранится в одном экземпляре – нет необходимости делать объектно-реляционное сопоставление при загрузке данных в базу данных или при извлечении вещей. Коллекция – это группа документов. Документ – это набор пар “ключ – значение”. Документ имеет динамическую схему. Это означает, что документ в одной и той же коллекции не обязан иметь одинаковый набор полей или структуру, а общие поля в коллекции могут иметь различные типы данных.

Для реализации базы данных в формате «ключ-значение» используются СУБД – Redis, MemcacheDB, Amazon DynamoDB. Redis реализует хранилище ключей и значений NoSQL в памяти с открытым исходным кодом, которое используется в основном в качестве кэша приложений или базы данных быстрого реагирования. Поскольку данные хранятся в памяти, а не на диске или твердотельном накопителе – SSD, обеспечивает высокую скорость, надежность и производительность. Memcached – это система кэширования распределенной памяти с открытым исходным кодом. Данная СУБД не предусматривает постоянное хранение данных, а также хранение данных в больших объёмах. Amazon DynamoDB – СУБД класса NoSQL в формате «ключ – значение». Данная СУБД может масштабировать ресурсы таблицы на большое количество серверов в различных зонах доступности для удовлетворения потребностей в хранении. Хранение данных осуществляется в документе JSON.

Для реализации документально-ориентированных баз данных используются СУБД MongoDB и CouchDB. CouchDB имеет открытый исходный код, разработанный Apache, совместим с сетью. Для хранения данных CouchDB использует JSON, JavaScript в качестве языка за-

просов для преобразования документов, используя MapReduce и HTTP для API. CouchDB занимает много; не поддерживает транзакции; репликация больших баз данных может завершиться ошибкой. MongoDB хранит данные в JSON-подобных документах с динамической схемой. Это означает, что можно хранить свои записи, не беспокоясь о структуре данных, например о количестве полей, или типах полей для хранения значений. Документы MongoDB аналогичны объектам JSON. MongoDB имеет полную облачную платформу данных приложений; гибкие схемы документов; мощные запросы и аналитику.

Для создания баз данных колоночного типа используются СУБД Apache Cassandra и BigTable. Bigtable – это распределенная система хранения для управления структурированными данными, предназначенная для масштабирования до очень больших размеров: петабайт данных на тысячах обычных серверов. Основным недостатком использования данной системы является высокая стоимость. Apache Cassandra – это СУБД, реализующая колоночную базу данных с открытым исходным кодом. Данные представлены в виде файла в формате JSON. Модель данных Cassandra, основана на комбинации столбцовых семейств пространства ключей.

Для реализации графовых баз данных используются такие СУБД, как Neo4j, а также СУБД реляционного типа с расширением возможностей работы с графовыми таблицами. Для реализации таких таблиц в Ms Sql Server используются таблицы двух видов – таблицы узлов и таблицы рёбер.

## 4.3. Виды ограничений целостности данных и способы их реализации в реляционных базах данных

В процессе проектирования баз данных необходимо реализовать целостность, восстанавливаемость, безопасность и эффективность хранения данных.

***Целостность данных*** – свойство базы данных создавать ограничения на значения и структуру данных и сохранять эти ограничения при любых операциях ведения данных.

***Виды ограничений на значения данных***:

1) Данные не превышают или менее *указанного значения*.
2) Данные должны находиться в заданном *диапазоне*.
3) Данные могут принимать значения из некоторого *множества*
4) Если поле является *первичным или альтернативным ключом*, то значение этого поля должно быть уникально.
5) Множество значений одного атрибута вычисляется через операции над другими (*вычисляемое поле*).

***Ограничения на структуру данных***:

1) Контроль *типов полей данных*.
2) *Ограничение на число полей в записи и порядок их следования*.
3) *Ссылочная целостность,* что означает согласованное выполнение операций во взаимосвязанных отношениях.

Виды ограничений ссылочной целостности – Restrict, Cascade. *Ссылочная целостность Restrict* означает *запрет* на выполнение операций удаления и изменения записей в родительской таблице, если в дочерней таблице имеются записи с тем же значением внешнего ключа.

*Ссылочная целостность Cascade* означает одновременное выполнение операций по удалению и изменению записей, как в родительской, так и в дочерней таблице, если поля связи в обеих таблицах имеет одинаковое значение. Все операции ссылочной целостности выполняются *над родительскими таблицами*.

Проектировщик базы данных при описании ограничений целостности должен сформулировать смысл ограничений, а также описать когда, какими средствами и при каких условиях будет выполняться проверка каждого ограничения.

Ниже приведены виды ограничений целостности и способы их реализации в реляционных базах данных.

**1. Ограничения на значения данных – на уровне поля таблицы**:

1) ***Ограничение на домен (***домен – множество однотипных значений), используя ***стандартный*** выбранный ***тип*** поля, имеющихся в перечне типов СУБД. Реализуется на уровне команды Create Table при описании поля (<поле> <стандартный тип>). Действует постоянно, пока существует таблица.

2) ***Ограничение на домен***, реализуется, используя объект базы данных – ***пользовательский тип*** (Create Type). Пользовательский тип уточняет стандартный тип данных по размерности и возможности использования неопределенного значения. Ограничение используется на уровне команды Create Table при описании поля (<поле> <пользовательский тип >). Действует постоянно, пока существует таблица.

3) Ограничение, допускающее или не допускающее использовать в качестве значения поля ***неопределенное значение*** – ***NULL***. Реализуется на уровне команды Create Table при описании поля (<поле> <тип поля> Null). Действует постоянно, пока существует таблица.

4) ***Ограничение значением по умолчанию при создании таблицы*** – ***Default***. Определяет значение поля, даже если оно не указано при операциях добавления и изменения. Реализуется на уровне команды Create Table при описании поля (<поле> <тип поля> Default). Действует постоянно, пока существует таблица.

5) ***Ограничение на значения поля при создании таблицы – Check***. Команда create table: Constraint Check <поле> <оператор отношения> <значение>. Действует постоянно, пока существует таблица.

6) ***Ограничение на значения данных на уровне записи при создании таблицы.*** Команда create table Constraint Check <поле1> <оператор отношения> <поле2>. Действует постоянно, пока существует таблица.

7) ***Ограничение первичного ключа*** (Primary Key). Позволяет создать неповторяющиеся записи отношения, что соответствует определению отношения. Первичный ключ один на уровне таблицы. По этому ключу записи в таблице упорядочиваются. При физической реализации кроме файла таблицы создается индексный файл первичного ключа, реализующий индексно-последовательный метод доступа к данным. Первичный ключ может быть атомарным, то есть состоять из одного поля таблицы, или составным, то есть включать несколько полей таблицы. Реализуется на уровне команды Create Table при описании поля (<поле> <тип поля> Primary Key для атомарного ключа) или после описания полей на уровне записи – Constraint Primary Key(список полей первичного ключа) для составного первичного ключа. Действует постоянно, пока существует таблица.

8) ***Ограничение альтернативного ключа*** (Unique). Позволяет контролировать кроме первичного ключа наличие неповторяющихся подмножеств значений полей в пределах отношения. Альтернативных ключей может быть несколько для одной таблицы. Записи таблицы не упорядочиваются по значениям альтернативных ключей. Альтернативные ключи позволяют на уровне логики контролировать уникальность некоторых подмножеств атрибутов отношения и могут использоваться для ускорения поиска данных в пределах этих подмножеств. При физической реализации кроме файла таблицы создаются индексные файлы альтернативных ключей, реализующие индексно-произвольный метод доступа к данным. Альтернативный ключ может быть атомарным, то есть состоять из одного поля таблицы, или составным, то есть включать несколько полей таблицы. Реализуется на уровне команды Create Table при описании поля (<поле> <тип поля> Unique для атомарного ключа) или после описания полей на уровне записи – Constraint Unique (список полей ключа) для составного альтернативного ключа. Действует постоянно, пока существует таблица.

9) ***Ограничение вторичного ключа.*** Вторичные ключи допускают повторяющие значения и используются для ускорения операций манипулирования данными. Вторичных ключей может быть несколько для одной таблицы. При физической реализации кроме файла таблицы создаются индексные файлы вторичных ключей, реализующие инвертированный метод доступа к данным. Вторичный ключ может быть атомарным, то есть состоять из одного поля таблицы, или составным, то есть включать несколько полей таблицы. Реализуется на уровне команды Create Table при описании поля. Действует постоянно, пока существует таблица.

10) ***Ограничение значением по умолчанию как объект базы данных – Default***. Определяет значение, которое может вставляться, если оно не указано при операциях добавления и изменения, в качестве значения одного или нескольких полей различных таблиц базы данных. Реализуется командой Create Default. Действует не постоянно, а от момента привязки до момента отвязки от поля таблицы и работает медленнее по сравнению с ограничением значения по умолчанию на уровне таблицы.

11) ***Ограничение на значения данных как объект базы данных – Rule.*** Правило – объект базы данных, определяющий ограничения на значения одного или нескольких полей различных таблиц базы данных. Реализуется командой Create Rule. Действует не постоянно, а от момента привязки до момента отвязки от поля таблицы и работает медленнее по сравнению с ограничением Check на уровне таблицы.

12) ***Сложные ограничения на значения данных, учитывающие логику предметной области – Trigger.*** Триггер – объект базы, являющийся специальным видом хранимой процедуры, автоматически срабатывающий при операциях ведения данных. Реализуется командой Create Trigger (сложное правило, сложное значение по умолчанию, затрагивающее не все записи отношения, а те, которые удовлетворяют некоторому условию).

**2. Ограничения на структуру данных (для реляционных баз данных это синхронное перемещение указателей между связанными таблицами и синхронное выполнение операций по ведению данных между связанными таблицами).**

1) ***Ограничение внешнего ключа –Foreign Key.*** Ограничение внешнего ключа реализует синхронное перемещение указателей между связанными таблицами. Для успешной реализации этого ограничения необходимо, чтобы таблица, на которую ссылается ограничение (главная

или родительская таблица при установлении связи) уже была создана. Для полей связи в родительской таблице должны существовать ограничения первичного ключа или альтернативного ключа (уникальности). Тип и размерность полей в родительской и подчиненной (дочерней) таблице, по которым устанавливается связь, должны совпадать. Ограничение внешнего ключа может быть атомарным, то есть создаваться для одного поля, или составным, то есть создаваться для нескольких полей таблицы. При физической реализации рекомендуется в дочерней таблице по полю связи создать ограничение вторичного ключа, если реализуется связь один ко многим, либо ограничение альтернативного ключа, если реализуется связь один к одному. Ограничение внешнего ключа реализуется на уровне команды Create Table при описании поля (<поле> <тип поля> References <имя родительской таблицы> <имя поля связи родительской таблицы> для атомарного внешнего ключа) или после описания полей на уровне записи – Constraint <имя ограничения> Foreign Key (<список полей дочерней таблицы>) References <имя родительской таблицы>(<список полей родительской таблицы>) для составного внешнего ключа. Действует постоянно, пока существует связь между таблицами или существуют родительская и дочерняя таблицы.

2) ***Ограничения ссылочной целостности – restrict***. Это ограничение ссылочной целостности используется для реализации запрета выполнения операций удаления и/или изменения записей в родительской таблице, если существуют записи в дочерней таблице с таким же значением поля связи. Это ограничение реализуется с помощью системного или пользовательского триггера.

