# 1 БЛОК

**Система баз данных (информационная система)** - это компьютеризированная система хранения данных, основная цель которой - содержать информацию и предоставлять её по требованию.

**Система управления базами данных (СУБД)** - программное обеспечение, предназначенное для использования и (или) модификации этих данных одним или несколькими лицами.

**Назначение СУБД**

1. обеспечить пользователя инструментарием, позволяющим оперировать данными в терминах, не связанных с особенностями их хранения в ЭВМ. В этом смысле СУБД действует как интерпретатор языка высокого уровня, предоставляя возможность описать данные и их обработку;

2. обеспечить секретность и разграничение прав доступа к информации;

3. защита целостности и непротиворечивость данных. Например, контроль, что число проданных билетов не превышало числа мест в самолете;

4. синхронизация доступа к информации при одновременном обращении нескольких пользователей (проблема многопользовательского доступа). Например, исключение возможности продажи двух билетов на одно и то же место в транспорте;

5. защита от отказов и восстановления состояния базы данных после отказа. При этом под отказами подразумеваются отказы оборудования, ошибки в работе программного обеспечения, технические ошибки персонала и т.д

**\*Основные компоненты системы баз данных.**

**Данные.** Различают 2 типа: однопользовательские и многопользовательские СУБД.

Основная задача многопользовательской системы обеспечить работу пользователю как в однопользовательской системе.

Мы будем рассматривать данные только в многопользовательских системах.

Данные в системе БД должны быть интегрированными и общими.

Данные в многопользовательских СУБД:

1. Интегрированные данные: подразумевают возможность представлять БД как объединение нескольких файлов данных, полностью или частично не перекрывающихся.
2. Общие данные: подразумевают возможность использования некоторых областей данных в БД несколькими пользователями одновременно.
3. Условно постоянные данные: данные, хранящиеся в БД. «Постоянные» - по отношению к другим данным: промежуточным, входным, выходным.
4. Входные данные – это информация, передаваемая системе (обычно с терминала или рабочей станции). Такая информация может стать причиной изменения постоянных данных.
5. Выходные данные – это сообщения и результаты, выдаваемые системой: на печать, отображаемые на дисплее, или записываемые на диски).

На больших предприятиях в настоящее время все чаще используются два вида БД:

1. операционная БД - для поддержания повседневной работы предприятия;
2. база данных, содержащая отчетную информацию: данные для поддержания принятия решений по управлению предприятием. Эти данные периодически обновляются (раз в день, раз в неделю и т.д.), получая информацию из операционной БД.

**Аппаратное обеспечение:**

Накопители.

Сетевое оборудование.

Процессор.

Оперативная память.

**Программное обеспечение:**

СУБД.

Утилиты – программы специального назначения.

Средства разработки приложений (программ для поддержки работы пользователей)

Средства проектирования БД.

Генераторы отчетов .

**Пользователи:**

Прикладные программисты – пользователи, которые отвечают за написания прикладных программ (приложений), использующих БД .

Конечные пользователи – пользователи, которые работают с базой данных через рабочую станцию (терминал). Конечный пользователь получает доступ к БД через приложения или используя интегрированный интерфейс СУБД. При этом, часто используется интерфейс, основанный на меню и различных формах, что облегчает его работу.

Администраторы базы данных организуют и отвечают за работу с БД.

**\*Уровни абстракции в СУБД**

Абстрагирование – отбрасывание лишних элементов в моделях объектов реального мира с выделением основных объектов, их свойств и связей между ними.

Цель информационной системы – обработка данных об объектах реального мира.

В широком смысле БД – это совокупность сведений о конкретных объектах реального мира в какой–либо предметной области. Таким образом, база данных это информационная модель предметной области

Каждый конечный пользователь должен получать возможность взаимодействовать с информационной системой на понятном ему языке, в соответствии с его представлением о предметной области.

*Представление* каждого пользователя описывает явления и объекты из предметной области лишь с некоторой точностью, необходимой для его деятельности. Получается такое представление путем выделения основных, с точки зрения пользователя, явлений, объектов, свойств и отбрасыванием второстепенных.

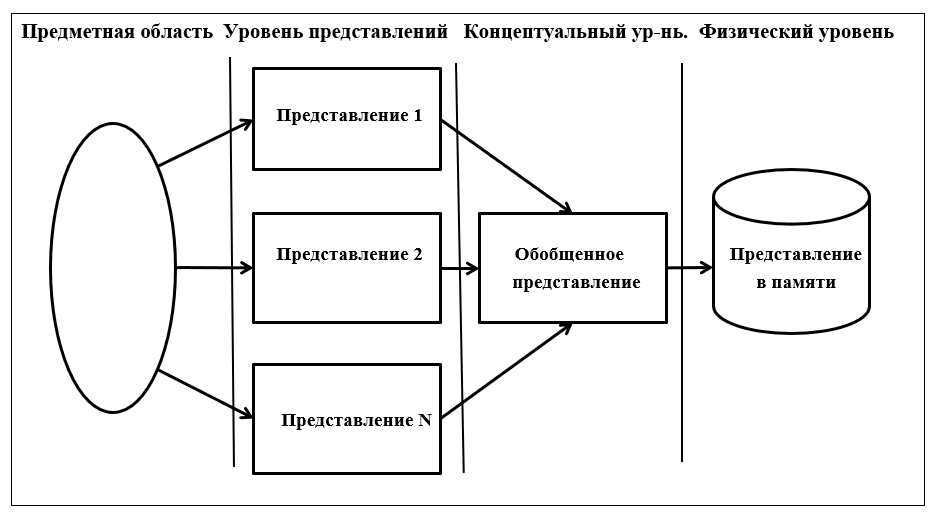
Процесс отвлечения от ряда свойств и связей изучаемого явления, с одновременным выделением интересующих исследователя свойств называется *абстрагированием*.

*Представление конечного пользователя* реализуется в информационной системе с помощью приложений. Т.к. обычно имеется несколько пользователей информационной системы, то в БД должны быть реализованы несколько представлений.

Образы объектов и явлений реального мира, выделенные на основании представлений пользователей, записываются в памяти ЭВМ в цифровом виде. При этом, возникает задача разработать такую архитектуру баз данных, которая позволила бы весь период эксплуатации БД обеспечивать стабильное развитие и, при необходимости, простую модификацию баз данных, разрабатываемых с помощью различных СУБД.

Перед разработчиками архитектуры информационной системы стоит задача, аналогичная задаче стандартизации архитектуры компьютерных сетей. Решения этих задач также аналогичные – использовать многоуровневую архитектуру, что позволяет развивать и совершенствовать одни уровни БД достаточно независимо от других.

Одна из архитектур организации информационной системы была предложена группой ANSI/X3/SPARC Study Group, которая была организована в 1972 г. комитетом Standards Planning and Requirements Committee (SPARC) института American National Standards Institute on Computers and Information Processing (ANSI/X3). Эта архитектура имеет трехуровневую систему организации информационной системы.



Существуют три уровня абстракции в архитектуре ANSI/X3/SPARC базы данных:

* Уровень представлений (внешняя модель базы данных).
* Концептуальный уровень - концептуальная база данных (концептуальная (логическая) модель базы данных).
* Физический уровень - физическая база данных (физическая модель базы данных).

**Внешняя модель или уровень представлений.** На этом уровне предметная область (т.е., та область деятельности, в которой осуществляется разработка данной системы) описывается будущими пользователями БД. Каждый пользователь описывает свое представление о предметной области. При этом, описывается: какие объекты важны в работе пользователя, какими свойствами они обладают, связи между объектами, прочие правила взаимодействия объектов, структура и алгоритмы построения документов, которые пользователи должны получать при обработке информации из БД. При описании представлений используются языки моделирования бизнес процессов. Например, в нотации функционально-логического моделирования IDEF0. Одним из первых языков являются графические схемы ER-моделей, в основе которых лежит объектное представление предметной области.

**Концептуальный или логический уровень.** На этом уровне прикладными программистами разрабатывается обобщенное описание предметной области, которое опирается на представления пользователей. В логической модели используются один из формальных языков. Выбор языка определяется моделью используемой в СУБД. Это может быть, например:

язык, описывающий деревья (иерархическая модель);

язык сетей, как класса графов (сети, сетевая модель);

язык отношений (реляционная модель);

ER – модели, как вариант языка сетевой модели.

**Внутренний (физический уровень).** Описание модели (концептуальной) на языке некоторой СУБД

*Внешний уровень* обычно поддерживается приложениями, написанными возможно на различных языках программирования. Этот уровень связан со способами представления данных для различных пользователей.

*Концептуальный уровень* является обобщенным представлением всех пользователей. Может быть несколько внешних представлений, каждое из которых состоит из представления определенной части базы данных, но может быть только одно концептуальное представление, состоящее из абстрактного представления базы данных в целом.

Есть единственное *внутреннее представление*, отражающее базу данных как физически хранимую.

**Влияние моделей концептуального уровня в трех уровневой архитектуре СУБД на проектирование БД**

При использовании реляционной модели концептуальный уровень будет определенно реляционным. Внешнее представление, в этом случае, также будет реляционным или близким к нему. На внутреннем же уровне используются структуры, которые не должны быть связаны с используемой в СУБД моделью данных.

Разработка внешних представления и концептуальной модели является ядром проблемы проектирования базы данных. В процессе проектирования активно используются языки моделей данных СУБД и понятия из предметной области. Для реализации базы данных СУБД должна иметь средства как определения и поддержки базы данных на всех уровнях, так и развитые интерфейсы для обеспечения взаимодействия всех трех уровней между собой. Последние интерфейсы называют отображениями одного уровня архитектуры на другой.

**План БД. Схемы и экземпляры БД. Основные определения**

*План БД* - перечень типов объектов, относящихся к БД и связей между ними. Иногда план называют *схемой*. На стадии проектирования БД разработчики имеют дело с *планом БД*. На стадии эксплуатации пользователи имеют дело с содержащимися в базе данных *актуальными данными*. Данные в БД при эксплуатации часто изменяются. *Планы* меняются значительно реже.

В соответствии с 3-х уровневой моделью вводят понятие *концептуальная* и *физическая* схема.

Схемы уровня *представлений* – обычно являются частями (подсхемами) *концептуальной схемы*.

*Экземпляр БД* – это совокупность информации, содержащейся в БД в каждый момент времени.

Для описания схем и экземпляров используют понятия:

*Хранимое поле* – это наименьшая единица хранимых данных. БД может содержать много экземпляров одного из нескольких типов полей (ФИО, №Детали).

*Хранимая запись* – это набор связанных хранимых полей, рассматриваемых как одно целое. Экземпляр записи состоит из группы связанных экземпляров хранимых полей.

Пример: запись о детали (номер, название, вес, цвет и т.д.).

*Хранимый файл* – это набор записей всех экземпляров хранимых записей одного типа (предполагаем, что в файле хранятся записи только одного типа)

**Схемы уровня представлений.**

Схемы уровня представлений могут быть частями концептуальной схемы. В некоторых случаях схемы представления могут быть более “абстрактными” чем концептуальные.

Пример: для отдела кадров поддерживается представление, которое включает возраст каждого сотрудника. Поддерживать возраст на концептуальном уровне не целесообразно, т.к. изменять соответствующее хранимое поле необходимо ежедневно для всех сотрудников. Целесообразно хранить дату рождения, а возраст вычислять как разность между текущей датой и датой рождения

Для получения схем представлений чаще всего используют различные подмножества из концептуальной схемы.

Пример исключения. 2-й уровень (концептуальная схема): в БД авиарейсы могут рассматриваться как множество пассажиров. 1-й уровень (уровень представлений): в представлении службы продажи билетов пассажир может рассматриваться как некоторое множество рейсов, на которые у пассажира есть билеты. Т.е. представление и концептуальная схема совсем не обязательно должны определяться с помощью одной и той же модели данных

**Особенности ER-модели. Ее преимущества и недостатки.**

Модели БД типа объект/отношение (сущность-связь) могут быть использованы на всех уровнях абстракции.

Особенностью модели является её тесная логическая связь с предметной областью. Данная модель относится к так называемым семантическим (содержательным) моделям.

*Преимущество*: проще описывать условия целостности.

*Недостаток*: сложно проводить формальную оптимизацию информационной модели, т.к. тесная логическая связь с предметной областью не дает разработчику отвлечься от ее особенностей

**\*Схема описание представление предметной области с помощью ER-модели.**

Чтобы описать представление предметной области с помощью ER-модели необходимо задать множество объектов реального мира.

Объект – семантическое понятие, которое может быть полезно при обсуждении устройства реального мира. Нечто существенное, полезное для описания предметной области.

Объект это сущность реального мира.

Объекты – не обязательно материальными понятия. Важным является только существенность и различимость каждого объекта от других.

Пример объектов: сотрудник, пассажир, самолет, деталь, счет в банке и т.д

Различают тип объекта и экземпляр объекта

Пример: тип – сотрудник; экземпляр – Иванов.

Объект обладает рядом свойств, которые иногда называют атрибутами объекта.

Различают тип атрибута и значение атрибута.

В информационной модели объект представляется значениями своих атрибутов.

Совокупность значений атрибутов объекта является его информационной моделью.

Пример: объект — сотрудник, его атрибуты: ФИО, дата рождения

Атрибуты или множество атрибутов значения, которых уникальным образом идентифицируют экземпляр объекта, называют ключом.

Не существует двух различных объектов, у которых совпадают значения ключей. Т.к. все экземпляры объекта в реальной жизни различимы, должны быть различимы и экземпляры одного типа объектов. Тогда, любые два экземпляра объектов одного и того же типа должны различаться значениями хотя бы одного атрибута. Следовательно, каждый тип объекта должен иметь ключ.

Пример: объект студент, ключ – номер зачетной книжки

**Связи между объектами. Виды связей.**

Между объектами могут возникать связи

Пример: между объектами деталь и поставщик может возникнуть связь – деталь поставляется поставщиком (для каждого экземпляра поставщика ставится в соответствие список деталей, которые он поставляет).

Связь является отношением, т.е. подмножеством прямого произведения, но не обязательно это бинарное отношение (поставщик – детали).

Возможна связь (отношение) связывающая экземпляры одного и того же типа объектов. (Например, между экземплярами объекта деталь может существовать связь – список деталей, образующих данную деталь)

Виды связей на примере базы данных «Хирургическое отделение больницы».

Выделим четыре типа (класса) объектов:

Хирург , Пациент, Палата, Место\_в\_палате.

Между объектами могут возникать следующие связи:

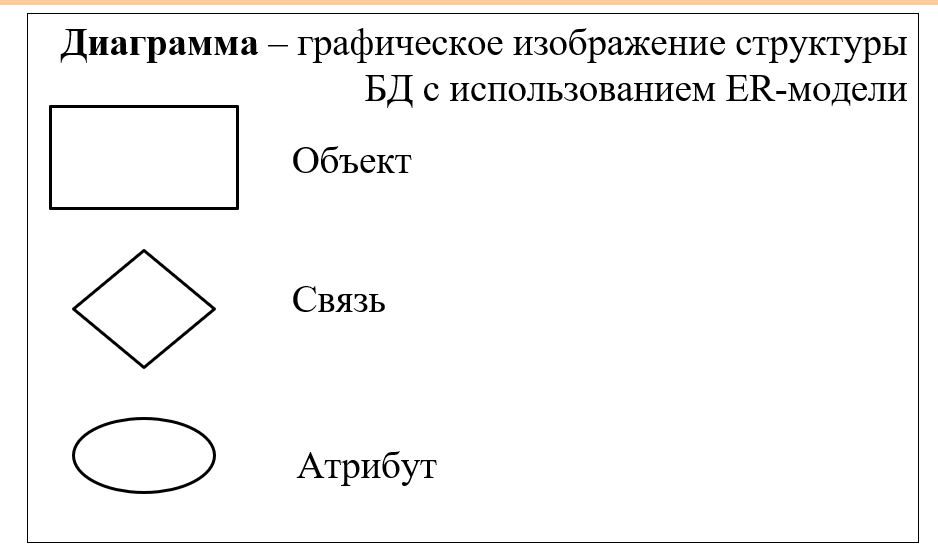
один к одному 1:1 (Пациент - Место\_в\_палате);

один к многим 1:n, или, как вариант - многие к одному n:1 (Палата - Место\_в\_палате);

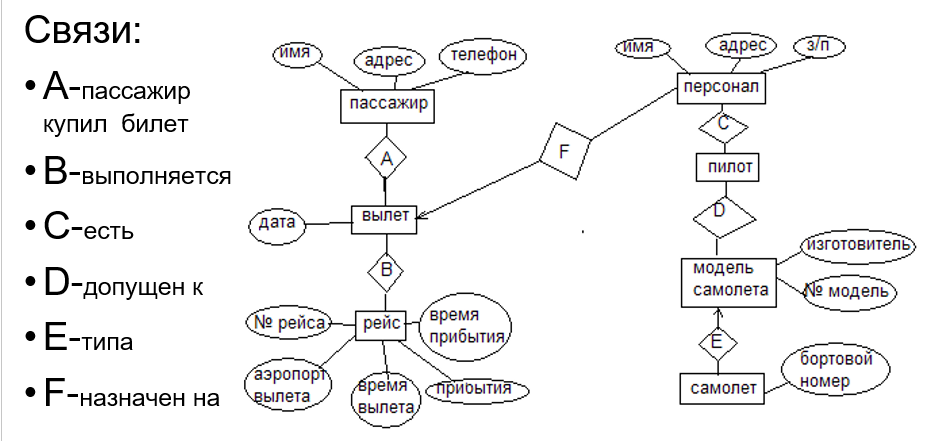
многие к многим n:n (Пациент - Хирург)

Связи могут иметь имена, например, 1. Назначено.

2. Содержит. 3. Операция.

**Диаграммы объектов / связей.** 

Пример диаграммы ER



**Влияние концептуальной схемы на внешнюю и физическую модели.**

Выбор концептуальной схемы является центральным пунктом в разработке баз данных. Концептуальные схемы оказывают существенное влияние на способы описания схем уровня представлений и физических схем.

Для определения базы данных на концептуальном уровне СУБД предоставляет так называемый язык определения данных (DDL data definition language).

Этот язык позволяет записать концептуальную схему в терминах некоторой модели данных.

В качестве модели данных может быть выбрана, например, ER-модель объектов/отношений. Основными понятиями этой модели являются понятия объект, атрибуты (свойства), отношения (связи).

Пример БД аэропорта является фактически концептуальной схемой БД в терминах ER-модели

**\*Модели, которые используются для описания концептуальных схем.**

**Иерархическая модель.** Математической моделью данных здесь является древовидный граф, т.е. связанный, ориентированный граф без циклов, у которого имеется только одна вершина, из которой дуги только выходят и ни одна дуга не входит. Эта вершина называется корнем дерева.

Вершина, в которую дуги только входят, называется лист. Все остальные вершины - корни.

Вершина для некоторой вершины А называется родительской, если из родительской вершины начинается дуга и заканчивается в А.

Братья это множество вершин имеющих одного родителя.

Типичным представителем СУБД, в которой реализована иерархическая модель, является СУБД IMS (INFORMATION MANGMENT SYSTEM) фирмы IBM, середина 60-х годов.

Эта СУБД официально считается вообще первой реализованной СУБД.

Появление этого типа СУБД объясняется простотой и наглядностью используемой модели, которая широко распространена в различных областях деятельности человека и в природе (государство, армия, предприятия, стая и т.д.).

Недостатком этой СУБД является сложность реализация связей n:n (многие к многим).

**Сетевая модель.** Математической моделью являются ориентированные графы, вершинами которых является множество однородных объектов, а дугами – ассоциации (связи).

ER-модель (объект/отношение) является примером сетевой модели.

Теоретические основы модели разработаны независимой ассоциацией программистов CODASYL, как обобщение иерархической модели и опыта эксплуатации информационных систем.

Пример СУБД сетевой структуры: CA-IDMS/BD, компания Computer Associates International Inc, (в СССР – ИНЕС).

**Реляционная модель.** Математическая модель базируется на теоретико – множественном понятии отношения. Используется аппарат математической логики и расширение теории множеств.

Модель разработана американским математиком Коддом (Codd), сотрудником фирмы IBM, и опубликована в 1970 г. В настоящее время является наиболее полно разработанной основой теории БД.

Примеры реляционных СУБД: ORACLE корпорация Oracle, DB/2 фирмы IBM, SYBASE компания Sybase Inc., MS-SQL фирмы MicroSoft.

К реляционным относятся свободно распространяемые СУБД — MySQL, PostgreSQL, SQLite.

**СУБД, опирающиеся на другие модели**: СУБД ADABAS, в которой используются так называемые инвертированные файлы (инвертированные списки).

**Постреляционные СУБД:**

1. дедуктивные СУБД;
2. экспертные СУБД
3. расширяемые СУБД;
4. объектно-ориентированные СУБД;
5. семантические СУБД;
6. универсальные реляционные СУБД;
7. многомерные СУБД;
8. NoSQL.

**\*Внутренняя (физическая) схема БД.**

Третьим уровнем архитектуры ANSI/SPARC является внутренний (физический) уровень, описывающий размещение БД на устройствах внешней памяти. Сама по себе физическая БД может быть представлена на нескольких уровнях абстракции от записей и файлов в языках программирования до уровня цилиндров, треков, секторов, байтов и бит, хранимых на внешних устройствах.

Физическая организация файлов определена файловой системой OS, под управлением которой работает СУБД, и рассматривается в дисциплине «Операционные системы».

Игнорирование при проектировании БД физической организации файлов может существенно снизить производительность и эффективность СУБД.

Представим физическую организацию файла на уровне некоторой модели (абстракции) реальной реализации, предполагая внешнюю память как бесконечное линейное адресное пространство.

Линейное в том смысле, что для указания адреса внешней памяти достаточно указать одно число (это соответствует понятию кластера файловой системы). На самом деле адресация памяти на дисках трехмерна: цилиндр, трек, сектор.

Абстрагируясь от реальной организации памяти на физическом уровне, будем рассматривать внутреннее представление на уровне, эквивалентном записям и файлам языка программирования.

Внутреннее представление, в таком случае, описывается внутренней схемой, которая определяет различные типы хранимых записей, способ представления хранимых полей, физическую последовательность хранимых записей, способы получения различных логических последовательностей записей

**\*Соотношение внутреннего и внешнего языка определения данных.**

Внешний уровень – это индивидуальный уровень пользователя. К пользователям относятся конечные пользователи и прикладные программисты. У каждого пользователя есть свой язык общения с БД:

* для конечного пользователя это, или специальный язык запросов, или язык специального назначения, возможно основанный на формах и меню, созданный прикладным программистом с учетом требований пользователя и поддерживаемый некоторыми приложениями;
* для прикладного программиста это, либо один из распространенных языков программирования, такой как С, либо специальный язык СУБД.

Как для конечного пользователя, так и для прикладного программиста используемые языки включают подъязыки данных, т.е. подмножество операторов всего языка, связанное только с объектами и операциями с БД. Подъязык встроен в базовый язык, который имеет ещё операторы, не связанные с БД (if, while и т.д.).

СУБД может поддерживать любое количество базовых языков и любое количество подъязыков данных. В настоящее время существует один язык, который поддерживается практически всеми СУБД, это язык SQL (Structured Query Language).

Для СУБД Oracle базовый язык PL/SQL. Для СУБД PostreSQL – PL/pgSQL

Базовый язык и включенный в него подъязык данных могут быть практически неразличимыми (например, SQL – как самостоятельный язык). Если они неразличимы или трудно различимы, их называют сильно связанными. Если они легко различаются, то говорят, что они слабо связанны.

СУБД с сильно связанными языками сложнее разработать, но существует тенденция продвижения разработчиков СУБД к сильно связанным системам.

Подъязык данных является комбинацией двух языков: языка определения данных (data definition language - DDL), который поддерживает объявление или определение объектов БД, и языка обработки данных (data manipulation lаnguage - DML), который поддерживает операции с такими объектами или их обработку.

Внешние и внутренние языки могут совпадать или не совпадать. Пример: для прикладного программиста – это использование языков С и SQL. Внутренняя схема описана на С, а запросы на SQL. В этом случае внутренний и внешний языки не совпадают.

Если внутренняя схема описана на SQL и запросы на SQL, то языки совпадают.

Для конечного пользователя совпадение языков крайне редко (продвинутые пользователи). Обычно внешний язык – язык меню, внутренний, или специальный язык (например, системы 1C), или SQL. Иногда продвинутые пользователи общаются с СУБД на языке SQL и, если внутренний язык также SQL, то это соответствует совпадению языков.

**Понятие независимости данных.**

Основное, качественное отличие СУБД от предшествующих систем обработки информации является независимость данных. Для предшествующих систем сведения об организации данных и способах доступа к ним были встроены в логику и код приложения. Если, по каким-то причинам, необходимо было изменить структуру записей или способ доступа к файлу с информацией, то для приложений, не использующих технологию СУБД, необходимо было переписать приложения (программы).

СУБД построены таким образом, чтобы обеспечивать развитие БД, не оказывая существенного влияния на приложения.

**\*Физическая независимость данных**

Три уровня абстракции в структуре СУБД, внешний – концептуальный – внутренний, обеспечивают два уровня независимости данных.

Первый уровень независимости данных обеспечивается отображением концептуального уровня на внутренний (физический) уровень и называется физической независимостью данных. Это отображение описывает, как концептуальные записи представлены на внутреннем уровне. Так как, отображение описывается в файле с информацией, то при изменении внутренней структуры хранимой базы, изменяется и отображение концептуальный – внутренний так, чтобы концептуальная схема осталась неизменной.

*Пример 1:* некоторое приложение работает с хранимым файлом, в котором описаны два поля «А» и «В». Если этот файл заменить на другой, в котором хранимые поля переставлены местами («В» и «А»), то это изменение структуры записи хранимого файла не приведет к нарушению работы данного приложения. При этом, работа приложения не зависит от расположения хранимого файла в компьютерной сети.

*Пример 2*: изменение размеров полей не приводит к нарушению работы ранее написанных приложений.

**\*Логическая независимость данных**

Второй вид независимости обеспечивается отображением внешний – концептуальный и называется логической независимостью данных. Отображение внешний – концептуальный определяет соответствие между некоторым внешним представлением и концептуальной схемой. По мере использования БД может потребоваться модификация концептуальной схемы, например, с целью включения информации о новых типах объектов или дополнительной информации об уже представленных в ней объектах.

*Пример:* требуется информация об уровнях шума и загрязнении атмосферы на авиарейсах компании. Если таких сведений в базе данных нет, то возможно включение в БД сведения о шуме и загрязнении каждого самолета, без перестройки уже разработанных приложений.

Некоторые модификации концептуальной схемы можно выполнить, не затрагивая существующих представлений. Другие модификации становятся возможными, если переопределить отображение схемы уровня представлений на концептуальную схему. При этом не требуется каких-либо изменений в существующих приложениях.

Единственный вид изменений в концептуальной схеме, который не может быть осуществлен путем переопределения соответствия между схемой представления и концептуальной схемой, это удаление информации из концептуальной схемы, соответствующей представленной во внешней схеме (схеме представлений). Такие изменения обычно требуют переработки приложений или отказ от некоторых программ.

**Администратор данных (системный аналитик)**

Для реализации большой БД важную роль играют два специалиста: администратор данных (системный аналитик) и администратор БД. Иногда функции этих специалистов совмещаются в одном лице.