3) ***Ограничения ссылочной целостности – cascade***. Это ограничение ссылочной целостности используется для реализации синхронного выполнения операций удаления и/или изменения записей в родительской и дочерней таблицах, если значения полей связей в обеих таблицах совпадают. Реализуется это ограничение с помощью системного или пользовательского триггеров.

### 4.4. Рекомендации по физическому проектированию баз данных

1. С учетом выбранной СУБД проектируется физическая схема базы данных. Определяются типы полей записей с указанием размерности полей, свойств полей.

2. В соответствии с логической схемой базы данных определяются ограничения целостности данных и средства их реализации.

3. Выполняется автоматическая генерация схемы базы данных, соответствующая физической схеме для выбранной СУБД, используя CASE – средство проектирования базы данных (например, ERWin Data Modeler).

После физического проектирования создается физическая схема базы данных, можно заполнять отношения записями и работать с запросами в информационной системе

# ГЛАВА 5. РАБОТА С БАЗАМИ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ НА ПРИМЕРЕ MS SQL SERVER

## 5.1. Особенности клиент-серверной технологии. Менеджеры для MS SQL Server

*Сервер базы данных* – это логический процесс, отвечающий за обработку запросов к базе данных. Осуществляет управление хранением данных и контроль целостности данных. Реализуется с помощью ядра СУБД (MS SQL Server). *Клиент базы данных* – это логический процесс, посылающий серверу запрос на обслуживание. Реализуется с помощью менеджера – отдельной программы, с которой работает пользователь и обращается к серверу базы данных.

Примеры менеджеров для СУБД MS SQL Server – MS SQL Server Management Studio, DBeaver**.** Менеджер MS SQL Server Management Studio может ставиться одновременно с установкой MS SQL Server или отдельно. Он представляет графический клиент для работы с сервером, через который в удобном виде можно создавать, удалять, изменять базы данных и управлять ими, осуществлять контроль вводимых данных, управлять интерфейсом пользователя, приводить данные к нужному формату для пользователя. Является удобным средством администрирования и управления данными. Менеджер DBeaver также является клиентом SQL и инструментом администрирования реляционных баз данных**,** реализуемых средствами таких СУБД как *MySQL, MS SQL Server, PostreSQL, SQLite, Oracle, DB2, MariaDB, Sybase, Teradata, Netezza и другие.* Этот менеджер может работать и некоторыми базами данных NoSQL, включая *MongoDB, Cassandra, Redis, Apache Hive и другие*. Это настольное приложение, написанное на Java и позволяющее работать с реляционными базами данных через драйвер JDBC.

## 5.2. Запросы к базе данных. Аналитические запросы

*Команда выборки* данных, реализующая операцию селекции, выглядит следующим образом [5]:

**SELECT** [<ограничения на количество выводимых записей>]

<список выбираемых элементов>

**FROM** <откуда осуществляется выборка>

[**WHERE** <условие выборки>]

[**GROUP BY** <список группируемых элементов>[**HAVING** <условие на группируемые данные>]]

[**ORDER BY** <список сортируемых элементов>]

В разделе команды SELECT <ограничения на количество выводимых записей> могут присутствовать следующие ключевые слова: ALL, (означающее выборку всех записей из полученного набора данных, без учета наличия возможных повторений), DISTINCT (выборка неповторяющихся записей из набора), TOP <количество> [PERCENT] (выборка указанного количества записей, возможно не записей а числа записей в процентном отношении от общего количества записей выборки).

В разделе команды SELECT <список выбираемых элементов> указывается через запятую список отбираемых элементов набора данных. Элементом может быть поле таблицы, может быть скалярная функция или функция агрегирования, может быть подзапрос. Элементу можно присвоить новое имя или псевдоним, которое видно только в пределах самой команды. Для введения псевдонима используется синтаксическая конструкция: <элемент> AS <псевдоним>.

В разделе команды SELECT <откуда осуществляется выборка> создается реляционный набор данных. В этом разделе указывается список элементов, включающий таблицы, виды, подзапросы, которые могут быть соединены с помощью операций соединения. Каждый элемент этого раздела также можно переименовать, введя псевдоним. Соединение элементов осуществляется следующим образом:

<элемент> [[AS] <псевдоним>] [<тип соединения>] JOIN <элемент> [[AS] <псевдоним>] ON <условие соединения>.

Тип соединения может быть внутренним (INNER), левым внешним (LEFT OUTER), правым внешним (RIGHT OUTER), полным (FULL) [5].

В разделе команды SELECT <условие выборки> присутствует предикат, сформированный с помощью операций отношений, логических операций, скалярных функций с применением имен элементом присутствующих в разделах <список выбираемых элементов> и <откуда осуществляется выборка>.

В разделе команды SELECT <список группируемых элементов> указывается список имен элементов из раздела <список выбираемых элементов> или номеров этих элементов. В результате выполнения этого раздела команды SELECT данные набора объединяются (группируются), возможно, в несколько множеств записей, каждое из которых имеет одинаковые значения элементов, присутствующих в списке группируемых элементов. В сочетании с группировкой обычно используется выполнение функций агрегирования для каждого множества сгруппированных данных. При необходимости ввести ограничение на функции агрегирования используется раздел Having <условие на группируемые данные>.

В разделе команды SELECT <список сортируемых элементов> указывается через запятую имена или номера элементов из раздела <список выбираемых элементов>. В результате выполнения этого раздела команды данные в результате выборки сортируются в указанном порядке, насколько это возможно в реляционной модели данных.

*Подзапрос* – запрос, вложенный в основной запрос. Подзапрос синтаксически, внутри основного запроса, заключается в круглые скобки. Подзапросы могут встречаться в разных разделах инструкции SELECT. Различают следующие виды подзапросов. *Скалярный подзапрос* – подзапрос, возвращающий одно значение. *Табличный подзапрос* – подзапрос, возвращающий несколько значений. *Вложенный табличный подзапрос* – подзапрос, возвращающий несколько столбцов и несколько строк. Каждый из названных подзапросов могут участвовать в коррелированных подзапросах. *Коррелированный подзапрос* – подзапрос, зависящий от значения во внешнем запросе. То есть в коррелированном запросе внутренний запрос выполняется по одному разу для каждой записи, извлеченной внешним запросом. Если существует несколько уровней вложенности подзапросов, коррелированный подзапрос может ссылаться на любой уровень главного запроса, который выше его собствен-

ного уровня. Схематично разделы команды SELECT с точки зрения видов допустимых подзапросов можно представить таким образом:

SELECT (<скалярный подзапрос>) FROM (<вложенный табличный подзапрос>) AS<псевдоним> WHERE <нечто>=(<скалярный подзапрос>)|<нечто> IN (<табличный подзапрос>) |EXISTS (<любой вид подзапроса>).

В информационных системах выполняются не только простые запросы к базе данных, но и аналитические запросы. Аналитические запросы могут использовать следующие разделы:

1. Функции ROLLUP, CUBE в сочетании с группировкой;
2. Использование оконных функций;
3. Использование функций ранжирования;
4. Использование аналитических функций в сочетании с оконными функциями.

Функции ROLLUP, CUBE позволяют дополнительно с группировкой выполнять анализ и подсчет по отдельным полям. Ниже на примере показаны особенности применения этих функций. Пусть имеется следующая таблица Tab1

| **a** | **b** | **c** |
|---|---|---|
| 1 | 1 | 10 |
| 1 | 1 | 20 |
| 1 | 1 | 30 |
| 2 | 1 | 10 |
| 2 | 2 | 10 |
| 2 | 3 | 10 |

Запрос с обычной группировкой: SELECT a, b, sum(c) AS P FROM tab1 Group By a, b

Результат запроса –

| **a** | **b** | **P** |
|---|---|---|
| 1 | 1 | 60 |
| 2 | 1 | 10 |
| 2 | 2 | 10 |
| 2 | 3 | 10 |

Запрос группировкой с функцией Rollup**:** SELECT a, b,sum(c) FROM tab1 Group By Rollup(a, b)

Результат этого запроса –

| **a** | **b** | **P** |
|---|---|---|
| 1 | 1 | 60 |
| **1** | **null** | **60** |
| 2 | 1 | 10 |
| 2 | 2 | 10 |
| 2 | 3 | 10 |
| **2** | **null** | **30** |
| **Null** | **Null** | **90** |

Запрос с группировкой и функцией CUBE:

SELECT a,b, sum(c) AS P FROM tab1 Group By Cube (a,b)

Результат выполнения этого запроса –

| **a** | **b** | **P** |
|---|---|---|
| 1 | 1 | 60 |
| **1** | **null** | **60** |
| 2 | 1 | 10 |
| 2 | 2 | 10 |
| 2 | 3 | 10 |
| **2** | **null** | **30** |
| **null** | **1** | **70** |
| **null** | **2** | **10** |
| **null** | **3** | **10** |
| **null** | **null** | **90** |

Можно управлять иерархией в предложениях Cube Rollup, указав Having, например a=1.

Если используется анализ по 3-м полям – a, b, d, sum(c), то можно, например, записать в Group By a, Rollup(b,d) или Group By d, Cube (a,b).

*Оконные функции*

<функция – агрегирования (аргумент) или ранжирования>

OVER (Partition by p1,…..pn [Order by o1,…ok])

Оконные функции вычисляются внутри разделов – Partition**,** определяемых списком p1,…..pn имен полей. В разделе p1,…..pn определяется начальная и конечная точка «окна» для каждой строки. Вычисление оконных функций в запросах осуществляется после разделов Where, Group By, Having. Раздел Order by используется только с функциями ранжирования. Если Partition by отсутствует, то агрегатные функции применяются ко всему результирующему набору строк запроса. В отличие от группировки, где на каждую группу получается одна строка с агрегатной функцией, в Partition by можно добавить агрегат к несгруппированным строкам. То есть окно – набор строк, определяемый пользователем, а оконная функция вычисляет значения для каждой строки в результирующем наборе, полученном из окна. В одном запросе можно использовать несколько статистических функций или функций ранжирования, использующих свое окно. Как и все аналитические функции, оконные функции работают с готовой выборкой. Ниже рассмотрен пример использования оконной функции. Запрос имеет результат, аналогичный результату запроса с самосоединением таблиц. Дана таблица tab1

| a | b | c |
|---|---|---|
| 1 | 1 | 10 |
| 1 | 2 | 10 |
| 1 | 3 | 10 |
| 2 | 1 | 10 |
| 2 | 2 | 10 |
| 2 | 3 | 10 |

Запрос с самосоединением таблиц:

SELECT s.a, s.b, t.d FROM tab1 s JOIN (SELECT b, sum(c) d FROM tab1 Group By b) AS t ON s.b=t.b

Результат запроса –

| **a** | **b** | **d** |
|---|---|---|
| 1 | 1 | 20 |
| 2 | 1 | 20 |
| 1 | 2 | 20 |
| 2 | 2 | 20 |
| 1 | 3 | 20 |
| 2 | 3 | 20 |

Результат подзапроса – 3 строки : 1 20, 2 20, 3 20

Запрос с оконной функцией – STLECT a, b, sum(c) AS d **OVER (Partition By b) FROM** tab1 будет иметь тот же результат

Ранг – номер, порядок строки. Функция ранжирования возвращает ранг записи внутри «окна». В общем случае ранг – это число, отражающее положение или «вес» записи относительно других записей в наборе. Для функций ранжирования обязательна инструкция Order by. Функции ранжирования случайные или недетерминированные, то есть для одного и того же набора может быть получен разный результат функции. Различают функции: ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE()

Функция ROW_NUMBER() нумерует записи внутри окна. Если в OVER отсутствует Partition By, то есть OVER(), то за окно принимается вся выборка, и выполнятся нумерация всех записей выборки. Нумерация всегда начинается с единицы. Ниже рассмотрены примеры работы запросов с использованием различных функций ранжирования.