*Администратор данных (системный аналитик)* – это специалист, отвечающий за стратегию и политику принятия решений, связанных с данными предприятия на уровне представлений. Он определяет, какие именно данные необходимо сохранять в БД. Определяет объекты, в которых заинтересовано предприятие, а также информацию, которую необходимо записывать об этих объектах. Этот процесс иногда называют логическим проектированием БД. При этом осуществляется тесное взаимодействие с конкретными пользователями и формализация схем уровня представлений.

**\*Администратор БД. Функций администратора БД при совмещении им функций системного аналитика.**

Для выполнения этой работы необходим специалист знакомый с предметной областью (или способный освоить любую предметную область) и одновременно знакомый с законами и принципами программирования. Часто этот специалист уже на уровне разработки представлений может дать совет по оптимизации работы предприятия и на западе называется системным аналитиком (человек способный проанализировать систему).

*Администратор БД* – это специалист, обеспечивающий необходимую техническую поддержку реализации логического проектирования.

Список функций администратора БД при совмещении им функций системного аналитика:

1. Создание схем представлений.
2. Определение концептуальной схемы с помощью концептуального языка определения данных. При этом используется одна из 3-х моделей концептуального уровня.
3. Определение внутренней схемы (физическое представление) с использованием внутреннего языка определения данных. Определение соответствия между внутренней и концептуальной схемами.
4. Взаимодействие с пользователями:

* предоставление пользователям полномочий на использование БД или её частей;
* помощь пользователям в разработке схем представлений;
* определение соответствия между схемами представления и концептуальной схемой;
* консультации прикладных программистов по разработке приложений;
* обеспечение технического образования пользователей, помощь в выявлении и решении возникающих проблем.

1. Определение правил безопасности и целостности. Т.е. описание условий непротиворечивости и ограничений в доступе к файлам БД.
2. Определение процедур резервного копирования и восстановления информации после аппаратных или программных сбоев.
3. Управление производительностью и реагирование на изменяющиеся требования:

* модификация реализации концептуальной схемы внутренней (физической), если сведения об использовании БД показывают, что другая организация была бы более эффективной;
* модификация концептуальной схемы с целью устранения недостатков первоначального проекта, или в связи с изменением требований предприятия.

**\*Функции СУБД**

1. Определение данных.

СУБД должна допускать определение данных (внешние схемы, концептуальную схему, внутреннюю схему, а также все связанные отношения). Описания должны быть произведены на некотором исходном языке и СУБД должна преобразовать эти описания в форматы соответствующих компонент БД (т.е. включать обработку синтаксиса языка определения данных).

2. Обработка данных.

СУБД должна обрабатывать запросы пользователей на выборку, изменение, удаление существующих данных в БД или на добавление новых данных. Т.е. СУБД должна включать в себя компонент процессора языка обработки данных.

Замечание: Запросы языка обработки данных бывают планируемые и непланируемые.

Планируемый запрос – это запрос, необходимость которого предусмотрена заранее. Администратор БД должен настроить физический проект БД таким образом, чтобы гарантировать достаточное быстродействие для таких запросов.

Непланируемый запрос – это специальный запрос, необходимость которого не была предусмотрена заранее. Оптимальная возможность реализации такого запроса может быть предусмотрена в физической реализации БД.

Получение наибольшей производительности для непланируемых запросов представляет собой одну из проблем.

Планируемые запросы обычно осуществляются из написанных заранее приложений, а непланируемые формируются интерактивно.

3. Безопасность и целостность данных.

СУБД должна контролировать пользовательские запросы и пресекать попытки нарушения правил безопасности и целостности, определенных Администратором Базы Данных.

4. Восстановление данных и дублирование.

СУБД, или другой связанный с ней программный компонент, должен осуществлять необходимый контроль над восстановлением данных и дублированием.

5. СУБД должна обеспечить функцию словаря данных.

Словарь данных является базой данных, но не пользовательской, а системной. Словарь данных содержит данные о данных (так называемые метаданные), т.е. определения других объектов системы. В словаре, например, могут храниться определения различных схем (внешней, концептуальной, внутренней), отображения схем, требования к целостности данных и другая информация.

6. Обеспечение производительности.

Очевидно, что СУБД должна выполнять все указанные функции с максимально возможной производительностью.

Можно сформулировать, что назначение СУБД в предоставлении пользовательского интерфейса с БД. Этот интерфейс можно рассматривать как границу в системе, ниже которой всё невидимо для пользователя.

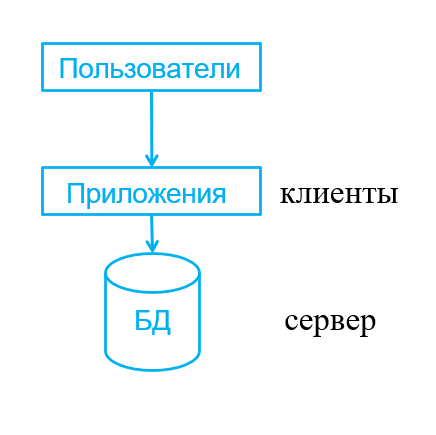
Следовательно, пользовательский интерфейс находится на внешнем уровне. Для подготовленных пользователей уровня прикладных программистов, внешнее представление может практически не отличается от относящейся к нему части концептуальной схемы.

**\*Архитектура клиент/сервер.**

С точки зрения основной цели создания СУБД (поддержка и разработка приложений БД, через которые осуществляется доступ к информации) Система Баз Данных (Информационная Система) состоит из двух частей:

непосредственно СУБД (сервер или машина БД);

внешний интерфейс (клиенты).



Сервер (машина БД) – это собственно СУБД, т.е. программное обеспечение. Для повышения производительности это программное обеспечение может быть реализовано на специальной ЭВМ. Сервер поддерживает все основные функции СУБД: определение данных, обработку данных, защиту и целостность и т.д. Обеспечивает поддержку на внешнем, концептуальном и внутреннем уровне. В данном контексте Сервер– это другое имя СУБД

Клиенты – это различные приложения, которые выполняются «над» СУБД

Типы приложений:

* приложения, написанные пользователями;
* приложения, поставляемые поставщиками СУБД или другими поставщиками программного обеспечения.

Общим является то, что все приложения используют один и тот же интерфейс сервера, а именно интерфейс внешнего уровня.

*Приложения написанные пользователем.* Это прикладные программы, написанные на одном из языков программирования (например: С, Java) или специализированном языке СУБД. В любом случае эти языки должны иметь интерфейс с подъязыком данных

*Приложения, поставляемые поставщиками* (часто называют инструментальными средствами). В целом назначение таких средств – содействовать процессу создания и выполнения других приложений. К таким инструментальным средствам относятся, прежде всего:

* процессоры языков запросов;
* генераторы отчетов, т.е. системы визуализации результатов построенных запросов;
* графические бизнес – системы;
* электронные таблицы;
* процессоры обычных языков программирования;
* средства управления копированием;
* визуальные дизайнеры приложений;
* другие средства разработки приложений, включая CASE–продукты (COMPUTER AIDED SOFTWARE ENGINEERING);
* системы автоматизация разработки программного обеспечения.

Количество и качество имеющихся клиентских инструментальных средств должно быть одним из основных факторов при выборе СУБД для проектирования Базы Данных.

**Распределенная обработка в архитектуре клиент/сервер**

Понятие распределенной обработки. Т.к. СУБД в целом может быть четко разделена на две части (сервер БД и клиенты), то появляется возможность работы этих двух частей на разных машинах, т.е. возможность распределенной обработки.

Распределенная обработка предполагает, что отдельные машины можно соединить какой-нибудь коммуникационной сетью, чтобы определенная задача, обрабатывающая данные, могла быть выполнена на нескольких машинах в сети.

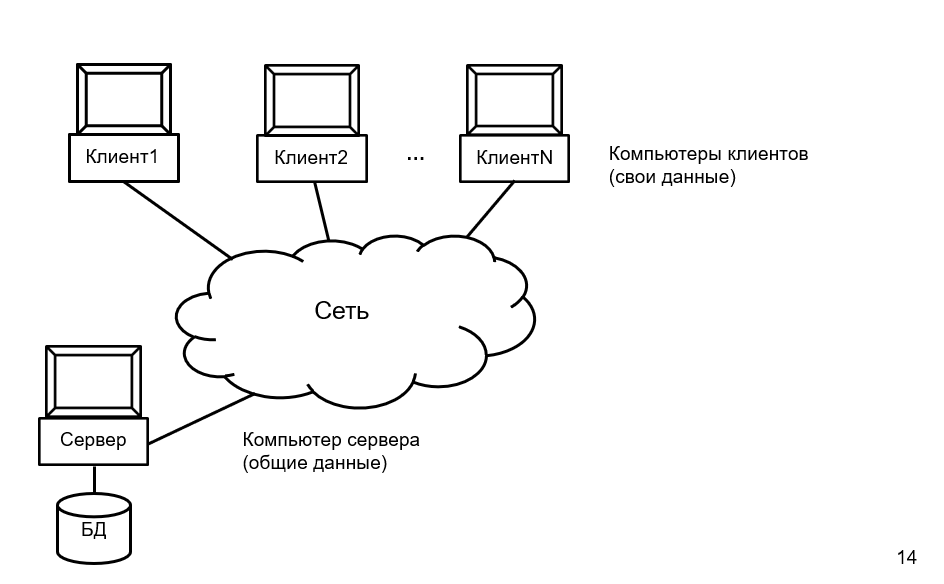
Термин клиент/сервер часто подразумевает только распределённую обработку, как дающую наибольшую выгоду, хотя такую технологию можно реализовать и на одной ЭВМ.

**\*Распределенная обработка в архитектуре клиент/сервер с одним сервером.**

Один из простых случаев распределенной обработки - это когда сервер СУБД запускается на одной машине, а клиентские приложения на другой.

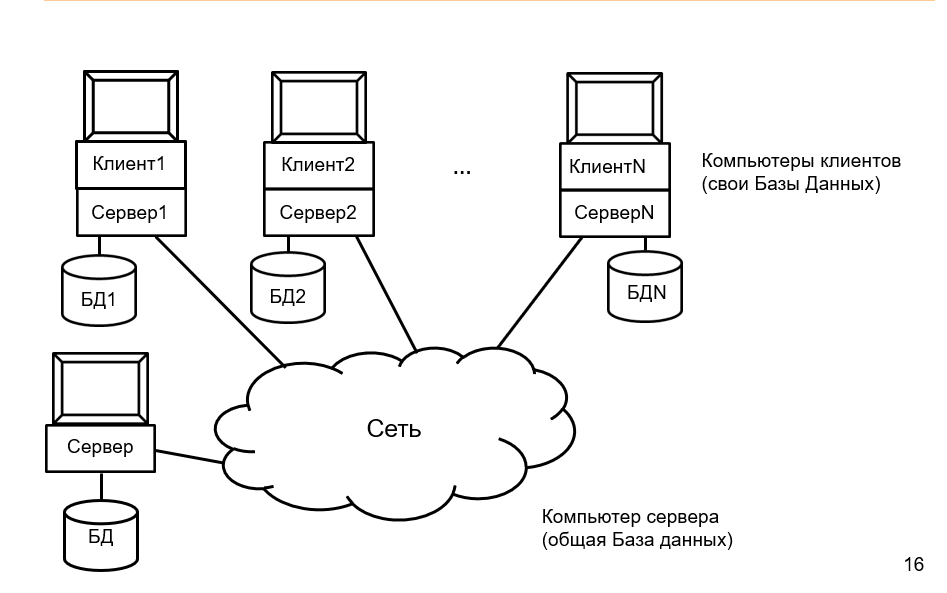
Преимущества такой схемы:

* параллельная обработка. Для всей задачи применяется несколько процессоров и обработка сервера (базы данных) и клиента (приложения) выполняются параллельно, время ответа уменьшается, производительность обработки растет
* машина сервера может быть изготовлена по специальному заказу, приспособлена для работы с СУБД, что обеспечит лучшую производительность;
* машина клиента может быть персональной станцией, приспособленной к потребностям конечного пользователя, обеспечивать лучший интерфейс и в целом дополнительные удобства;
* несколько разных машин клиентов могут иметь доступ к одной и той же машине сервера. Таким образом, одна БД может совместно использоваться несколькими отдельными клиентами системы. Такая структура соответствует структуре большинства предприятий.



**\*Распределенная обработка в архитектуре клиент/сервер с несколькими серверами.**

В некоторых случаях (например, для банка) весьма вероятно, что пользователям одного отделения банка необходим доступ к данным, сохраняемым в другом отделении. Таким образом, вообще говоря, каждая машина будет выступать в роли сервера для одних пользователей и в роли клиента для других. Иначе каждая машина будет поддерживать полную систему БД.



Распределение информации в архитектуре системы БД с несколькими серверами точнее описывает структуру предприятия. Доступ возможен двумя способами:

Клиент может получить доступ к любому числу серверов, но лишь к одному в одно, и тоже время. В этом случае нет возможности за один запрос получить комбинацию данных с двух и более серверов, кроме того, приложение должно иметь доступ к информации о том, на какой машине, какая часть данных содержится.

При втором способе клиент получает доступ к любому числу серверов одновременно (в запросе пользователя возможны комбинации серверов). В этом случае все серверы рассматриваются клиентом как один логический сервер. От пользователя скрыто на какой именно машине какая часть данных содержится. Такая реализация прозрачна для клиента, но сложна в реализации. Именно этот случай (последний рисунок) называется распределённой системой БД.

**\*Двухзвенная модель распределение функций в модели клиент/сервер.**

В системе БД выполняются три основные функции:

управление данными, обработка и представление данных пользователю.

Если эти три функции делятся между двумя ЭВМ, то говорят о двухзвенной модели клиент/сервер



*1-й вариант:* Распределенное представление – клиент вырожден. Клиентской приложение может быть реализовано на компьютере с процессором с ограниченным числом команд (так называемый, X-терминал).

Преимущество: простота сопровождения системы БД, т.к всё программное обеспечение, реализующее основные функции СУБД, расположено на ЭВМ сервера.

Недостатки:

* практически отсутствует распределение функций;
* уязвимость центрального компьютера.

*2-й вариант:* Удаленное представление. Процедуры обработки хранятся на сервере, представление на клиенте.

Преимущества:

* хорошее центральное обслуживание приложений (в одном месте правятся все функции обработки);
* достаточно эффективное использование ресурсов сети и сервера.

Недостатки:

* сложность выполнения обработки на сервере параллельно с доступом к данным;
* низкая эффективность использования ЭВМ клиента.

*3-й вариант:* Распределенная функция обработки. Часть процедур обработки хранятся на сервере, часть - на клиенте.

Преимущество:

* повышение эффективности использования ЭВМ клиента. Часть процедур обработки информации перенесены с ЭВМ сервера на ЭВМ клиентов.

Недостаток:

* сложность поддержки процедур обработки. При изменении процедур обработки клиентов, они должны быть обновлены на всех ЭВМ клиентов.

*4-й вариант*: Удаленный доступ к данным (по отношению к клиенту). Все процедуры управления данными хранятся на сервере. На клиенте установлены все процедуры обработки и представления.

Преимущества:

* управление данными и обработка полностью распределены;
* высокая эффективность использования клиента и сервера.

Недостатки:

* более высокая загрузка сети;
* сложность модификации и сопровождения (функции обработки установлены на всех клиентах).

*5-й вариант*: Распределенная БД. Возможны два варианта реализации:

1. локальные и удаленная БД рассматривается как целое;
2. локальные БД являются копиями удалённой БД (репликация БД). Клиенты работают со своими локальными БД и выполняют синхронизацию с удаленной.

Преимущества:

* гибкость создаваемой системы БД;
* высокая живучесть системы БД.

Недостатки:

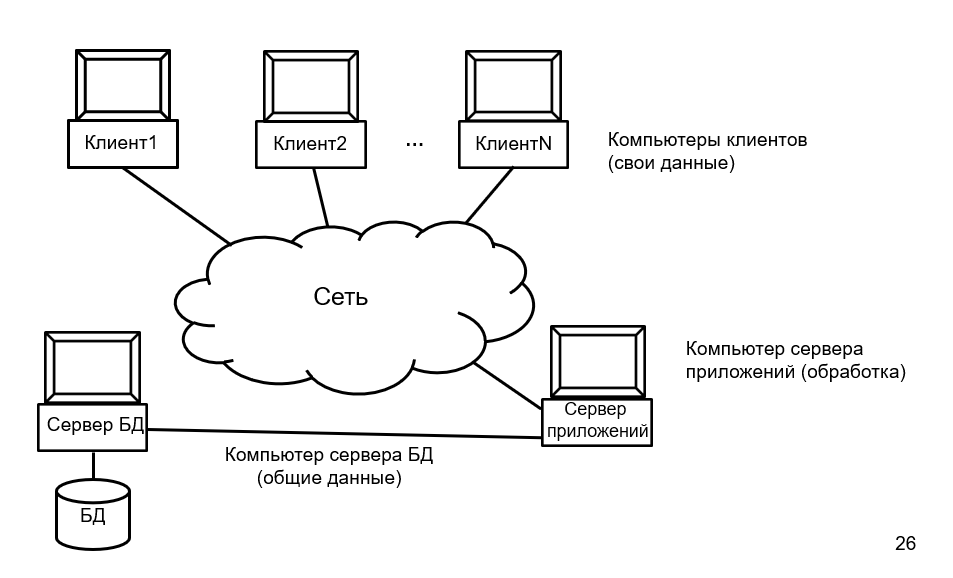
* сложность сопровождения клиентских приложений;
* нужен механизм синхронизации баз данных.

В пятом варианте распределения функций в системе БД, доступ к серверам возможен 2-мя способами:

клиент может получить доступ к любому числу серверов, но лишь к 1-му в одно и то же время. В этом случае нет возможности за один запрос получить комбинацию данных с 2-х и более серверов;

клиент получает доступ к любому числу серверов одновременно. Возможна комбинация информации. Клиенты рассматривают все серверы как единый логический сервер. Доступ прозрачен для клиента. Именно этот вариант называется *Распределенная БД*

**\*Понятие о трехзвенной архитектуре модели клиент/сервер.**



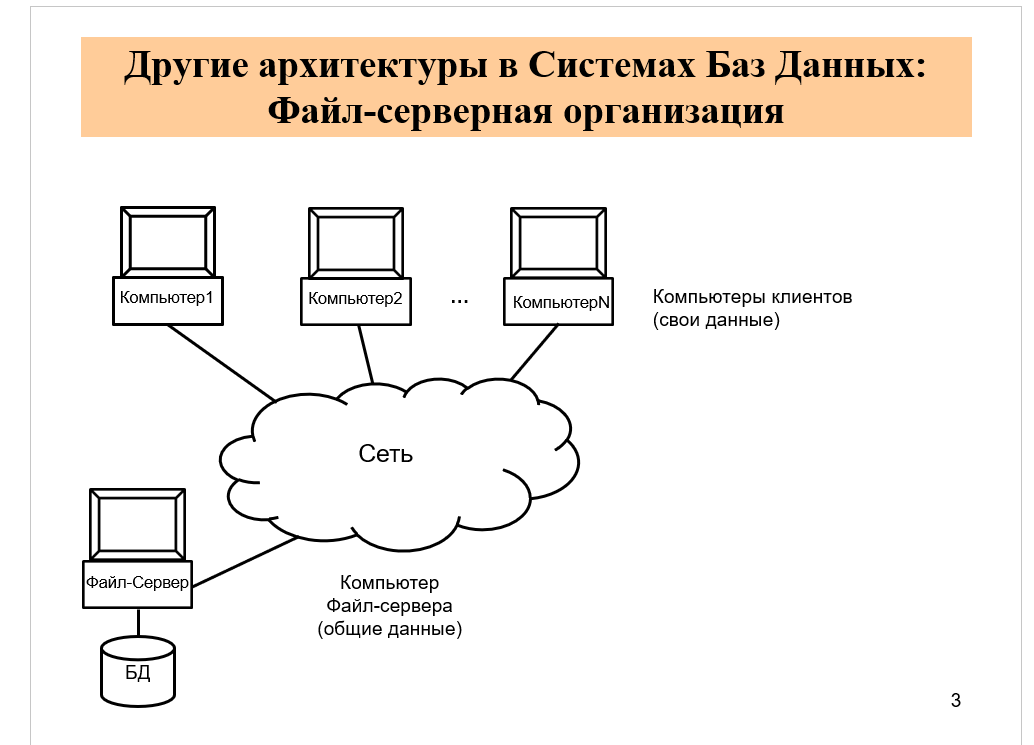
Если управление данными, обработка и представление данных делятся между тремя ЭВМ, то говорят о 3-х уровневой модели с двумя серверами в архитектуре клиент/сервер.

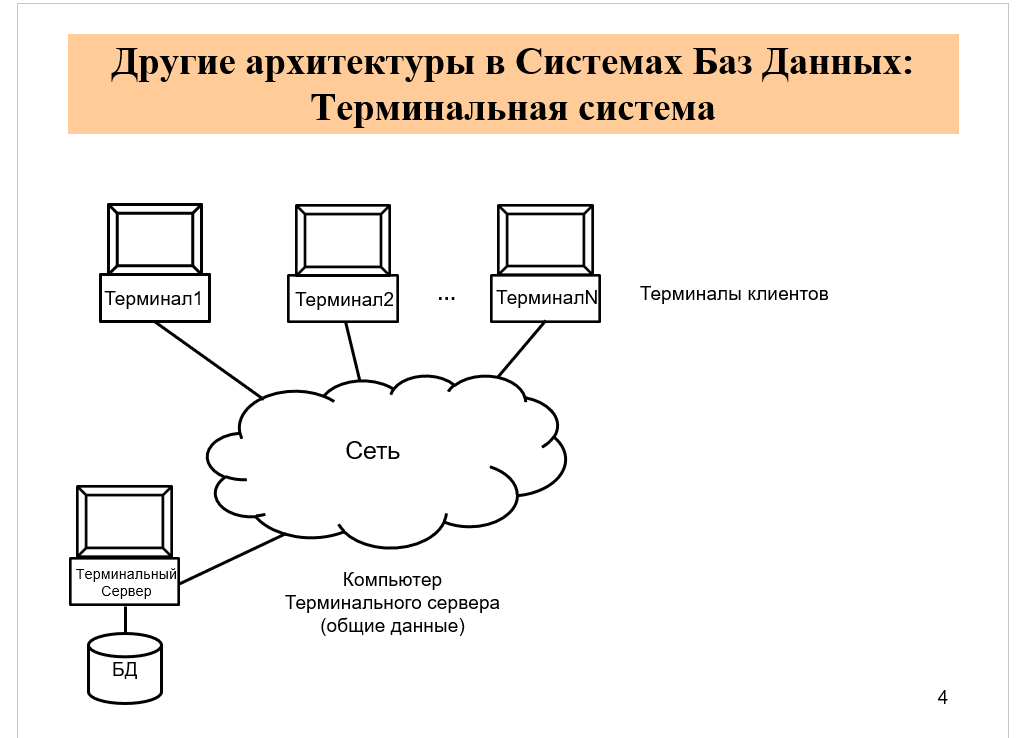
Преимущества:

1. Более полно используются технические возможности компьютеров. Сервер БД может быть реализован на компьютере, который ориентирован на возможности дисковой подсистемы. Клиентская машина – на удобный интерфейс;
2. Высокая централизация функций. Чтобы их изменить, достаточно обновить программное обеспечение на серверах);
3. Достаточно эффективное распараллеливание функций между тремя компьютерами (два сервера и клиент).

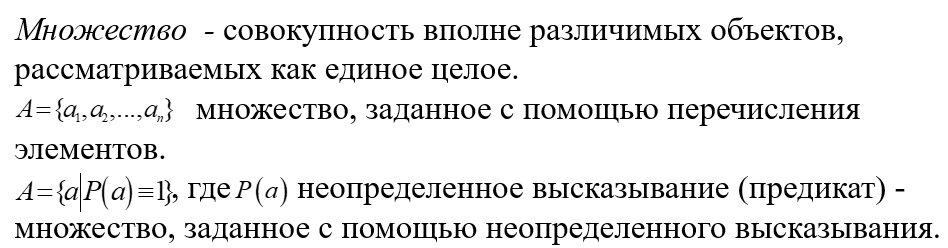
Недостаток:

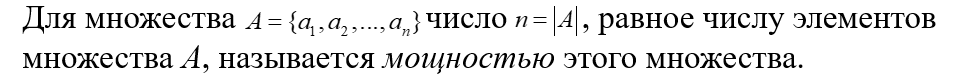
1. Повышенные требования к надежности компьютеров серверов.



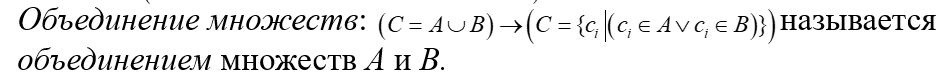


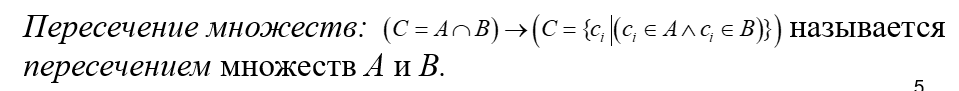
**\*Основные понятия теории множеств**

**Множество **

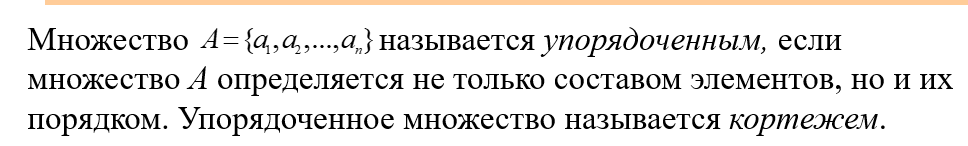
**Мощность множества. **

**Подмножество. **

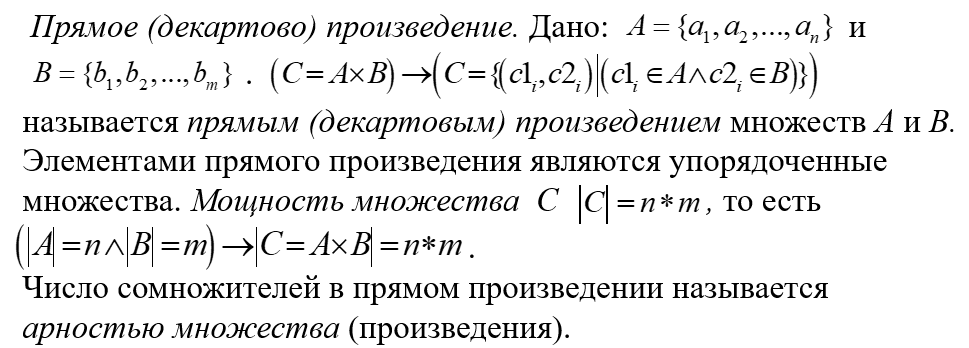
**Объединение множеств. **

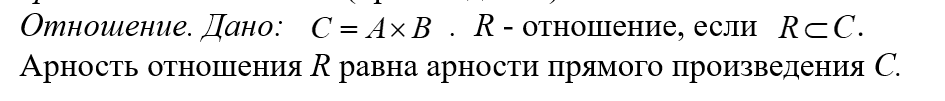
**Пересечение множеств. **

**Разность множеств. ????**

**Упорядоченное множество **

**Прямое (декартово) произведение.**

****

**Отношение. **

**\*Основные термины и понятия реляционных БД.**

**Объект** – сущность предметной области.

**Атрибут (имя Атрибута, реквизит)** – параметр объекта предметной области. (Свойство некоторой сущности).

**Домен (атрибута) -** множество допустимых значений, которые может принимать атрибут.

**Схема отношения -**конечное множество [имен] атрибутов, определяющих объект. (Мощность схемы отношения = арности кортежей.)