Дана таблица tab1

| a | b | c |
|---|---|---|
| 1 | 1 | 10 |
| 1 | 2 | 20 |
| 1 | 3 | 30 |
| 2 | 1 | 30 |
| 2 | 2 | 20 |
| 2 | 3 | 10 |
| 2 | 2 | 10 |

Результат запроса SELECT b, c, **ROW_NUMBER() OVER** (Partition By b **Order by** c Desc) AS d FROM tab1

| : b | c | d |
|---|---|---|
| 1 | 30 | 1 |
| 1 | 10 | 2 |
| 2 | 20 | 1 |
| 2 | 20 | 2 |
| 2 | 10 | 3 |
| 3 | 30 | 1 |
| 3 | 10 | 2 |

Функция RANK() предназначена для ранжирования записи внутри окна, при этом, если колонка для группировки не задана явным образом, то за окно принимается вся выборка. Рангом каждой записи является количество уже ранжированных записей с более высоким рангом, чем текущая, плюс единица. Если встретится несколько записей с одинаковым значением, по которому проводится ранжирование, то эти записям будет присвоен одинаковый ранг. Однако при этом следующая запись с новым значение получит такой ранг, как будто бы предыдущие записи получили свой новый уникальный номер, то есть получается дырка в нумерации, равная количеству одинаковых записей минус единица.

Результат запроса **S**ELECT b, c, **RANK()** OVER(Partition By b Order By c Desc) AS d FROM tab1:

| b | c | d |
|---|---|---|
| 1 | 30 | 1 |
| 1 | 10 | 2 |
| 2 | 20 | **1** |
| 2 | 20 | **1** |
| 2 | 10 | **3** |
| 3 | 30 | 1 |
| 3 | 10 | 2 |

Функция DENSE_RANK() выполняет «плотное» ранжирование, то есть работает как Rank(), но без дырок.

Результат запроса SELECT b, c, **DENSE_RANK()** OVER(Partition By b Order By c Desc) as D **from** tab1 –

| b | c | d |
|---|---|---|
| 1 | 30 | 1 |
| 1 | 10 | 2 |
| 2 | 20 | **1** |
| 2 | 20 | **2** |
| 2 | 10 | **3** |
| 3 | 30 | 1 |
| 3 | 10 | 2 |

Функция Ntile() позволяет разделить записи внутри «окна» на указанное количество групп. Для каждой записи она вернет номер группы, к которой принадлежит данная запись. Нумерация групп также начинается с единицы. Если количество записей в окне не делится на количество групп, то получится два типа групп с разным количеством записей, отличающимся на единицу, при этом сначала будут выведены группы с большим количеством записей, а затем с меньшим.

Операторы Union, Intersect, Except в запросах используются для объединения, пересечения или разности множества кортежей, полученных в нескольких запросах.

Команда Union показывает все строки всех запросов как один результирующий запрос, при этом итоговый набор имеет заголовки полей выборки первого запроса. Команда Intersect извлекает идентичные строки из одного или нескольких запросов (аналог внутреннего соединения или пересечения в теории множеств). Команда Except включает все записи, полученные первым запросом, которые не обнаружены в результате последующего запроса (разность в теории множеств). Для всех названных команд должно выполняться требование – порядок, количество столбцов во всех запросах должно быть одинаковое, типы данных столбцов совместимы. Синтаксис:

<команда SELECT 1>
UNION или INTERSECT или EXCEPT
<команда SELECT 2>

### 5.3. Оптимизация запросов. План и статистика выполнения запросов

Проектировщик базы данных должен быть знаком с оптимизатором запросов выбранной Субд. При реализации запросов к базе данных каждое средство реализации имеет программу оптимизации запросов. В качестве примера, ниже рассмотрены стратегии оптимизации запросов в MS SQL Server. Оптимизатор запросов определяет наилучший план извлечения данных и наиболее эффективный путь их изменения. Он выбирает способ поиска записей, объединения таблиц и сортировки. *Стоимость запроса* – оценка предполагаемого количества и объема операций ввода/вывода, исходя из предполагаемого количества возвращаемых записей. При этом используется информация о структуре данных, размерах таблиц и индексах, индексная статистика. *Исполнение запроса включает этапы: с*интаксический разбор, стандартизация – преобразование дерева запроса с целью удаления избыточных структур, оптимизация – выработка плана исполнения запроса, компиляция запроса, исполнение запроса.

*Процесс оптимизации запроса включает этапы*:

1. Анализ запроса; 2. Выбор индексов; 3. Выбор порядка соединения; 4. Выбор наилучшего плана; 5. Анализ запроса.

*Анализ запроса* включает определение аргументов поиска, условий или (OR), условий межтабличного соединения (Join). *Цель определения аргументов поиска* – установление возможности использования индексов. Аргументом поиска является условие, позволяющее сузить область поиска данных. К таким относятся условие, задаваемые простыми операторами сравнения, где сравниваются поля и константы.

*Вторая фаза оптимизации – выбор индексов*. При этом проверяется, существует ли условие по полю, на котором настроен индекс. При этом условии должно быть аргументом поиска или условием соединения. Если существует индекс, который может быть задействован для доступа к данным, определяется избирательность заданного условия. *Избирательность условия* – это отношение количества записей, удовлетворяющих условию, к общему количеству записей в таблице.

*Выбор порядка соединения таблиц* – третья основная фаза оптимизации запроса, цель которой – выбор лучшей, с точки зрения минимизации количества операций чтения, последовательности таблиц. Когда в запросе указано несколько таблиц, требуется выбрать оптимальный план соединения таблиц. Оптимизатор рассматривает возможные варианты, оценивает необходимое количество чтений, и выбирает наименее затратный муть. Для оценки стоимости соединения оптимизатор использует избирательность соединения.

# ГЛАВА 6. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ НА УРОВНЕ БАЗ ДАННЫХ НА ПРИМЕРЕ MS SQL SERVER

Важным вопросом проектирования баз данных информационных систем является обеспечение безопасности данных. Под *безопасностью данных* понимается защита *от непреднамеренного* доступа к данным, *идентификации* и *разрушении*. *Под секретностью данных* понимается защита от преднамеренного доступа, модификации и разрушения.

## 6.1. Разграничение прав доступа к данным. Уровни системы безопасности

Для доступа к объектам и данным в базах данных пользователь проходит два уровня проверки безопасности – уровень аутентификации и уровень проверки разрешений.

*Аутентификация пользователя* – процесс допуска или не допуска к соединению с базой данных. В SQL Server существует два режима контроля проверки возможности подключения: *на уровне Windows NT* (Windows NT authentication - по умолчанию) и *смешанная аутентификация на уровне SQL Server и Windows NT* (SQL Server authentication). Вид аутентификации задается в свойствах сервера или соединения. В SQL Server *уровни безопасности реализуются* на уровне сервера базы данных и на уровне базы данных.

## 6.2. Учетные записи. Пользователи и роли. Система разрешений

*Учетная запись* - имя пользователя для входа в систему**.** При аутентификации Windows NT имеется только одна учетная запись – **sa**. Эта учетная запись имеет все права на доступ ко всем объектам всех баз данных. При смешанной аутентификации можно задать собственное имя пользователя (со своим набором прав или разрешений) для входа и пароль во время соединения. *На уровне сервера базы данных* для смешанной аутентификации можно добавить новую **учетную запись** с помощью мастера создания нового пользователя New Login через Security сервера дерева административной консоли, либо с помощью системной хранимой процедуры: Sp_addlogin или с помощью команды **CREATE LOGIN**. Удаление имен пользователей для входа: Sp_droplogin ‘имя пользователя для входа’ (DROP LOGIN). *На уровне базы данных* можно создать *пользователя* базы данных, который будет входить под конкретной и только одной учетной записью, с помощью мастера создания нового пользователя New User через Security базы данных дерева административной консоли, либо с помощью системной хранимой процедуры: Sp_addluser или с помощью команды **CREATE USER**.

**Роль** – именованный набор прав в рамках сервера или конкретной базы данных. Виды ролей – *встроенные роли, пользовательские роли, роли приложений.* Встроенные роли в свою очередь могут быть серверными или ролями базы данных [9].

Для добавления *пользовательских ролей* можно воспользоваться системной хранимой процедурой: Sp_addrole <параметры> или воспользоваться командой **CREATE ROLE**.

В каждой базе данных имеется *специальная роль – роль* **public**. Каждый пользователь базы данных – член этой роли. Она не может быть удалена, её нельзя присвоить вручную, ей можно присвоить определенные разрешения, которые затем автоматически передаются всем пользователям базы данных. Кроме пользовательских ролей можно создавать *роли приложений*. Эти роли нельзя присвоить пользователям. Чтобы использовать предоставленные этой ролью разрешения, её вначале надо активизировать. Роли приложений создаются хранимой процедурой: Sp_addapprole ‘роль’,’пароль’ или с помощью команды **CREATE APPLICATION ROLE**. Активизация роли приложения: Sp_setapprole ‘имя роли’, ‘пароль’.

При планировании разрешений решаются вопросы, **кому** будут разрешения предоставлены и **на какие объекты**. **Кому** – *разрешения представляются* **учетным записям**, **пользовательским ролям**, **роли public, ролям приложений**. *На какие объекты – предоставление разрешений* можно выполнить **базам данных, таблицам, видам, хранимым процедурам.** Разрешения бывают объектные и командные. *Объектные разрешения* управляют доступом к таблицам, видам, хранимым процедурам.

Для каждого объекта могут быть предоставлены различные **типы объектных разрешений**: **SELECT** – можно таблицу, вид (читать), **UPDATE** – можно таблицу, вид (изменять), **INSERT** – можно таблицу, вид (добавлять), **DELETE** – можно таблицу, вид (удалять), **EXECUTE** – можно выполнить хранимую процедуру, **ALL** – предоставляет любые разрешения. Разрешать или запрещать доступ к объектам базы данных может владелец объекта.

Предоставление прав для объектных разрешений реализуется командой –

**GRANT** {список_прав доступа |All}
**ON** {таблица|вид [(столбец, …)] | хранимая процедура}
**TO** {список_имен ролей или учетных записей | PUBLIC}

Удаление прав реализуется командой –

**REVOKE** {список_прав доступа |All}
**ON** {таблица| вид [(столбец, …)] | хранимая процедура}
**TO** {список_имен ролей или учетных записей | PUBLIC}

Запрещение доступа к объектам реализуется командой –

**DENY** { ALL | список разрешений}
**ON** {таблица| вид [(столбец, …)] | хранимая процедура }
**TO** {список_имен ролей или учетных записей | PUBLIC}

**Схема –** это **объект**, являющийся **контейнером объектов базы данных**. Схеме, как и роли можно предоставить систему разрешений. Только роль присваивается учетной записи, а схема объединяет набор объектов. Таким образом, схема используется для отделения владельца объекта от конкретного пользователя. Имя схемы по умолчанию, объединяющей все объекты базы данных – **dbo**. Команда, создающая схему базы данных – **CREATE SHEMA**. Например: CREATE LOGIN Im1 WITH Password =’123’, Default_database =Prim

**CREATE DATABASE** Prim
**USE** Prim
**CREATE SHEMA** Sh1
**CREATE TABLE** Sh1.tab1(<список полей>)
**CREATE USER** Us1 FOR Login Im1 WITH Default_Schema=Sh1
**GRANT SELECT ON SCHEMA**::sh1 TO Us1

*Синоним* – это объект базы данных, который задает альтернативное имя, обычно более короткое, чем исходное. Полное имя объекта в реляционных базах данных предполагает следующую цепочку разыменований – Имя сервера. Имя базы данных. Имя схемы. Имя объекта. Имя подобъекта. Команда, определяющая короткий синоним –

**CREATE SYNONYM** <альтернативное имя> **FOR** <исходное имя>

### 6.3. Шифрование данных

Для обеспечения защиты данных от недопустимого просмотра данных в случае нарушения прав доступа к данным используется шифрование данных. Для шифрования данных в MS SQL Server предусмотрено четыре способа шифрование при помощи паролей, шифрование при помощи симметричных ключей, шифрование при помощи асимметричных ключей, шифрование при помощи сертификатов. В MS SQL Server реализована ***иерархия ключей***.