**Отношение -** конечное множество кортежей (подмножество прямого произведения), составленных из допустимых значений атрибутов схемы отношений.



**Ключ (первичный ключ)** – множество атрибутов, значение которых уникальным образом идентифицирует кортеж в отношении. В любой момент времени в отношении не может существовать два кортежа имеющих одинаковые значения первичного ключа.

**Схема реляционной базы** – множество используемых в приложениях схем отношений данных.

**Реляционная база данных (РБД)** – множество отношений (предполагается, что отношения логически связаны между собой).

**Реляционные операции –** операции над отношениями. Результатом любой реляционной операции является также отношение.

**Реляционное выражение** – выражение над отношениями, составленное из реляционных операций. Результатом реляционного выражение является тоже отношение.

**Реляционный запрос** – описание свойств (условий), которые должны удовлетворять интересующие пользователя данные. Эквивалентной формой описания запроса является реляционное выражение.

**СУБД** – набор программных средств, обеспечивающих хранение и обработку данных в базе. Взаимодействие прикладной программы (приложения) с базой данных выполняется через СУБД. Приложение взаимодействует с СУБД на некотором языке.

**Язык описания данных** – язык, позволяющий описать структуру БД и создать БД с требуемой структурой.

**Язык манипулирования данными** – язык, позволяющий описать действия по чтению, добавлению, обновлению и удалению данных в БД.

**Язык запросов** – часть языка манипулирования данными, предназначенный для удобного определения сложных реляционных запросов.

**Целостность базы данных** (или логическая целостность данных)– свойство БД, при наличии, которой БД содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений.

**Ограничения целостности** – набор условий, определяющие целостность базы данных. Различают ограничения диапазонов возможных значений атрибутов и структурные ограничения (т.е. ограничения на кортежи, имеющиеся в отношениях). Примером ограничения диапазонов является определение доменов атрибутов. Примером структурного ограничения является определение первичных ключей.

**Транзакция** – операция по изменению содержания БД. После завершения транзакции БД должна обладать целостностью. В течение транзакции целостность базы данных не проверяется. Выполнение транзакции завершается двумя способами:

1. отмена транзакции (возврат в предыдущее, целостное состояние);
2. регистрация транзакции, т.е. проверка и восстановление целостности БД.

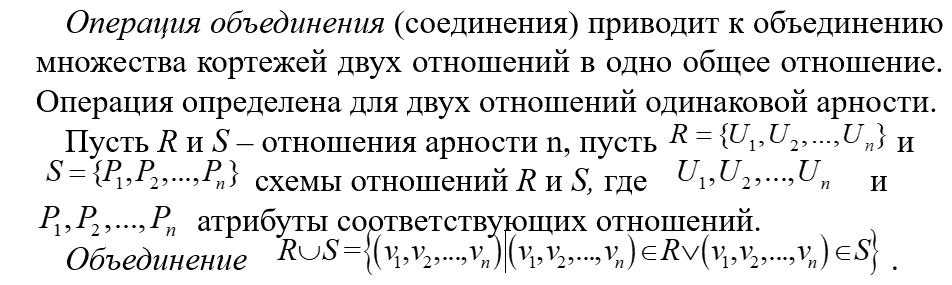
**Защита баз данных** – это:

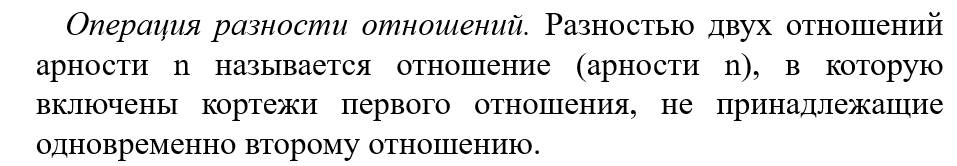
1. защита БД от физических и логических разрушений;
2. обеспечение санкционированного доступа к данным.

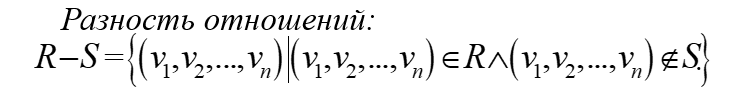
Разрушенная база данных не обладает целостностью и требует восстановления. Каждый пользователь может иметь свои санкции для доступа к базе данных (свою видимую область БД, свои права на выполнение каждой из операций над данными).

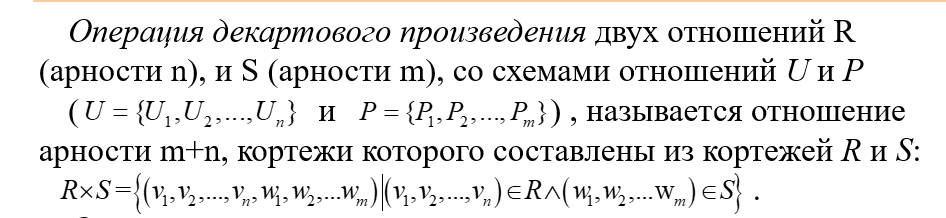
Для предотвращения физического доступа к данным используется хранение закодированных данных. Кодирование и декодирование автоматически выполняется СУБД незаметно для приложений и пользователей.

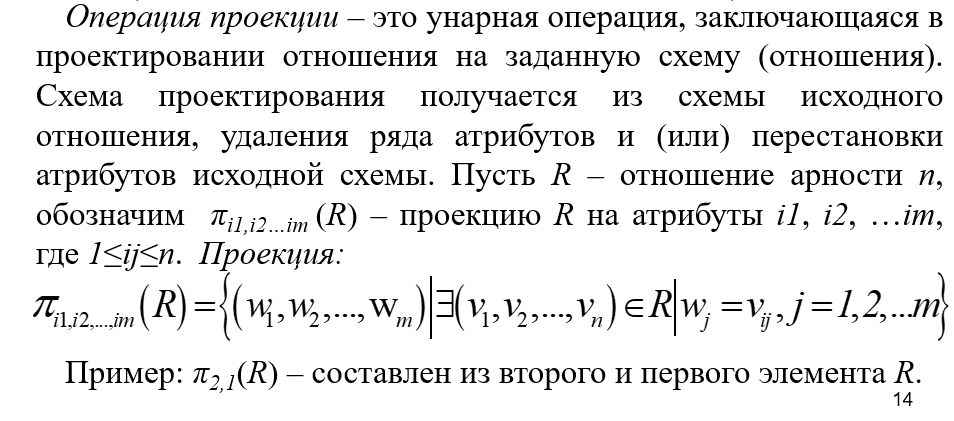
**\*Основные реляционные операции**

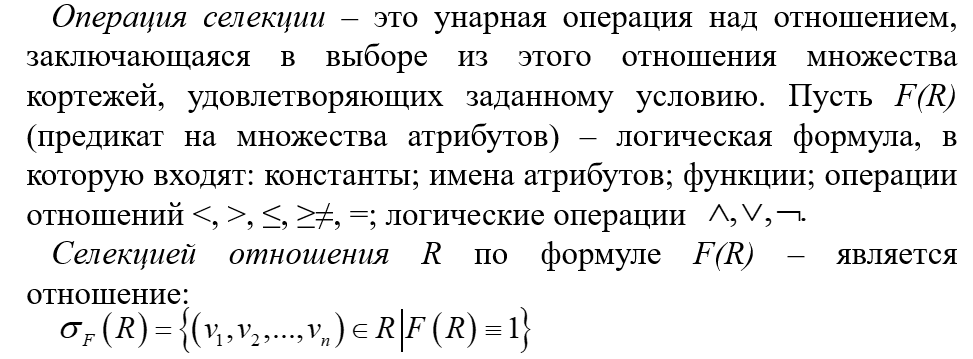
**Операция объединения (соединения)** 

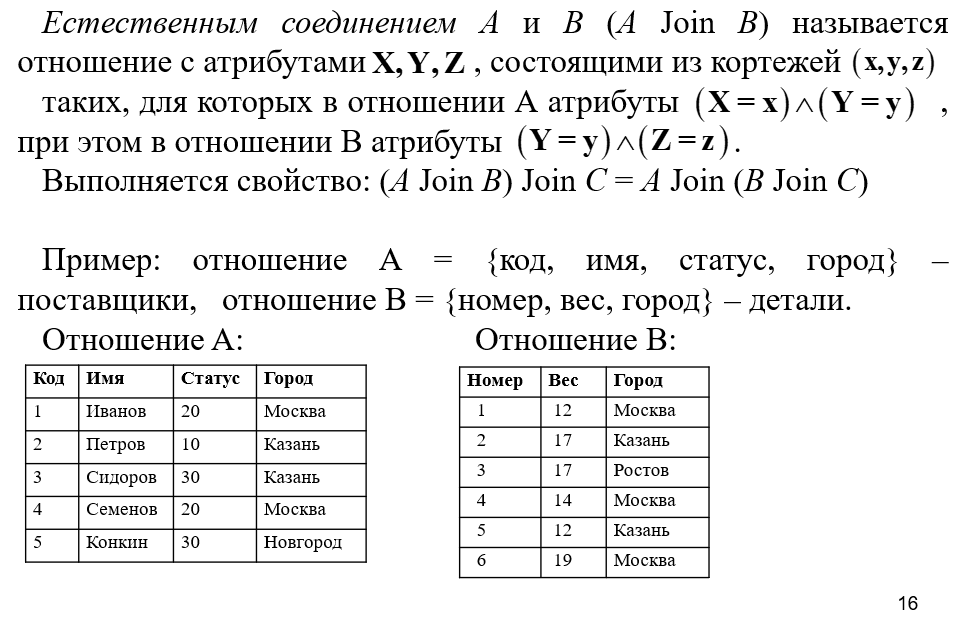
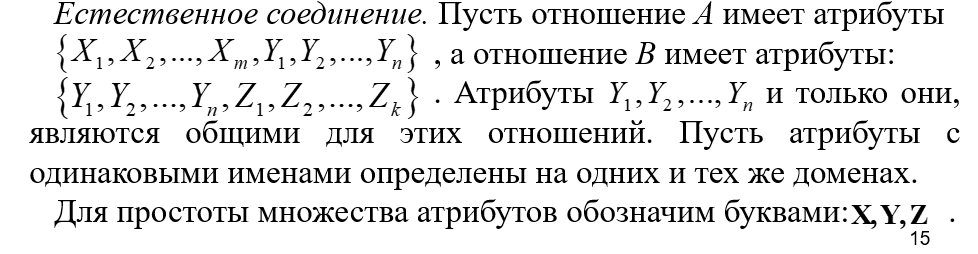
**Операция разности отношений. **

**Разность отношений **

**Операция декартового произведения **

**Операция проекции **

**Операция селекции **

**Естественное соединение.**

**\*Правила Кодда (требования к реляционным БД)**

**Явное представление данных.** Все данные должны быть представлены явно и их значения не должны рассчитываться косвенными алгоритмами (исключение – однозначные отображения).

Пример: если, явно не указан пол сотрудника, то его нельзя (ошибочно) получать из фамилии, т.к. различные алгоритмы интерпретации фамилии в различных приложениях могут вызвать противоречия (нарушить целостность) в БД. Для явного представления данных используются типы: числа, строки, даты, время, логический.

**Гарантированный доступ к данным.** Вся информация в БД должна быть доступной для приложения. Выделение любого значения в РБД выполняется при указании:

* имени отношения
* указателя на кортеж
* имени атрибута

Тройка ( «имя отношения», «первичный ключ», «имя атрибута») однозначно указывает на значение атрибута.

**Полная обработка неопределенных значений.** Неопределенные значения, отличные от любого определенного значения, должны поддерживаться для всех типов данных, при выполнении всех операций. Это правило относится, прежде всего, к атрибутам.

**Доступ к базе данных в терминах реляционной модели.**Описание БД (перечень отношений, определения схем отношений и ключей, статистическая информация и т.д.) должно быть выполнено на реляционном языке.

Пользователь должен иметь доступ к этой информации с помощью реляционного языка.

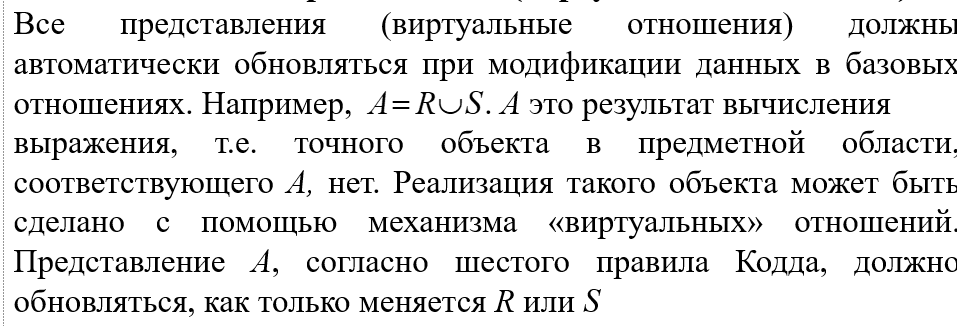
Т.е. должна быть возможность администрирования баз данных независимо от приложений.