Имеется *главный сервисный ключ*, который создается автоматически, при установке системы, он управляет всеми ключами и сертификатами, но его расшифровать нельзя, можно сохранить для возможности восстановления при разрушении. Каждая база данных имеет *главный ключ базы данных*, который создается при использовании оператора CREATE MASTER KEY. Этот ключ защищен главным сервисным ключом и используется при создании симметричных, асимметричных ключей и сертификата.

Шифрование при помощи паролей реализуется следующим образом –

**CREATE TABLE** dbo.Primer (Pole **nvarchar(100)**)

Добавляем зашифрованную запись, используя функцию **EncryptByPassPhrase**

**INSERT INTO Primer** (**EncryptByPassPhrase**(‘P@s’,N’Зашифрованные данные’) )

Для дешифровки используется функция **DecryptByPassphrase**

**SELECT** (**Convert** (Nvarchar(100), **DecryptByPassphrase**(‘P@s’,Pole))) **FROM** Primer

Шифрование при помощи симметричных ключей производится ***быстрее*** по сравнению с другими способами, поэтому его используют при необходимости шифрования ***данных больших объемов.*** При создании симметричного ключа его можно защищать не только паролем, но и другим симметричным ключом, асимметричным ключом или сертификатом. Кроме этого можно указать один из алгоритмов шифрования. Например, AES_128, Des,AES_192,AES_256 и другие. Сначала необходимо **создать симметричный ключ** –

**CREATE SYMMETRIC KEY** SymKey1 **With Algoritm** = AES_128 Encryption **Password**=’P@s’

Перед использованием симметричного ключа ключ надо **открыть:**

**OPEN SYMMETRIC KEY** SymKey1 **Decryption By** Password=’P@s’

Затем *добавляем зашифрованные ключом данные в таблицу, используя функцию* – **EncryptByKey INSERT** into Primer (**EncryptByKey**(Key_Guid(‘SymKey1’),N’Зашифрованные данные’))

При расшифровке используем ту же функцию и выполняем действие –

**SELECT** (Convert(Nvarchar(100), **EncryptByKey**(Pole)) **FROM** Primer

При шифрование при помощи асимметричных ключей сначала необходимо создать асимметричный ключ

**CREATE ASymmetric Key** ASymKey1 With Algoritm = RSA_512 Encryption Password=’P@s’

Затем добавляем зашифрованные ключом данные в таблицу, используя функцию **EncryptByAsymKey** –

**INSERT INTO Primer** (**EncryptByAsymKey**(**AsymKey_ID**(‘ASymKey1’),
N’Зашифрованные данные’))

При расшифровке необходимо выполнить команду –

**SELECT**(**Convert**(Nvarchar(100),
**EncryptByAsymKey**(AsymKey_ID(‘ASymKey1’),Pole,N’P@s’))) **FROM** Primer

Для шифрования при помощи сертификата сначала создается сертификат –

**CREATE Certificate** Sert1 **With** Subject = ‘Пример шифрования’ **Encryption Password**=’P@s’

Добавление зашифрованной записи с помощью функции **EncryptByCert**

**INSERT** into Primer (**EncryptByCert**(Cert_ID(‘Sert1’),N’Зашифрованные данные’))

Расшифровка значения поля –

**SELECT** (**Convert**(Nvarchar(100), **EncryptByAsymKey**(**EncryptByCert**(Cert_ID(‘Sert1’),
Pole, N’P@s’))) **FROM** Primer

### 6.4. Архивирование и восстановление данных

При возникновении проблем с физическими устройствами, возникновении сбоев в программных средствах, при непреднамеренном уничтожении данных из базы данных необходимо производить резервное копирование, с последующим восстановлением.

Существует три основных способа резервного копирования:

1. **Полное архивирование** (или **дамп базы данных**) подразумевает создание копий всех страниц с данными в БД, включая все системные таблицы. При выполнении полного архивирования создается копия данных и журнала транзакций.

2. **Разностное архивирование** (предусматривает архивирование данных без журнала транзакций).

3. **Архивирование журнала транзакций**

При планировании резервного копирования и устройство резервного копирования необходимо:

1.Определить место копии данных – файл или устройство для резервного копирования данных. Использование устройства резервного копирования предпочтительнее.

2. Создать устройство резервного копирования средствами SQL Server, например, используя хранимую процедуру sp_addumpdevice или используя инструментальные средства. **Также для того, чтобы определить устройство резервного копирования, вы должны быть в базе данных master**, так как устройство резервного копирования – это ресурс, определяемый для всего сервера, и записи о нем должны храниться в таблице sys_devices базы данных master.

Например, для создания устройства резервного копирования выполняются команды:

**USE** master

**EXEC** sp_addumpdevice ‘disk’,’newdumpdev’,’c:\dump\newdump.dat’

**BACKUP DATABASE {'имя базы данных’} TO устройство** [<опции>]

**BACKUP LOG {‘имя базы данных’} TO устройство резервного копирования** [<опции>]

При резервном копировании учитывается *размер базы данных, частоту обновления* в ней информация. Во время резервного копирования можно делать все транзакции, включая операции без записи в журнал транзакций.

*Способ восстановления базы данных зависит от способа создания резервной копии.* Если была создана резервная копия только базы данных, то восстанавливаться будет база данных. Если – полная резервная копия, то восстанавливается база данных вместе с журналом транзакции. Команды восстановления баз данных и журналов транзакций:

**RESTORE DATABASE** [<опции>]**, RESTORE LOG** [<опции>]

При восстановлении на основе архивной копии база данных должна быть **закрыта**, так как данные переписываются на загружаемые данные.

### 6.5. Перемещение и тиражирование данных

*Перемещение данных* – передача данных в базу данных *из разных источников*. При перемещении *возможно преобразование* данных в другой формат; *распространение* данных *на другие серверы* по сети; *загрузка* архивной копии данных. Служба **Data Transformation Service (DTS)** используется для **импорта, экспорта и трансформации данных между разнородными источниками данных**. Утилита Bulk Copy Program (bcp.exe или **bcp**) используется для **импорта и экспорта данных из таблиц баз данных SQL Server**. Она может работать с файлами самых разных форматов –

**BCP <имя таблицы>** | <имя вида> | 'запрос' {**IN |OUT** | queryout | format} **<файл данных>** [- C кодировка] [-S имя сервера] [-T] [<опции>]

*Удаленный доступ к данным* – это работа с данными, находящимися на разных серверах. Различают три способа обращения к данным, находящимся на разных серверах: удаленный вызов процедур, распределенные запросы, распределенные транзакции.

Удаленный вызов процедур (RCP, remote procedure call) используется для доступа к данным, которые находятся на другом сервере SQL Server. Оба сервера необходимо настраиваются на взаимодействие друг с другом. Распределенные запросы позволяют обратиться к данным связанного сервера. Связанный сервер представляет собой заранее сконфигурированный источник данных OLE DB. Функции OPENQUERY и OPENROWSET используются для работы с распределенными запросами [9].

*Распределенные транзакции* работают с данными, находящимися на двух и более серверах. Для явного указания распределенной транзакции используется команда:

**BEGIN DISTRIBUTED TRANSACTION** [<имя транзакции>]

*Распределенные данные* **–** это множество копий одних и тех же данных, размещенных на разных серверах. Кроме распределенных транзакций для тиражирования данных используется настройка репликаций в системе. *Распределенные транзакции гарантируют согласованность всех копий данных*, но отказ транзакции на одном из узлов ведет за собой отказ на всех узлах, поэтому распределенные транзакции используются только при постоянной синхронизации данных. Распределенные транзакции характеризуются практически нулевой задержкой.

*Репликация (тиражирование) – распространение* данных, которые дублируются и распространяются из исходной базы данных в другую базу данных, обычно расположенную на

другом сервере с *задержкой,* возможно *автономно* и *без согласования*. *Задержка* – это интервал между обновлением распределенных данных. При репликации возможны задержки от нескольких секунд до нескольких дней. *Автономность узла* характеризует степень его независимости. В распределенных транзакциях узлы должны быть постоянно подключены к сети, при репликации допускается задержка от нескольких секунд до нескольких дней. *Согласованность* предполагает корректное изменение во всех копиях данных. Распределенные транзакции гарантируют полную и постоянную согласованность. При репликации может вообще не гарантироваться согласованности модификации данных.

При реплицировании данных используется специальная терминология. **Издатель** (Published) – сервер с исходной базой данных. Издатель у любого подписчика только один. **Подписчик** (Subscriber) – это база данных, которая запрашивает и получает публикации точно так же, как и в случае с обычными журналами. **Распространитель** (Distributor) – сервер-получатель копии всех изменений в опубликованных данных, сохраняет изменения и открывает доступ к ним соответствующим подписчикам. **Статья** (Article) – это единый реплицируемый элемент данных. **Публикация** – это набор статей, предназначенных для тиражирования. **Подписка** – это конфигурация, определяющая способ получения подписчиком публикации от издателя. Существует **два вида подписок**: активные (**Push**) и пассивные(**Pull**). Виды репликации: снимочная репликация, транзакционная репликация, репликация сведением (слиянием). ***Снимочная репликация (репликация моментальных снимков)*** – периодическая рассылка всей публикации серверам-подписчикам. Это самая простая в настройке и сопровождении репликация (snapshot). Моментальный снимок создается с помощью агента моментальных списков **Snapshot Agent**, который записывает структуру и данные. Затем моментальный снимок посылается агентом распределения **Distribution Agent**, который передает данные подписчикам. Подписчики получают не изменения к уже имеющимся данным, а полностью новый набор данных, который заменяет прежний. При этом старые данные уничтожаются и на их место записываются новые. ***Репликация транзакций*** – передаются не новые таблицы целиком, а только изменения к опубликованным таблицам и команды на исполнение хранимых процедур с соответствующими параметрами. В процессе репликации участвуют агент моментальных снимков **Snapshot Agent**, агент распределения **Distribution Agent**, агент чтения журнала **Log Reader Agent**. **Snapshot Agent** используется для предоставления подписчикам начального набора данных. Log Reader Agent просматривает журнал транзакций на предмет наличия записей типа insert, update, delete, относящимся к опубликованным объектам, и осуществляет их пакетное применение к базе данных распределения. ***Репликация сведением (слиянием)*** – изменения в базе данных – источнике отслеживаются, а затем синхронизируются с базами данных издателя и подписчика. **Snapshot Agent** записывает структуру и данные базы данных подписчика и передает их агенту слияния (сведения). **Merge Agent**, который обеспечивает передачу начальных данных от Snapshot Agent, и согласовывает изменения между издателем и подписчиком.

# ГЛАВА 7. СРЕДСТВА РЕАЛИЗАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ

## 7.1. Технологии доступа к данным

*Технологией доступа к данным* называется система интерфейсов, обеспечивающая взаимодействие между приложением и базой данных.

Фирмы, разрабатывающие программные продукты, обеспечивающие возможность создания приложений, создают и свою линейку технологий доступа к данным, хранение которых обеспечивается определенными СУБД. Так, например фирма Microsoft, разработавшая такие языки программирования, как C, C#,C++ и Субд MS SQL Server, Access, реализовала технологии доступа ODBC,OLE DB,ADO,ADO.Net. Фирма Borland разработавшая такие языки программирования, как Object Pascal, Delphi и Субд IBX, Paradox, использует технологии доступа ODBC, BDE, dbExpress, dbGo, IBX. Фирма Oracle разработавшая язык программирования Java и СУБД Oracle, использует технологии доступа ODBC, JDBC.

Технология ODBC (Open Database connectivity) – открытый интерфейс доступа к базам данных. Это стандарт, разработанный фирмой Microsoft, как интерфейс для доступа к реляционным базам данных и базам данных с табличной организацией. Архитектура ODBC включает диспетчер драйверов, определяющий в приложении тип СУБД; драйвер СУБД, определяющий и передающий запросы СУБД; ядро СУБД, осуществляющую обработку запросов к базе данных. Уровни соответствия ODBC – описывают функции, доступные через API драйверу ODBC. Виды уровней соответствия ODBC – Базовый уровень – Core Api (соединение с источником данных, подготовка и выполнение запросов), первый уровень – Level 1 Api (возможности базового уровня, плюс информация о параметрах драйвера), второй уровень – level 2 Api (возможности первого уровня плюс обзор соединений и возможность обработки двунаправленных курсоров). Уровни соответствия SQL определяют вид операторов SQL. Различают следующие уровни соответствия языку SQL – минимальный синтаксис (Minimum SQL grammar) – работа с таблицами, ведение данных; Базовый синтаксис (Core SQL grammar) –минимальный уровень, плюс работа с видами, система разрешений; расширенный синтаксис (Extended SQL grammar) – базовый синтаксис, расширенные функции, пакетная обработка, работа с хранимыми процедурами и курсорами. Главный недостаток ODBC состоит в том, что он не предоставляет универсальности обработки наборов данных, драйверы во многом зависят от способа организации конкретной СУБД. Поэтому была предложена OLE DB – объектно-ориентированная технология разработки повторно-используемых программных компонентов.

OLE DB – (Object Linking and Embedding for Databases) – объектно-ориентированная технология связывания и встраивания объектов для наборов данных. Это низкоуровневый интерфейс, написанный на C++, обеспечивающий доступ к различным источникам – реляционным и нереляционным, содержащим текст, файлы электронной почты и другие объекты. Технология OLE DB определяет набор интерфейсов компонентной модели (COM-объекты). Основными компонентами OLE DB считаются потребители, провайдеры данных и провайдеры сервисов. *Потребителем OLE DB* может являться любое приложение, использующее интерфейсы OLE DB (это может быть прикладная программа базы данных, средство генерации отчетов, объектная модель ADO, средство разработки). *Провайдер данных* – это объект, владеющий

данными и связанный с ними. В этой технологии все провайдеры реализованы в виде виртуальных таблиц. Провайдер данных принимает запросы на доступ к данным от потребителя, выполняет выборку и обновление данных и результат передает потребителю. *Провайдер сервисов* сам не владеет данными, а реализует расширенные возможности по управлению транзакций, сортировки данных и может работать с набором данных напрямую или через провайдер данных. Основные объекты – Data Source, Session,Command, Row set.

Технология ADO (ActiveX Data Object) является высокоуровневой объектной надстройкой над OLE DB. Технология ADO может использоваться для работы с любым провайдером OLE DB. Технология доступа к данным ADO поддерживает работу с хранимыми процедурами с входными и выходными параметрами, работу с курсорами различных видов, поддерживает обработку нескольких наборов данных одновременно.

Технология Ado.Net наследует технологию ADO и позволяет взаимодействовать с базой данных автономно. Архитектуру ADO.Net можно разделить на две части – подключенную и автономную. Класс объектов Data Adapter является посредником между подключенной и автономной частями. Подключаемая часть – это набор объектов подключений для различных СУБД, а само подключение осуществляется через строку подключения с указанием всех параметров подключения. Data Adapter выполняет запросы автономно, используя объекты подключения. Имеется автономный контейнер табличных данных, не знающий о СУБД, то есть о поставщике, – это Data Set. Связь с базой данных осуществляет компонент Binding Source, его свойство Data Source  содержит источник данных. Отображение данных источника осуществляется через DataGridView.

### 7.2. Особенности разработки интерфейса информационных систем

Интерфейс информационных систем позволяет работать со справочными, оперативными данными, реализовывать операции по поиску и выборке данных (запросы к базе данных), формировать печатные документы на основе информации в базе данных (отчеты).

*Справочные данные (справочники)* – это данные, которые редко изменяются, но часто используются в оперативных данных, реализуют возможность выбора из множества значений при хранении данных. Виды справочников – *простые справочники* (не используют других справочников, имеют от одного до 5 полей), *сложные справочники* (используют информацию из простых справочников). На форме для ведения справочников не отображаются суррогатные ключи, которые являются счетчиками (автоинкрементное поле), доступ к которым не имеет пользователь. Каждая форма при работе с любыми справочниками должна иметь возможность ведения справочных данных – выполнения операций добавления, удаления, изменения и просмотра данных. Каждый справочник должен реализовать ограничение целостности на данные (первичного ключа и альтернативного ключа), ссылочную целостность. К справочникам из примера базы данных «Испытания изделий» относятся такие отношения, как Подразделение, Вид испытания, Вид изделия (простые справочники), Сотрудники (сложный справочник).

*Оперативные данные* – это данные, которые часто изменяются и по логике отражающие суть функционирования системы. К оперативным данным для информационной системы «Испытания изделий» относится учет испытаний, включая формирование комиссий и проведение

испытаний. Главный принцип интерфейсных решений при работе с оперативными данными – это не отображение таблиц, а отображение в удобном виде набора данных для выполнения работы с такими данными, включающими данные из нескольких связанных таблиц. Пользователю системы не важно, как организовано хранение данных, ему нужно удобное средство для управления и ведения данными. С точки зрения интерфейсных решений представление информации для ведения оперативных данных является самой сложной задачей для проектировщика.

*Запрос* – выборка данных из базы данных с указанием различных параметров для быстрого просмотра. В качестве примера запросов в информационной системы «Испытания изделий» можно указать вывод сведений о сотрудниках конкретной комиссии, вывод сведений о проведенных испытаниях на указанную дату, вывод сведений о сотрудниках подразделения. Каждая форма при работе с запросами должна предоставить пользователю возможность ввода или выбора определенных параметров.

*Отчеты* – печатные формы в виде документов определенной системы. Вопросы, связанные с генерированием отчетов, более подробно рассмотрены в пункте 7.3. Формы, позволяющие формировать отчеты, могут иметь средства для ввода параметров. В качестве примера отчетов в информационной системе «Испытания изделий» можно указать формирование итогового отчета по годам по проведенным испытаниям, по проведенным испытаниям за указанный период времени, оценки за которые не ниже введенного параметра.

Информационные системы должны обеспечивать разграничение прав доступа пользователей. Это реализуется, например, за счет указания логина пользователя и пароля. Чаще всего в информационных системах выделяют три группы пользователей:

- оператор (ведет справочники и оперативные данные, но не смотрит отчеты и запросы),
- руководитель (не ведет данные, а только смотрит отчеты и запросы),
- администратор (может все функции системы реализовывать, создает пользователей, работает с логинами и паролями). Разграничение прав доступа реализуется на уровне интерфейсных решений через создание выбираемых или не выбираемых пунктов меню и подменю.

Для всех пользователей рекомендуется показывать самую важную информацию сразу, без явного выбора пользователем действия, при запуске информационной системы. Например, для информационной системы «Испытания изделий», показывать информацию за прошедшую неделю от текущей даты сведения об испытаниях изделий с оценкой ниже 3.

### 7.3. Средства реализации отчетов в базах данных

***Отчёт*** *– печатный документ,* содержимое которого динамически формируется на основе информации, содержащейся в базе данных. Понятие отчёта и запроса в СУБД представляют собой разные объекты. Запрос – это результат операции просмотра и обработки данных в базе данных. Отчёт является текстовым или графическим представлением выбранных данных.*Документ* формируется для печати на листах бумаги различного формата (А4, А3 и т. д.), на клейкую ленту, на почтовые конверты. Составные части отчета – *заголовок, колонтитул, область данных, итоги.* Заголовок – текст в начале отчёта всего отчета. Колонтитул – текст, повторяющийся на каждой странице внизу или сверху листа (верхний или нижний колонтитул)**.** Область данных – область, в которой располагаются фактические данные из базы данных

(может быть область для данных, объединенных в группы)**.** Итоги **–** области, в которых расположены агрегатные данные (суммы, средние и т.д.). На рисунке 7.1 приведена классификация отчетов.

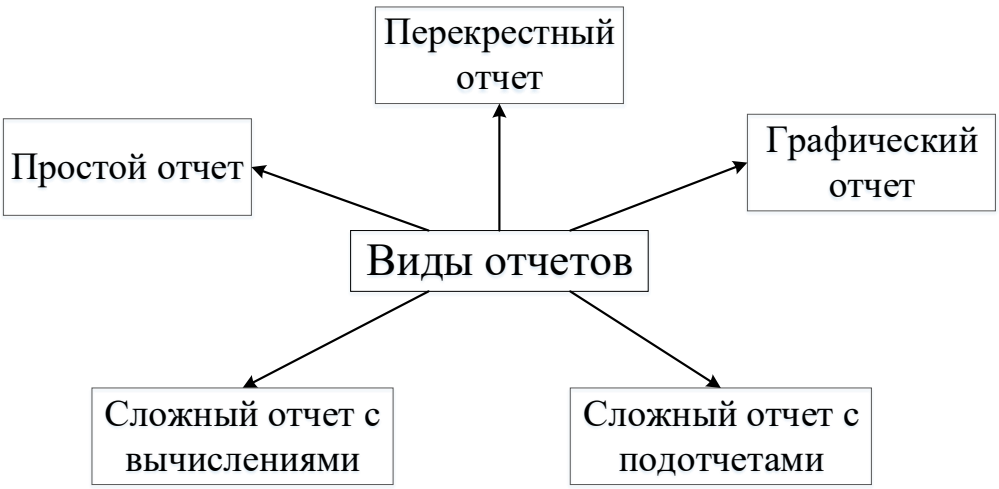



Рисунок 7.1 – Классификация отчетов

1. *Простые отчёты* – отчёты, создаваемые на основе одной или двух таблиц БД, не связанных между собой, а также отчёты на основе представления, не содержащие каких-то сложных вычислений, группировок, диаграмм, вложений.

2. *Сложные отчёты* – отчёты, созданные на основе двух или нескольких таблиц БД, связанных между собой связями различной мощности с вычислениями и группировкой.

3. *Графические отчёты* – отчёты, созданные на основе одной или нескольких таблиц БД. Данные в таком отчёте представляются в виде диаграммы, гистограммы или графика.