**Полнота подмножества реляционного языка.** Реляционная схема может поддерживать несколько языков, по крайней мере, языки DDL и DML. Однако хотя бы один из языков должен иметь синтаксис предложений, поддерживающий следующие понятия:

* определение данных (отношения, атрибуты, домены, ключи, ограничения целостности);
* определение виртуальных (мнимых) отношений образованных с помощью реляционных выражений на основе одного или нескольких отношений (определение представлений);
* манипулирование данными (интерактивное или программное);
* ограничения целостности;
* санкционированный доступ;
* управление транзакциями (начало транзакции, фиксация выполнения, отказ от выполнения).

Язык определения данных должен обеспечить выполнение первого и второго правила в самых сложных ситуациях.

Пример: если некоторое значение вычисляется на основе значений нескольких атрибутов, то оно может быть явно представлено в одном из виртуальных представлений. Виртуальное отношение формируется на основе некоторого выражения реляционного языка. Виртуальное представление может использоваться для доступа как обычное отношение БД. Корректность информации, доступной через виртуальное представление обуславливается следующим (шестым) правилом

**Обновляемость представлений. **

**Наличие высокоуровнего языка манипулирования данными.** Операции вставки, обновления и удаления должны применяться к отношению в целом, обеспечивая контроль над целостностью базы данных при модификации отношений. При выполнении модификации отношения в целом легко проверить ограничения: уникальности ключа, ограничения на значения и пр.

**Физическая независимость данных.** Прикладные программы не должны зависеть от используемых способов хранения данных на носителях информации и методов доступа к ним. Физическая независимость обеспечивает работоспособность приложений при изменении расположения данных в сети.

**Логическая независимость данных.** Прикладные программы не должны зависеть от реализации любого из используемых представлений (виртуальных отношений). Определив виртуальные отношение в рамках БД, можно разрабатывать приложения, использующие эти отношения, не беспокоясь о том, что структура БД изменится и виртуальные отношение будут строиться на основе других реляционных выражений.

**Независимость контроля целостности.** Все ограничения целостности и внешнее представления (виртуальные отношения, отчеты) должны определяться не в приложениях, а должны быть определены с помощью языка определения данных и сохранения в каталоге (словаре) базы данных.

**Дистрибутивная независимость.** Реляционная система должна быть распространяема и переносима. Создание разнородных компьютерных систем требует обеспечения доступа к базам данных в различных OS и на различных платформах. Дистрибутивная независимость предполагает полную реализацию СУБД для различных платформ или реализацию коммуникационных блоков в составе СУБД, позволяющих обмениваться данными различным СУБД

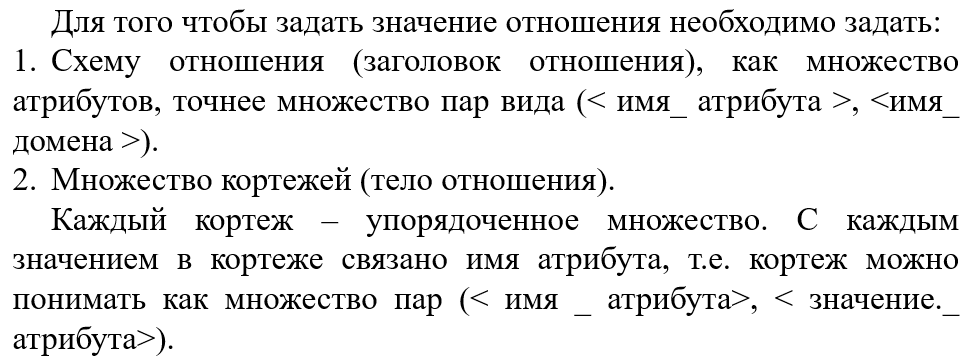
**Согласования языковых уровней.** Все ограничения целостности и внешние представления (виртуальные отношения, отчеты) должны определяться не в приложениях, а должны быть описаны с помощью языка определения данных (DDL) и сохранены в каталоге (словаре) базы данных.

# 2 БЛОК

**Отношения и реляционная алгебра.**

В реляционной модели База Данных рассматривается как множество отношений, а отношение – как множество кортежей. На множестве отношений заданы реляционные операции. В результате получена алгебра, т.е. система, позволяющая производить вычисления на множестве отношений. Данная система называется реляционной алгеброй. В реляционной алгебре операндами и результатом каждой операции являются отношения. Следовательно, полученная система является замкнутой (операции не выводят из множества отношений). Как и во всякой алгебре, мы приходим к понятиям «переменная» и «значение переменной» (в нашем случае – «переменная отношения» и «значение переменной отношения»).

**Переменная отношение** – это абстрактное понятие, под которым мы будем понимать произвольное отношение. Для некоторых операций – произвольное отношение с определенной схемой (заголовком отношения).

**Как задать переменную отношения**

**Свойства переменной отношения**

**Проблемы свойства атомарности атрибутов отношения и 1НФ**

Пояснения к свойствам отношений:

* Отношение это множество кортежей, следовательно, кортежи не упорядочены и в отношении не существует двух одинаковых кортежей, следовательно, в отношении существует хотя бы один первичный ключ. Поиск нужного кортежа можно реализовать с помощью ключей.
* Каждому атрибуту в кортеже соответствует уникальное имя из схемы отношения. При этом каждый атрибут является атомарным, в том смысле, что его невозможно разбить на более простые части, которые соответствуют каким-либо параметрам объекта из предметной области.

Очень часто требования к свойства отношений в конкретных языках СУБД нарушаются (свойства 1-3). Требование к атомарности атрибутов часто нарушаются разработчиками БД.

Примеры нарушений требований к отношениям:

В некоторых СУБД допускается, чтобы кортежи повторялись, вводится упорядоченность кортежей и атрибутов в кортеже. В языках СУБД есть операции: перейти на первый кортеж; на следующий кортеж; прочитать атрибут № 5.

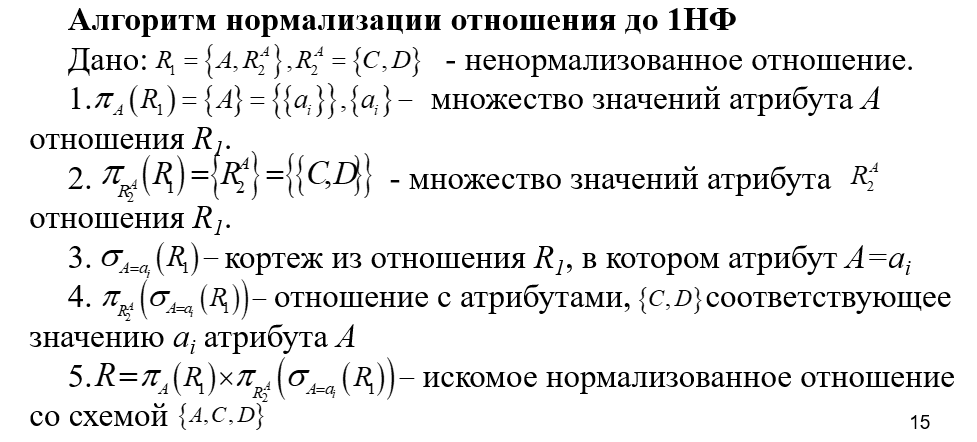
Проблема нарушения требования атомарности атрибутов.

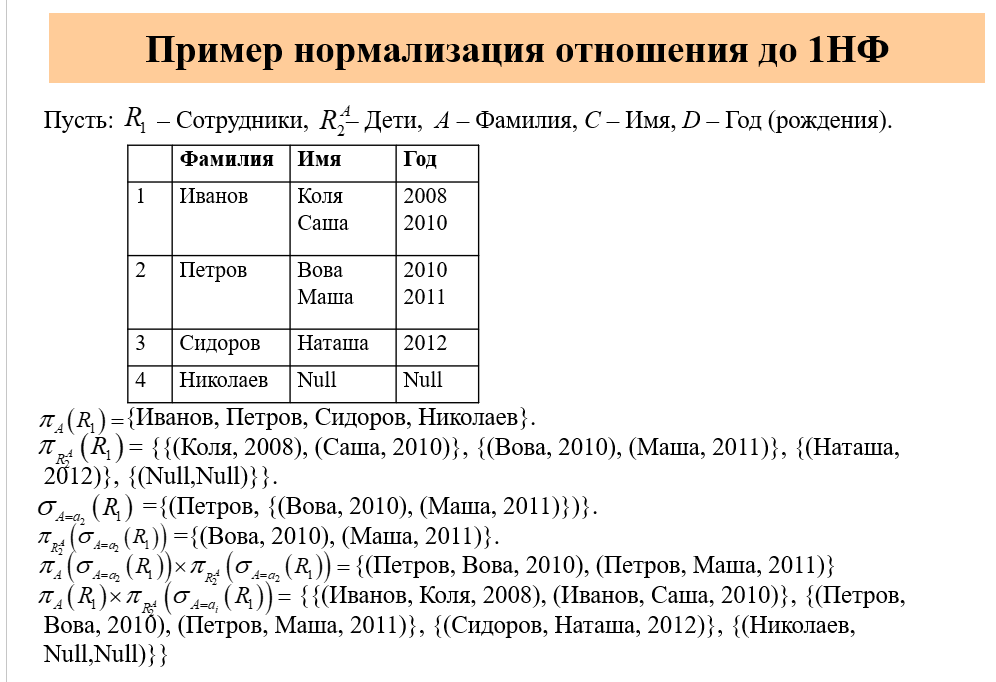
Теория множеств допускает, что элемент кортежа был бы отношением. В некоторых СУБД такое определение допускается.

Пример1: База данных сотрудников. У объекта «Сотрудник» есть атрибут Дети. Значение атрибута - список: пар (имя ребенка, год рождения). Данное отношение не удовлетворяет требованию атомарности атрибутов, но достаточно точно описывает предметную область

Пример2. Атрибут ФИО можно иногда рассматривать как атомарный атрибут, иногда как множество их 3-х атрибутов: «Фамилия», «Имя», «Отчество».

Отношения, у которых все атрибуты атомарны, называются нормализованными (находящимися в 1НФ). Реляционная теория БД рассматривает только нормализованные отношения. Математическое отношение необязательно нормализовано.

**Алгоритм нормализации отношения до 1НФ. Пример.**

****

**Виды отношений, используемых в реляционных системах.**

**Определение: Именованное отношение.** – это переменная типа отношение, у которой есть имя.

**Определение: Базовое отношение.** – это именованное отношение, которое не является производным от других отношений.

**Определение: Производное отношение.** – это отношение, определённое через другие именованные отношения (посредством реляционного выражения) и, в конечном итоге, через базовые отношения.

**Определение: Выражаемое отношение.** – это отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения. Каждое именованное отношение является выражаемым отношением, но необязательно, что выражаемое отношение имеет имя.

**Определение: Множество всех выражаемых отношений.** – это объединение множества всех базовых отношений и множества всех производных отношений.

**Определение: Представление.** - это именованное производное отношение. Представление, как и базовое отношение, является переменной отношения.

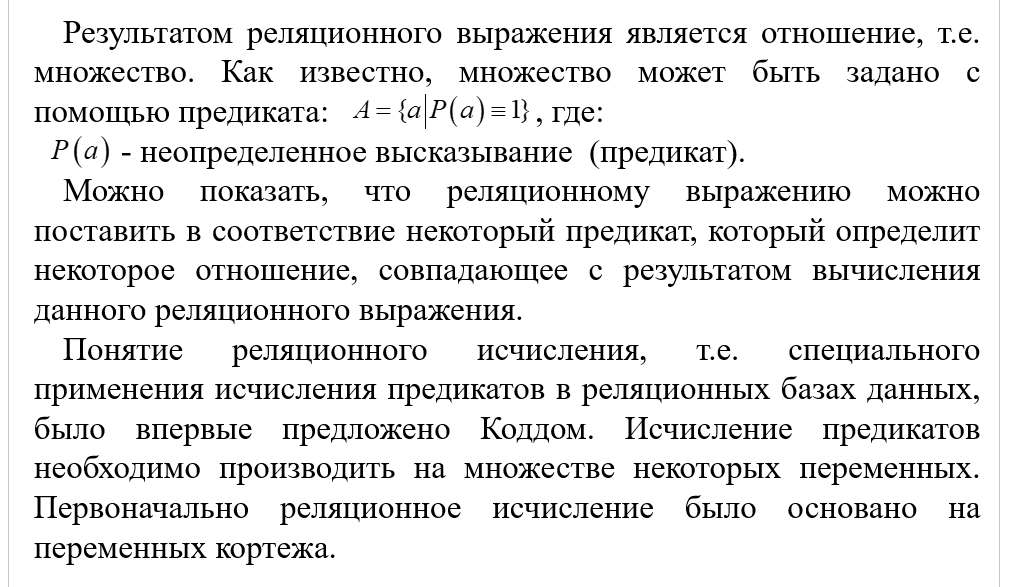
**Определение: Снимки.** – это именованные производные отношения (являются переменными отношения). Создание снимка похоже на выполнение запроса, за исключением того, что результат такого запроса сохраняется в базе данных под некоторым именем как отношение, доступное только для чтения. Периодически (например: раз в сутки) снимок обновляется.

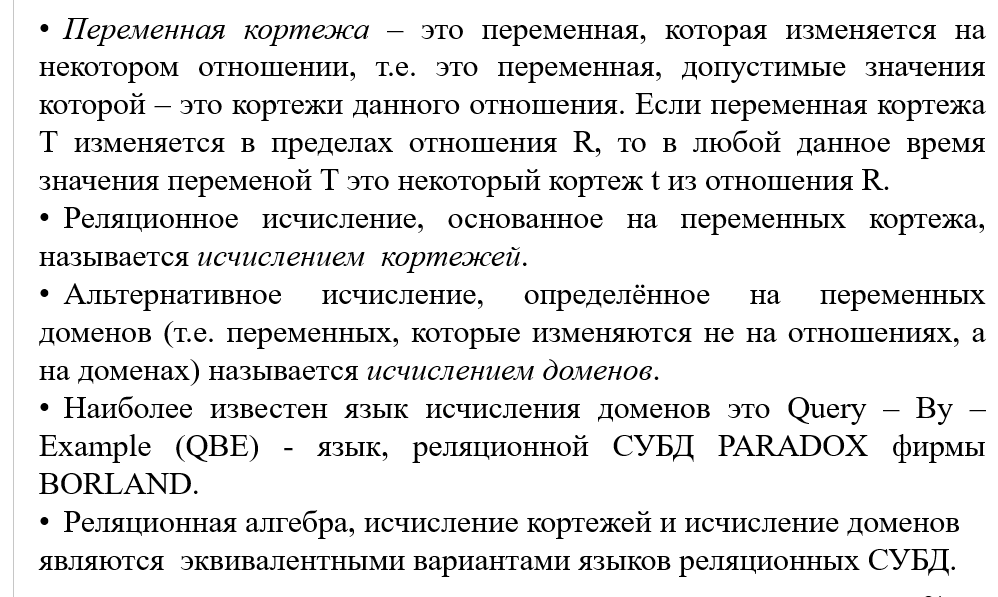
**Определение: Результаты запроса.** – неименованное производное отношение, является результатом вычисления некоторого реляционного выражения. База данных не обеспечивает постоянное хранение результатов запроса. Для этого результат запроса необходимо присвоить некоторому именованному отношению.

**Определение: Промежуточный результат.**называется неименованное производное отношение, являющееся результатом некоторого реляционного выражения.

**Определение: Хранимое отношение.**– это отношение, которое поддерживается во внешней памяти.

**\*Исчисление предикатов.**

****

****

**Понятие целостности БД**

Целостность БД – свойство БД, при наличии которого БД содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений.

Некоторые условия целостности проверяются с помощью типов (например, даты). В сложных случаях целостность проверяется вычислением логических выражений. Разработка таких выражений существенно опирается на предметную область.

Большинство правил целостности являются специфическими для данной БД.

Имеется два правила целостности, применимые к любой базе данных. Эти два правила относятся к так называемым потенциальным и внешним ключам.

**Определение потенциального ключа. Пример.**

Потенциальный ключ – это обобщение понятия первичного ключа. Первичный ключ уникальным образом идентифицирует кортеж в отношении.

Потенциальные ключи обладают свойством уникальности; в

отношении их может быть несколько; первичный ключ только один и выбирается из потенциальных ключей.

Определение потенциального ключа:

Пусть R – некоторая переменная отношения, тогда потенциальный ключ K для R это подмножество атрибутов R, всегда обладающее следующими свойствами:

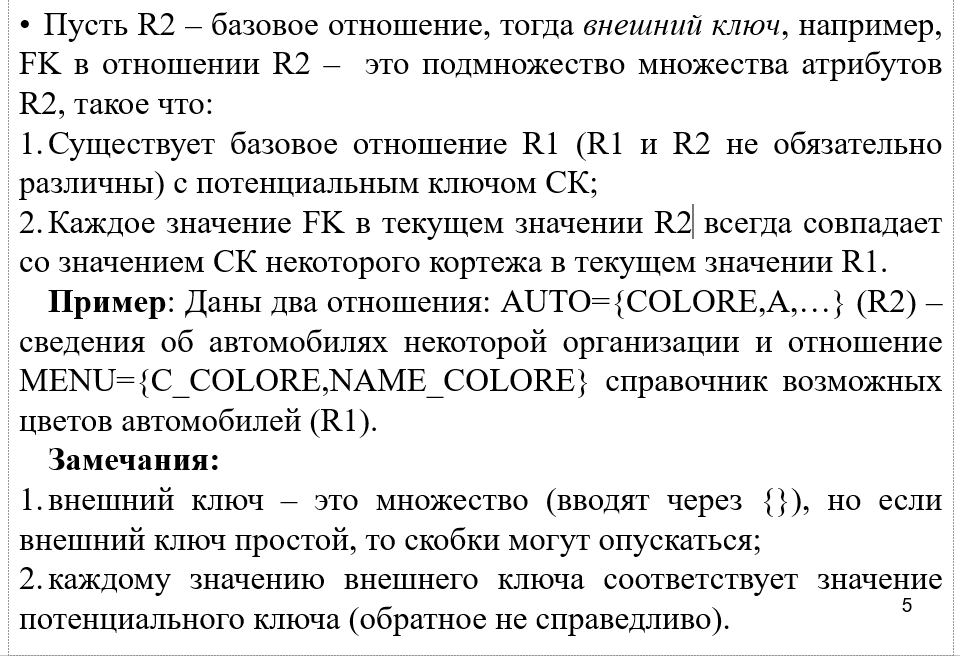
1. Свойство уникальности: нет двух различных кортежей в текущем значении переменной R с одинаковыми значениями K.
2. Свойство не избыточности: никакое из подмножеств K не обладает свойством уникальности.

На практике в отношениях чаще всего встречается один потенциальный ключ, который выбирается первичным.

Пример исключения: дана БД элементов таблицы Менделеева: порядковый номер элемента, атомная масса, название элемента и т.д. – все это потенциальные ключи.

Из определения следует, что потенциальный ключ – множество атрибутов.

* Потенциальный ключ из 1-го атрибута – это простой ключ.
* Потенциальный ключ из нескольких атрибутов – это составной ключ.
* Если потенциальных ключей несколько, то один должен быть выбран в качестве первичного ключа, а остальные будем называть альтернативными ключами.

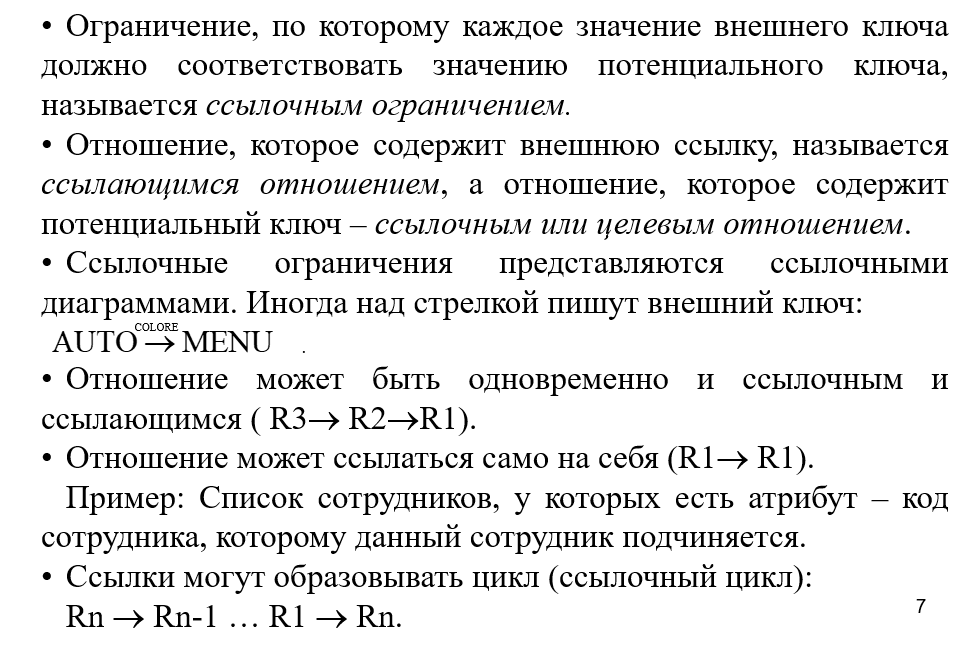
**Определение внешнего ключа. Пример.**

**Основные требования к ключам отношений. Примеры.**

* Если внешний ключ составной, то соответствующий ему потенциальный ключ тоже составной. Если внешний ключ простой, то потенциальный ключ тоже простой.
* Каждый атрибут, входящий в данный внешний ключ, должен быть определен на том же домене, что и соответствующий атрибут, соответствующего потенциального ключа.
* Для внешнего ключа не требуется, чтобы он был компонентой первичного ключа данного отношения.
* Значение внешнего ключа является ссылкой на кортеж, содержащий соответствующее значение потенциального ключа (так называемый ссылочный или целевой кортеж). Проблема обеспечения того, что база данных не должна включать никаких неверных значений внешних ключей называемой проблемой ссылочной целостности.

**\*Ссылочная целостность. Пример.**

Значение внешнего ключа является ссылкой на кортеж, содержащий соответствующее значение потенциального ключа (так называемый ссылочный или целевой кортеж). Проблема обеспечения того, что база данных не должна включать никаких неверных значений внешних ключей называемой проблемой ссылочной целостности.

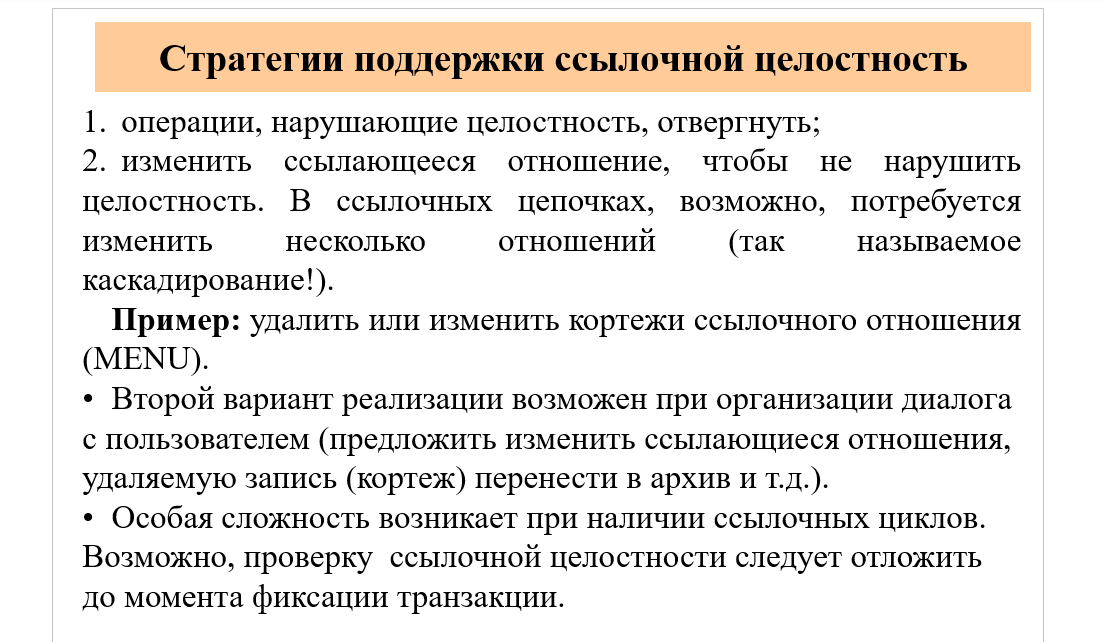


База данных обладает ссылочной целостностью, если она не содержит значений внешних ключей, которым не соответствует значение первичного ключа. Удаление в ссылающемся отношении никогда не нарушает ссылочную целостность.

Проверить ссылочную целостность в момент добавления или коррекции ссылающегося отношения не является сложным. Вся сложность лишь в предоставлении возможности добавить кортеж в ссылочное отношение из среды работы с ссылающимся отношением.

Добавление в ссылочное отношение никогда не нарушает ссылочную целостность.

Проблемы проверки ссылочной целостности возникают лишь при удалении и корректировки ссылочного отношения. Если при выполнении этих операций нарушается ссылочная целостность, то используют две основные стратегии:



**\*Правило целостности объекта. Предпосылки введения правила. Пример.**

Вместе с понятием первичного ключа реляционная модель включает правило целостности объекта, которое заключается в следующем:

Ни один элемент первичного ключа базового отношения не может иметь Null – значение.

Предпосылки введения такого правила следующие:

1. кортежи базового отношения соответствуют объектам реального мира;
2. объекты реального мира различны (т.е. некоторым образом опознаваемы);
3. значит, должны быть различимы представления реального мира, т.е. кортежи базового отношения.

Если первичный ключ или его часть имеет Null – значение, то идентичность кортежа объекту теряется.

В реляционной БД никогда не следует записываем информацию о чём – то, чего невозможно идентифицировать.

Проблема использования Null – значений до конца ещё не решена. Введение и поддержка этого правила имеет несколько противоречий.

Пример: допустим, что в базе AUTO мы ввели понятие Null – значение для атрибута COLORE. Составим список цветов автомобилей. В этом списке (отношении) возможно, будет значение – цвет не определён. Наш запрос не является базовым отношением. И для него требования целостности объекта может не применяться. Но если результат запроса сохранить как базовое отношение, то это отношение будет состоять только из одного атрибута – цвет, которое должно быть первичным ключом. Одно из значений ключа – Null.

Альтернативной стратегией Null – значениям является использование значений по – умолчанию.

**\*Внешние ключи и Null – значения. Уточнение определения внешнего ключа.**

Внешние ключи могут принимать Null – значения.

Пример: если рассмотреть самоссылающееся отношение <сотрудники>, в котором есть атрибут <код\_начальника>, то у президента компании это атрибут содержит Null – значение. Т.о. понятие внешнего ключа должно быть дополнено возможностью принимать значение Null.

Определение внешнего ключа должно быть расширено:

Пусть R2 – базовое отношение. Тогда внешний ключ, например, FK в отношении R2 – это подмножество множества атрибутов R2, такое что:

1. Существует базовое отношение R1 (R1 и R2 не обязательно различны) с потенциальным ключом СК.
2. Всегда каждое значение FK в текущем значении R2 или является Null – значением, или совпадает со значением СК некоторого кортежа в текущем значении R1.

Возможность включения или не включения Null – значений в первичные и внешние ключи регулируются соответствующими опциями операторов языка определения данных БД.

Замечание: Допустимость принятия Null – значения для внешнего ключа в некоторой степени решает проблему ссылочной целостности.