4. *Перекрёстный отчёт* – представляет собой таблицу с заранее неизвестным числом строк и столбцов. Значения из одного поля составляют столбцы таблицы отчёта, значения из другого поля – строки. Результирующим значением является пересечение строк и столбцов. Оно задается какой-либо функцией.

Современные средства генерации отчётов можно условно разделить на *две категории*: у*ниверсальные средства* генерации отчётов и в*строенные в СУБД или среды разработки средства* генерации отчётов. Оба вида поддерживают многие механизмы доступа к данным, совместимы с СУБД, содержат графические средства, интегрируются с офисными приложениями, позволяют публиковать отчёты в Интернете. На рисунке 7.2 приведена классификация средств генерации отчетов. Универсальных средств генерации отчётов поддерживают различные технологии доступа к данным; имеют развитые визуальные инструменты, высокоточные инструменты позиционирования элементов; средства деловой графики, а иногда простейшие встроенные геоинформационные системы; могут быть интегрированы с офисными приложениями; имеют встроенные языки программирования для создания формул; возможности создания аналитических отчётов. Генераторы отчётов встроенные в среду разработки поддерживают не все механизмы доступа к данным, они не имеют высокоточных инструментов позиционирования элементов отчёта, не поддерживают (или ограниченно поддерживают) экспорт отчёта в файлы других форматов.

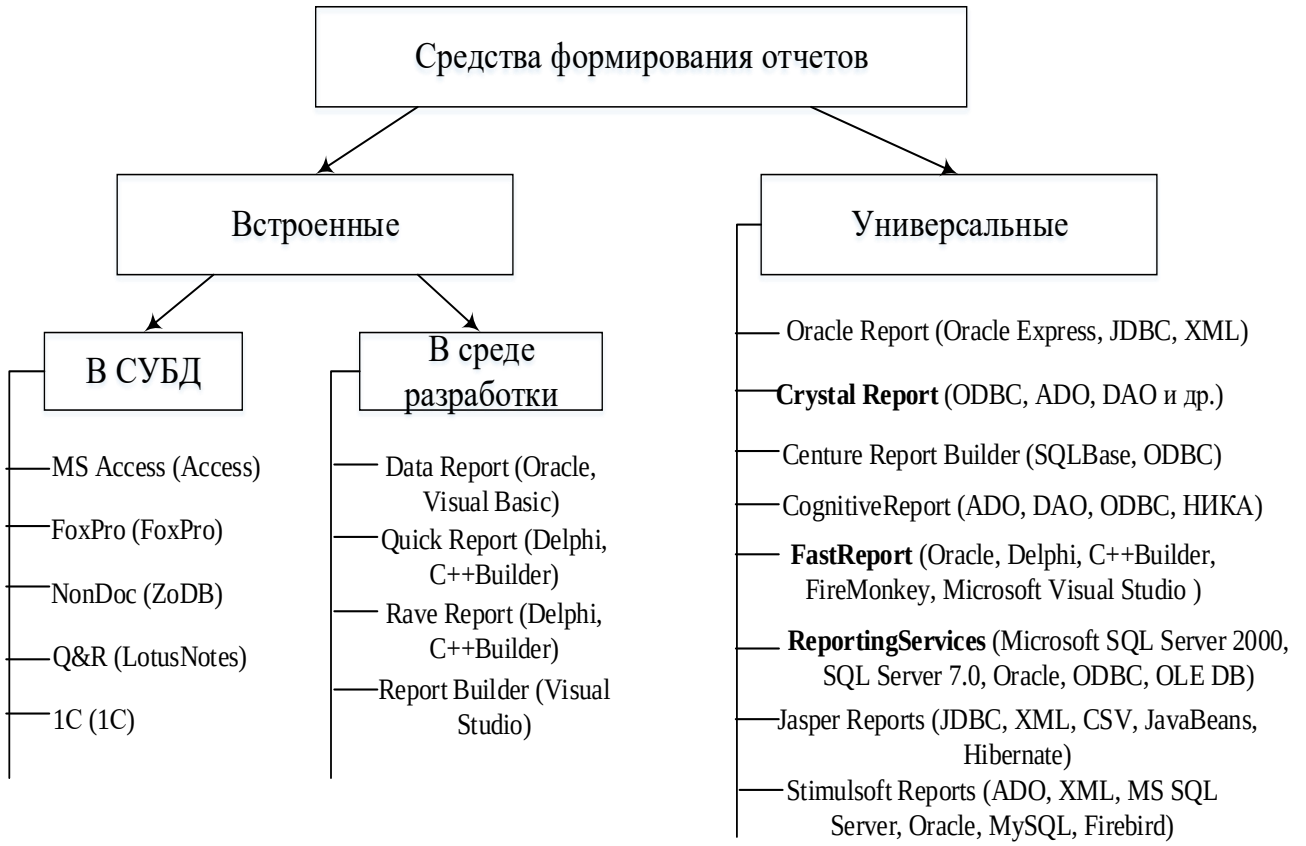



Рисунок 7.2 – Классификация средств генерации отчетов

Встроенные средства применяются, когда основными требованиями к печатному документу, является скорость и простота его получения, а не высокая точность печати или сложность алгоритма вычисления печатаемых данных. Встроенные генераторы отчётов не требуют отдельной установки. При разработке приложений на C# помимо встроенных генераторов отчётов можно использовать библиотеки Microsoft.Office.Interop.Word.dll и Microsoft.Office.Interop.Excel.dll. Они применяются при работе с документами word и excel. Аналогом для языка java является Apache POI – набор межплатформенных API для Java, предоставляющих доступ к чтению файлов и записи в них в форматах таких офисных приложений Microsoft Office, как Word, PowerPoint, Excel, Outlook, Visio и Publisher. Если БД содержит большие текстовые поля и их надо выводить в отчёт, то можно порекомендовать использовать Fast Reports. В Crystal Reports и Reporting Services имеется ограничение на длину текстового поля. Если нужно выводить большие объёмы текстовой информации в отчёт, то предпочтителен Crystal Reports. При наличии графики, лучшим вариантом будет Fast Reports. Генератор отчётов Crystal Reports при выводе графической информации работает медленнее. Если в отчёте присутствует много вычислений, то можно порекомендовать Reporting Services. Перекрёстные отчёты лучше всего генерируются с помощью Crystal Reports.

# ГЛАВА 8. ПРОЕКТИРОВАНИЕ ХРАНИЛИЩ ДАННЫХ

## 8.1. Понятие хранилища данных. Источники и приемники данных

*Хранилище данных (ХД)* – это предметно-ориентированная, интегрированная, неизменяемая и поддерживающая хронологию электронная коллекция данных для обеспечения процесса принятия решений. С точки зрения применения концепции *в бизнесе, производстве и технологиях* придерживаются следующего определения: ХД – структурно расширяемая вычислительная среда, спроектированная для анализа неизменяемых во времени данных, которые логически и физически преобразованы из различных источников, используемая для быстрого анализа. С точки зрения *реляционной алгебры хранилище данных* – это совокупность ненормализованных данных, создаваемая для реализации сложных аналитических запросов, а не для эффективного хранения или эффективного ведения данных. На рисунке 8.1 показаны источники и приемники информации в хранилищах данных.


Рисунок 8.1 – Источники и приемники данных в хранилищах данных

## 8.2. Концептуальное проектирование хранилищ данных

При концептуальном проектировании хранилищ данных используется многомерная модель. *Многомерная модель данных* – это многомерное представление информации при описании операций *манипулирования* данными. На рисунке 8.2 для предметной области, содержащей сведения о продажах автомобилей за текущий год, показано табличное представление. На рисунке 8.3 для той же предметной области показано многомерное представление тех же данных. Это представление похоже на перекрестную таблицу.

В процессе концептуального проектирования хранилищ не анализируются атрибуты, сущности и связи между ними, а идет анализ уже существующего способа организации хранения данных с точки зрения последующих операций манипулирования. Так для примера задачи

хранения информации о продажах автомобилей нужен анализ объема продаж в зависимости от модели автомобиля и месяца продажи.

| **Модель** | **Месяц** | **Объем** |
|---|---|---|
| Жигули | июнь | 10 |
| Жигули | июль | 12 |
| Жигули | август | 2 |
| Москвич | июнь | 3 |
| Москвич | июль | 13 |

Рисунок 8.2 – Табличное представление данных о продажах автомобилей

| | Июнь | Июль | Август |
|---|---|---|---|
| Жигули | 10 | 12 | 2 |
| Москвич | 3 | 13 | null |

Рисунок 8.3 – Многомерное представление данных о продажах автомобилей

В общем случае, многомерное моделирование проще, чем ER-моделирование. **Многомерное моделирование** – это метод моделирования и визуализации данных как множества числовых или лингвистических показателей или параметров, которые описывают общие аспекты, интересующие в рассматриваемой предметной области. Например, может интересовать число продаж, баланс, прибыль, вес – это числовые показатели, которые нужно определить. **Цель многомерного моделирования** не эффективное хранение и ведение данных, а выполнение операций манипулирования данных, чаще всего с подсчетами каких-то значений.

С точки зрения визуального представления, многомерную модель можно представить в виде **гиперкуба**, или **многомерного куба данных**, в ячейках которого хранятся анализируемые данные. К основным понятиям многомерного моделирования относятся, такие понятия как ***факты, атрибуты, измерения, элементы измерений, параметры (метрики), иерархия, гранулированность***. **Факт** (fact) – это *числовая величина, которая располагается в ячейках гиперкуба*. Каждый факт обычно представляет элемент данных, численно описывающий деятельность организации, бизнес-операцию или событие, которое может быть использовано для анализа деятельности организации или бизнес-процессов. **Атрибут** (Attribute) – это описание *характеристики реального объекта предметной области*. Обычно атрибут содержит заранее известное значение, характеризующее факт. Например, габариты упаковки товара. **Измерение** (dimension) – это *множество объектов одного или нескольких типов, организованных в виде иерархической структуры и обеспечивающих информационный контекст числового показателя*. Измерение принято визуализировать в виде ребра многомерного куба. Или, иначе, **измерение** – *это интерпретация факта с некоторой точки зрения в предметной области*. Измерение, как и атрибуты, содержат значения, которые по смыслу связаны между собой, это оси многомерного пространства, точками которого являются связанные с ним факты. Измерения задаются перечислением своих элементов. Объекты, совокупность которых и образует измерение, называются **элементами измерений** (members). Элементы измерений визуализируют как

точки или участи, откладываемые на осях гиперкуба. Элемент измерения имеет уникальный идентификатор, используемый для позиции измерения. *Например, измерение «Время» может содержать такие элементы, как «месяц», «квартал», «год».* Элементы измерения могут находиться в отношении *«часть-целое» или «родитель-потомок»*. Если используется отношение «родитель-потомок», то измерение может иметь одну или несколько **иерархий измерения**. Каждая иерархия может иметь несколько уровней. Каждый элемент иерархии может принадлежать только одному уровню иерархии, получая разбиение на непересекающиеся подмножества. Например, измерение «Время» имеет иерархии «Год» – «Полугодие» – «Квартал» – «Месяц». Элемент иерархии «Неделя» может принадлежать двум месяцам, поэтому для него нужна другая иерархия: «Неделя» – «День». С точки зрения **взаимосвязи измерений и фактов**, последние можно разбить на классы.

**Классы фактов** – **аддитивные факты** – используются с любым измерением при выполнении операций суммирования (например, объем продаж):

– **полуаддитивные факты** – используются совместно с некоторыми измерениями для суммирования (например, остаток на складе);

– **неаддитивные факты** – не имеет смысла использовать вместе с какими-либо измерениями для операции суммирования (например, измерение комнатной температуры);

– **числовые меры интенсивности** – является неаддитивным по времени, допускает агрегацию и суммирование по некоторому числу временных периодов (например, остаток на счете).

**Числовые характеристики фактов – Параметр, метрика или показатель** – это числовая характеристика факта, который определяет эффективность деятельности или бизнес – деятельности организации с точки зрения измерения. Например, факт – объем продаж товара. Его метрики – число проданных единиц и объем продаж в денежном выражении. **Гранулированность** – это уровень детализации данных, сохраняемых в хранилище данных. Например, ежедневные объемы продаж или ежечасные. В зависимости от этого на основе исходного куба могут строиться с учетом иерархии элементов измерения кубы с другими значениями элементов измерения. **Ячейка** (cell) – атомарная структура куба, соответствующая полному набору конкретных значений измерений. В ячейке хранится значение факта с учетом его метрики, которое зависит от набора измерений, на него влияющих.

В качестве примера рассмотрим многомерную модель для ХД «Торговое предприятие». В качестве факта выделим аддитивный факт – объем продаж товаров, на значение которого влияют такие измерения, как «Товар», «Место», «Время». При проектировании этой многомерной модели выделены такие возможные иерархии элементов измерений: 1) Для измерения «Товар»: Производитель – Категория товара – Товар (модель); 2) Для измерения «Место»: Федеральный округ-Город-Магазин; 3) Для измерения «Время»: Год-Квартал-Месяц или Неделя-День. Иерархия измерений задает путь суммирования объема продаж.

После построения куба можно быстро и эффективно выполнять операции манипулирования данными (поиск, запрос, фильтрация). С точки зрения многомерной модели о**сновными операциями над кубом** являются *развертка и свертка, срез, вращение*. **Развертка** – перемещение вниз по уровням иерархии измерения. **Свертка** – перемещение вверх по уровням иерархии измерения. При развертке от верхних уровней к нижним уровням происходит детализация

данных, при свертке от нижних уровней к верхним уровням происходит обобщение информации. Гранулированность гиперкуба имеет значения для измерения «Время» - Квартал, измерения «Товары» - Категория товара, измерения «Место» - Федеральный округ-Город. На рисунке 8.4 показана иерархия измерений куба ХД «Торговое предприятие».

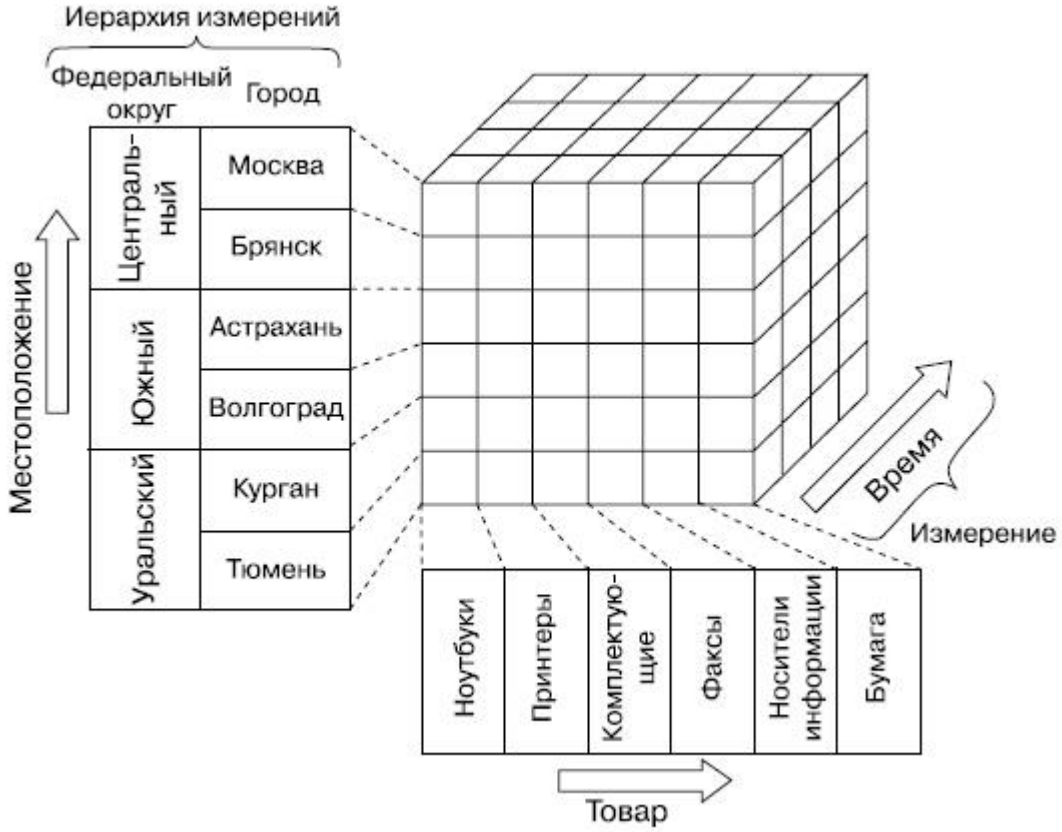



Рисунок 8.4 – Иерархия измерений куба ХД «Торговое предприятие»

Пример свертки куба по измерению «Федеральный округ» показан на рисунке 8.5. Уровень детализации (гранулированности) понизился, так как было удалено отображение городов по оси «Местоположение».

**Срез** – это операция выделения одного или нескольких измерений куба, в результате чего получается куб меньшего размера. После операции среза уменьшается размерность куба. Можно выполнить операцию среза куба по измерению время (квартал). В результате получен срез данных, например за второй квартал (Q2) (см. рисунок 8.6).

**Вращение** – это поворот куба. Операция вращения позволяет выполнить просмотр данных с разных точек зрения.

### 8.3. Логическое проектирование реляционных хранилищ данных

На *логическом уровне представления данных* (для реляционных баз данных) используется **размерная модель.** Размерная модель представляется в двух видах – в виде схема звезда и схема снежинка.

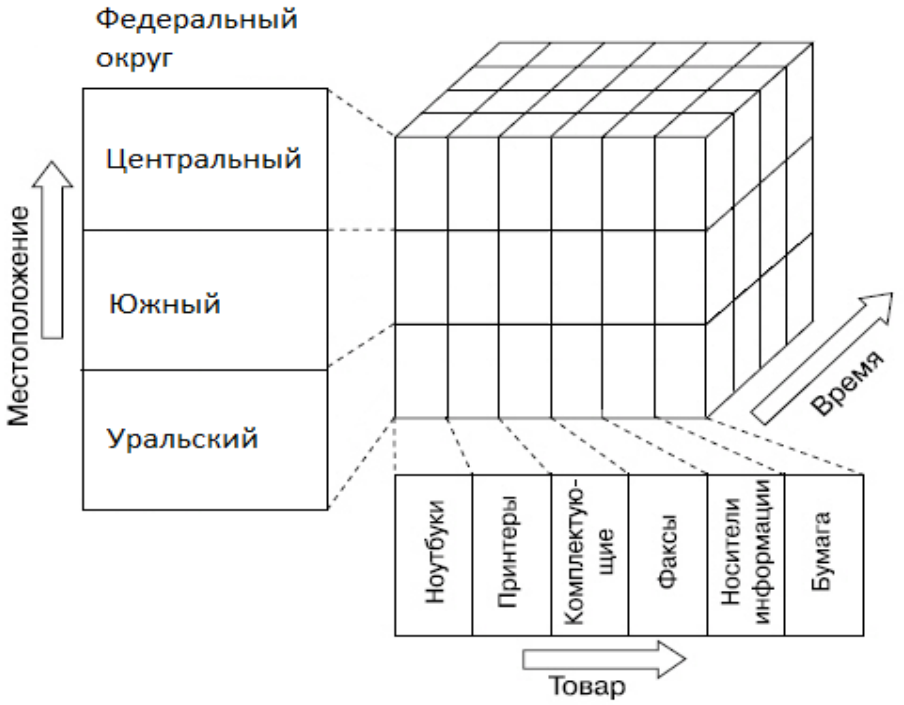



Рисунок 8.5 – Свертка куба ХД «Торговое предприятие» по измерению «Федеральный округ»

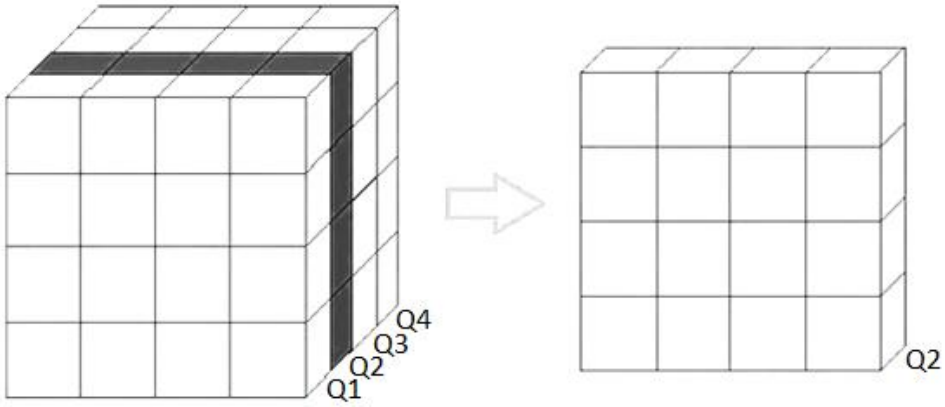



Рисунок 8.6 – Срез OLAP-куба

Размерная модель соответствует многомерной модели хранилища данных и с учетом терминов этой модели использует такие понятия, как **таблица фактов, таблица размерности** (или таблица измерений). Размерную модель можно создать, используя CASE – средства, например Erwin (Dimensional Model). *Таблица факта* на размерной модели всегда находится в центре и имеет множество полей являющихся фактами, определенными на этапе многомерного моделирования. Таблицы фактов иногда делят на три основные категории в зависимости от уровня детализации фактов. На размерной модели, реализующей хранилище данных, факты находятся в таблице фактов. Различают следующие виды таблиц фактов – **транзакционная таблица фактов, таблица фактов периодических моментальных снимков, таблица фактов коммулятивных моментальных снимков.**

В транзакционной таблице фактов сохраняются факты, которые фиксируют отдельные события (транзакции). Например, продажа товаров. В таблице фактов периодических момен-

тальных снимков хранится текущее состояние определенного представления или измерения. Например, продажи на текущую дату. В таблице фактов коммулятивных моментальных снимков собирают факты, фиксирующие некоторое итоговое состояние направления в бизнесе на текущий момент. Например, продажи текущего года на определенную дату. Первичный ключ в таблице фактов любого из перечисленных видов является, как правило, составным первичным ключом. Он состоит из множества внешних ключей, которые служат первичными ключами таблиц измерений, связанных с таблицей факта.

В общем виде схема звезда размерной модели показана на рисунке 8.7.

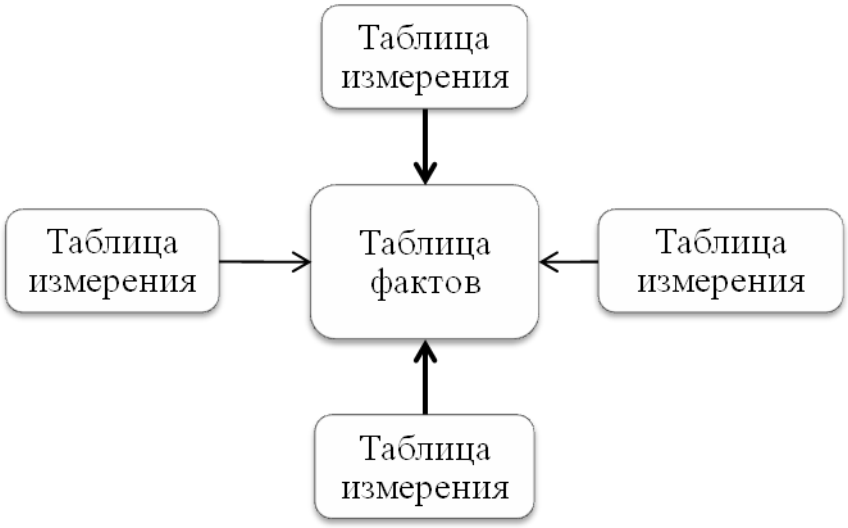



Рисунок 8.7 – Общий вид схемы звезда размерной модели

Преимущества схемы типа звезда: из-за денормализации таблиц упрощается восприятие структуры данных пользователем и формулировка запросов, уменьшается количество операций соединения таблиц при обработке запросов. Промышленные СУБД и инструменты класса OLAP / Reporting используют схему "звезда" для сокращения времени выполнения запросов. Недостатки схемы типа звезда: денормализация таблиц вносит избыточность данных, возрастает требуемый для их хранения объем памяти, поэтому происходит медленная обработка измерений. Далее описаны особенности схемы звезда размерной модели в нотации IDEF1X. Таблица фактов (fact table) денормализована и является центральной в схеме. Несколько таблиц измерений (dimensional table) связаны с таблицей факта. Таблицы размерностей содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации. Таблица фактов и таблицы размерности связаны идентифицирующими связями, при этом первичные ключи таблицы размерности мигрируют в таблицу фактов в качестве внешних ключей. Первичный ключ таблицы факта целиком состоит из первичных ключей всех таблиц размерности; агрегированные данные хранятся совместно с исходными данными.

На рисунке 8.8 приведено описание размерной модели в IDEF1X по схеме «Звезда» для многомерной модели торгового предприятия.


Рисунок 8.8 – Описание размерной модели в IDEF1X по схеме «Звезда» для многомерной модели торгового предприятия

Схема снежинка кроме таблицы факта и размерной таблицы (таблица измерения) включает консольные таблицы. На рисунке 8.9 показан общий вид схемы снежинка.

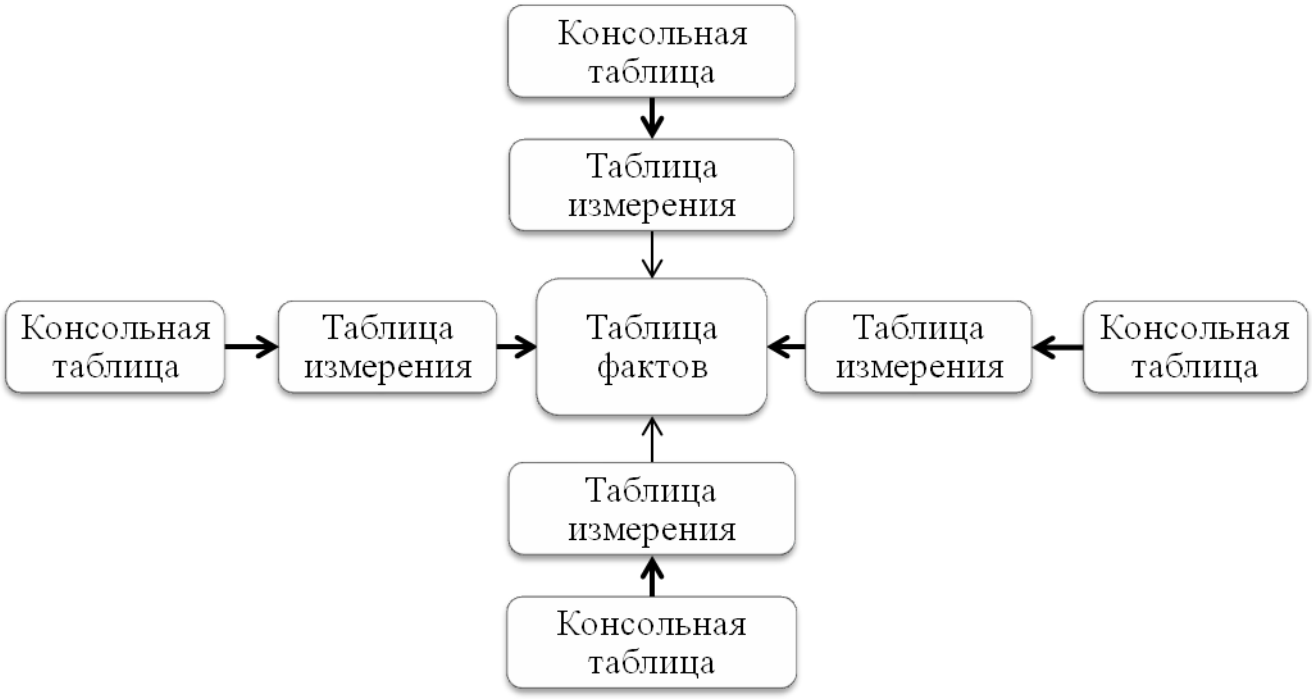



Рисунок 8.9 – Общий вид схемы снежинка размерной модели

Причем количество консольных таблиц, связанных таблицей размерности не обязательно единица. Консольных таблиц может не быть для некоторых таблиц размерности, а с некоторыми таблицами размерности на схеме снежинка может быть связано несколько консольных таблиц. В схеме снежинка уменьшена избыточность данных в таблице факта, что существенно повышает гранулированность процесса анализа, но появляется нормализация, а значит, замедляется процесс выполнения запросов. В схеме "снежинка" агрегированные данные могут храниться отдельно от исходных данных. Таблица фактов, которая денормализована, является центральной в схеме, может состоять из миллионов строк и содержать суммируемые или фактические данные, с помощью которых можно ответить на различные вопросы.

Ниже рассмотрены особенности схемы типа "снежинка" в нотации IDEF1X. Таблица факта (fact table), находящаяся в центре, связана с таблицами размерности (dimensional table) идентифицирующими связями и является всегда дочерней в этих связях. Консольные таблицы (outrigger table), присоединенные к таблицам измерений, являются главными (родительскими) по отношению к таблицам измерений и связаны с ними либо неидентифицирующими, либо идентифицирующими связями.

На рисунке 8.10 приведен пример размерной модели в  нотации IDEF1X по схеме «Снежинка» для многомерной модели торгового предприятия.


Рисунок 8.10 – Описание размерной модели в нотации IDEF1X по схеме «Снежинка» для многомерной модели торгового предприятия

### 8.4. Физическое проектирование реляционных хранилищ данных

Каждая реляционная СУБД имеет средства создания хранилищ данных. Чаще всего это специальные утилиты, службы, позволяющие на основе созданной базы данных построить гиперкуб, обновлять его и работать с ним (например, в MS SQL Server, это **MS SQL Server Analysis Services**), реализуя различные операции манипулирования данными в кубе. В СУБД ORACLE кроме такой службы, имеются языковые средства, позволяющие создать и работать с хранилищем данных с помощью материализованных видов.

**Материализованное представление** это представление, которое **физически существует** в базе данных. В это представление могут быть включены соединения и/или составные значения (агрегаты). В качестве источника для материализованного представления могут служить таблицы, представления, а также другие материализованные представления. Исходные таблицы называются главными таблицами (master tables), а в хранилище данных их часто также называют таблицами измерений. Таким образом, работу с материализованным видом можно представить как создание таблицы факта, на которую ссылаются таблицы в базе данных, играющие роль таблиц размерностей.

Синтаксис команды для создания материализованного вида выглядит следующим образом:

CREATE MATERIALIZED VIEW <имя вида> <опции> AS <команда Select>

Основные опции оператора *CREATE MATERIALIZED VIEW*.

- *BUILD IMMEDIATE* немедленно наполняет материализованное представление; эта опция принята по умолчанию. Альтернатива заключается в использовании опции *BUILD DEFERRED*, которая в действительности загружает материализованное представление данными позднее, в указанное время.
- *REFRESG FAST* специфицирует, что материализованное представление должно использовать метод обновления *FAST*, что для фиксации всех изменений главных таблиц требует наличия двух журналов материализованных представлений, которые были созданы на предыдущем шаге. Часть *COMMIT* конструкции *REFRESH* указывает на то, что все зафиксированные изменения главных таблиц распространялись на материализованное представление немедленно после фиксации этих изменений.
- *ENABLE QUERY REWRITE* означает, что оптимизатор Oracle прозрачно перепишет все запросы для использования материализованных представлений вместо лежащих в основе главных таблиц. Для обновления всех материализованных видов можно воспользоваться командой: DBMS_MVIEW.REFRESH_ALL_MVIEWS.

## ЗАКЛЮЧЕНИЕ

Курс «Проектирование баз данных интегрированных информационных систем» опирается на сведения, полученные при изучении курса «Базы данных», рассматривает особенности проектирования баз данных и хранилищ данных, а также особенности проектирования систем баз. Обсуждаемые в данном учебном пособии вопросы являются частью общей информационной культуры и могут быть использованы для дальнейшего изучения современных баз данных.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Дейт К.Дж. Введение в системы баз данных. 6-е издание; пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 1999. – 848 с.

2. Конноли Томас, Бегг Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е изд.; пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1440 с.

3. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: учебник для высших учебных заведений / Под редакцией проф. А.Д. Хомоненко. – СПб.: КОРОНА принт, 2000. – 416 с.

4. Гэри Хансен, Джеймс Хансен. Базы данных: разработка и управление: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 1999. – 704 с.

5. Клайн Кевин. SQL. Справочник. 3-е изд.; пер. с англ. – М.: КУДИЦ-ОБРАЗ, 2010. – 832 с.

6. Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite. – М.: ДИАЛОГ-МИФИ, 2003 – 432 с.

7. Гарбуз Дж., Паскузи Д., Чанг Э. Database Design on SQL Server 7. Сертификационный экзамен – экстерном (экзамен 70-029). – СПб.: Издательство «Питер», 2000. – 560 с.

8. Гарбуз Дж., Паскузи Д., Чанг Э. Administering SQL Server 7. Сертификационный экзамен – экстерном (экзамен 70-028). – СПб.: Издательство «Питер», 2000. – 560 с.

9. Чигарина Е.И. Базы данных [Электронный ресурс]: [учеб. пособие по направлению подгот. бакалавров 230100.62 Информатика и вычисл. техника]; М-во образования и науки Рос. Федерации, Сам. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т)(СГАУ). – Самара: Изд-во СГАУ, 2015. – on-line.

Учебное издание

***Чигарина Елена Ивановна***

**ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ ИНТЕГРИРОВАННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ**

*Учебное пособие*

Редакционно-издательская обработка
издательства Самарского университета

Подписано в печать 21.05.2025. Формат 60×84
1/16. Бумага офсетная. Печ. л. 8,5.
Тираж 27 экз. Заказ № . Арт. – 15(Р1УП)/2025.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

---

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.