Пример: при удалении некоторого цвета из справочника цветов достаточно в записях об автомобилях (отношение AUTO), у которых атрибут <Цвет> совпадал с удаляемым из отношения <MENU> цветом, установить цвет в Null.

**\*Введение в теорию проектирования БД.**

Под проектированием будем понимать, прежде всего, проектирование логической схемы, т.е. схемы концептуального уровня. Чаще всего процесс проектирования состоит из нескольких циклов: – внешняя модель; – логическое проектирование; – физическая реализация. Исправление ошибок: – изменение представлений и логического проектирования.

Под проектированием будет понимать построение первоначального проекта схемы концептуального уровня.

Несмотря на то, что имеется теория проектирования только для реляционной модели, разработаны средства перехода от реляционной к любой другой модели и наоборот. Следовательно, теория проектирования может использоваться в любой модели.

Теория проектирования в полном объеме недостаточно формализована. Процесс проектирования в большой степени является, все-таки, искусством.

Проектирование БД – это не единственное условие получения рациональной структуры организации данных. Вторым условием является условие описания и проверки целостности БД.

При проектировании БД интерес представляет организация данных, а не приложения, т.е. то, как эти данные используются не интересно. Критерием правильного проекта является то, что схема БД остается стабильной и работоспособной при возникновении новых требований к данным в приложениях.

Задача логического проектирования, фактически сводится к тому, чтобы решить какие базовые отношения и с какими атрибутами следует использовать в проекте БД.

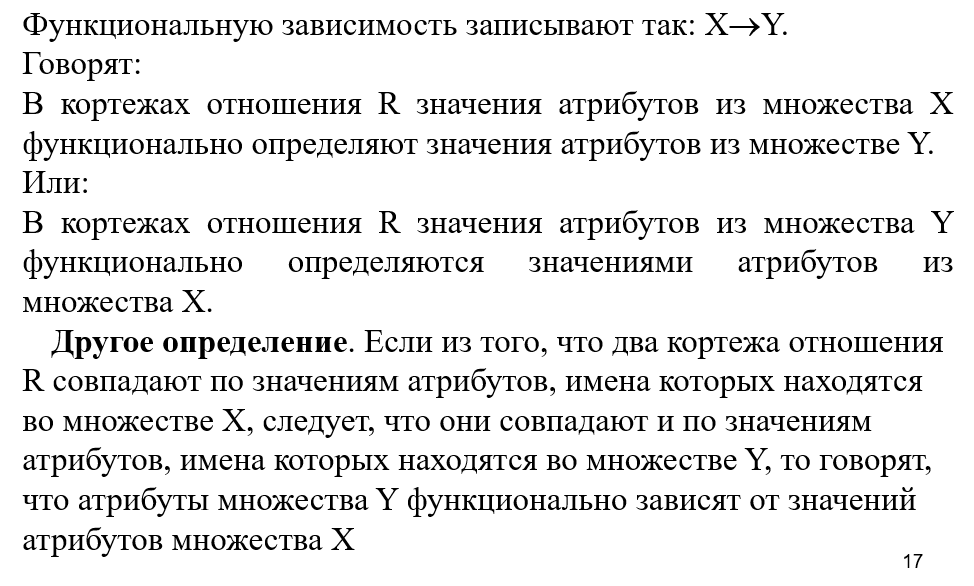
**\*Определение функциональной зависимости первого типа. Пример.**

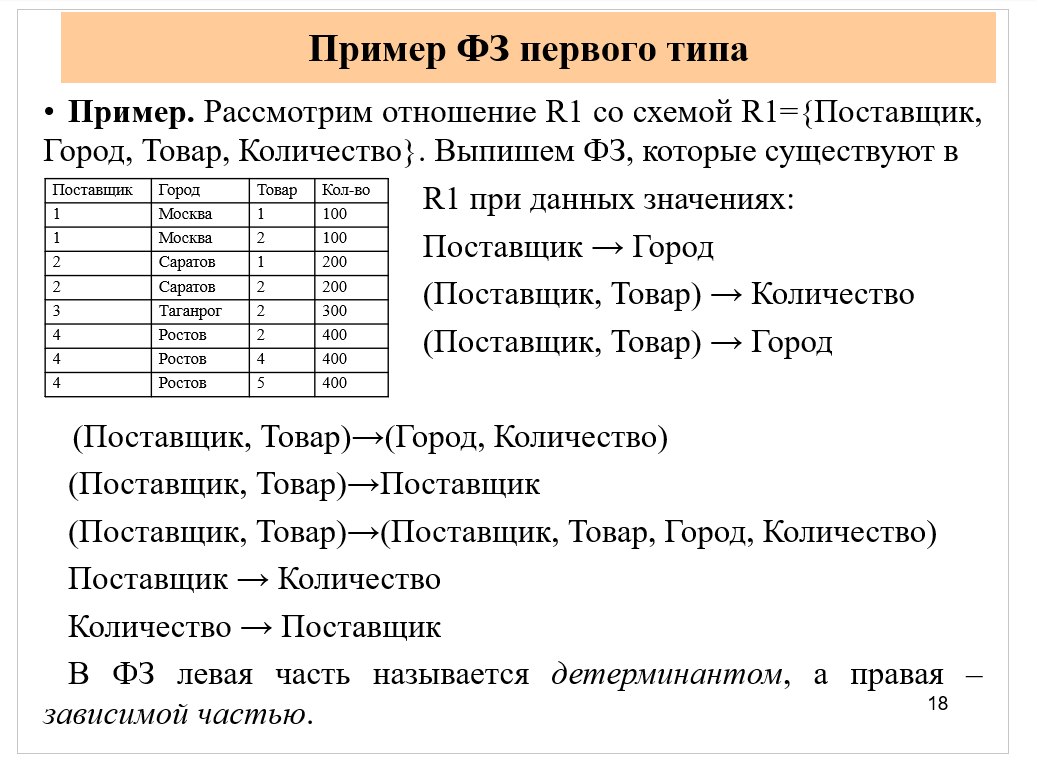
Определение функциональных зависимостей даётся в двух видах:

1. Для значения элементов отношений в некоторый момент времени;
2. Для всевозможных значений элементов отношения, в любой момент времени.

Определение Функциональных Зависимостей (ФЗ) первого типа:

* Пусть R – является переменной отношения, а X и Y являются произвольными подмножествами множества атрибутов отношения R.
* Говорят, что Y функционально зависит от X тогда и только тогда, когда каждому значению атрибутов в кортежах отношении R, имена которых находятся в множестве Х, соответствует одно и только одно значения атрибутов в кортежах R, имена которых находятся в множестве Y.





**\*Функциональные зависимости второго типа. Определение. Пример.**

При проектировании БД важную роль играют ФЗ второго типа, определение которых справедливо не только для некоторого значения переменной отношения, но и для случая, когда оно остается справедливым и для любого значения переменной отношения.

Предположим, что ФЗ поставщик→город справедлива для любого значения переменной отношения R1, т.е. справедлива в любой момент времени.

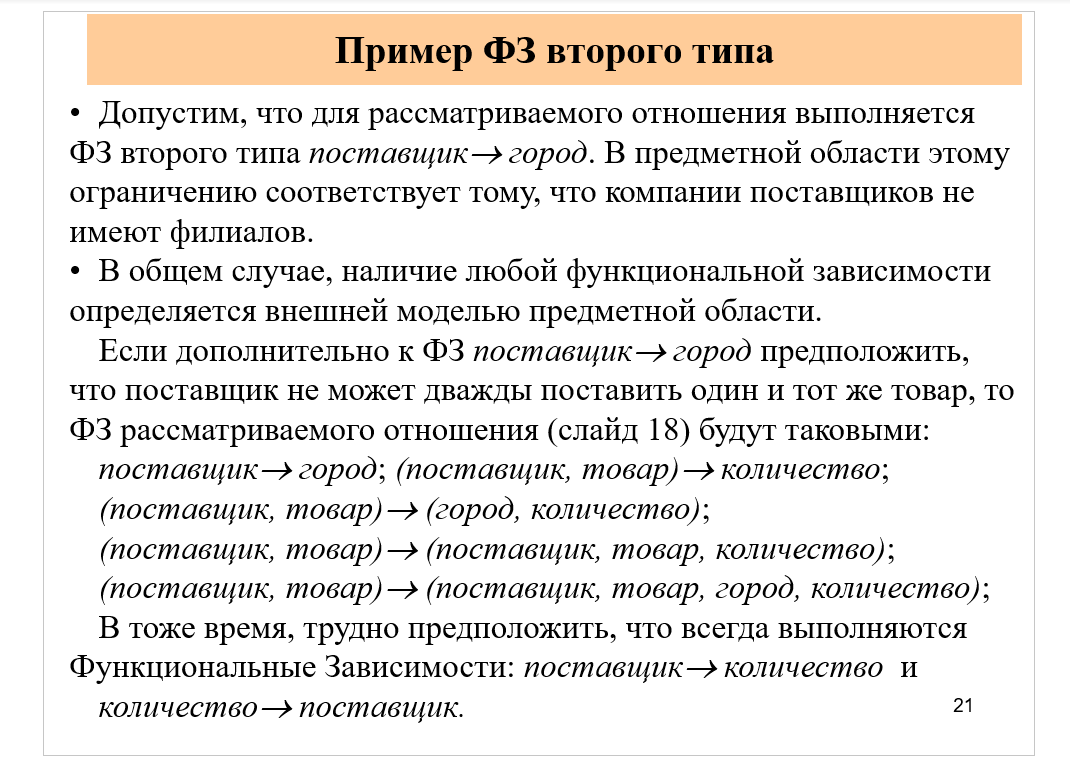
Фактически это является ограничением целостности БД. В предметной области этому ограничению соответствует то, что компании поставщиков не имеют филиалов. Или предполагается что, если компании могут иметь филиалы, то во всех документах указывается город, в котором расположен головной офис компании.

Определение функциональных зависимостей второго типа:

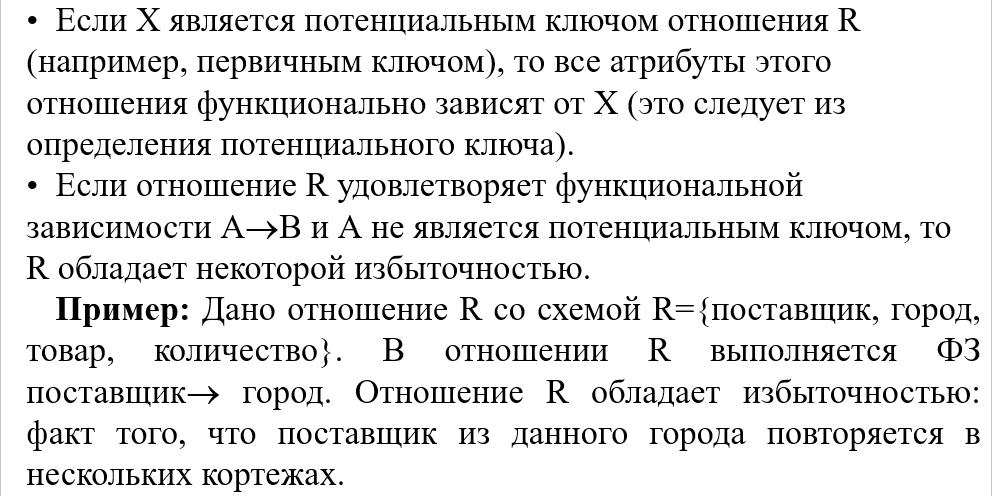
Пусть R – является переменной отношения, а X и Y являются произвольными подмножествами множества атрибутов отношения R.

Говорят, что Y функционально зависит от X для любого допустимого значения отношения R тогда и только тогда, когда каждому значению атрибутов в кортежах отношении R, имена которых находятся в множестве Х, соответствует одно и только одно значения атрибутов в кортежах R, имена которых находятся в множестве Y.

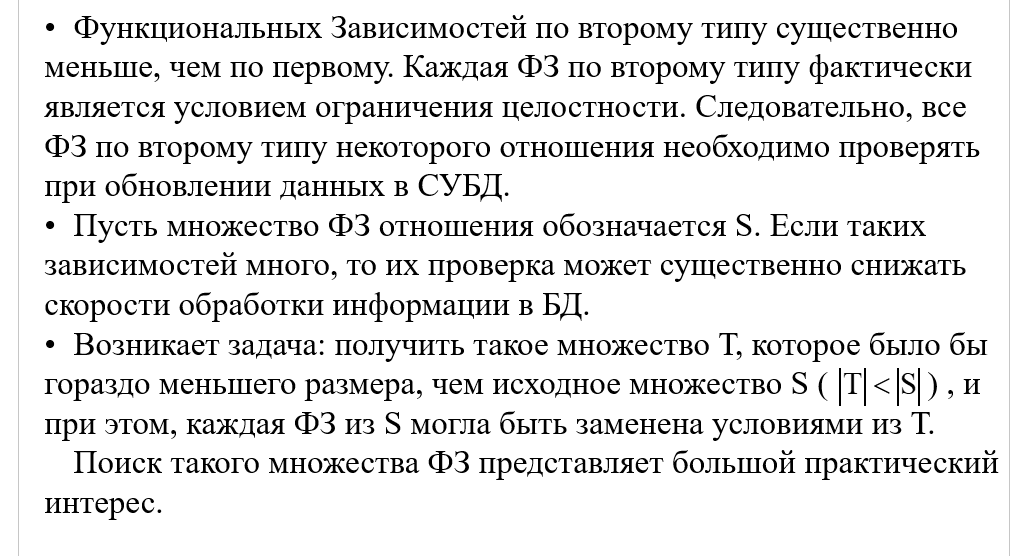
Другое определение. Если для любого допустимого значения отношения R, из того, что два кортежа отношения R совпадают по значениям атрибутов из множества X, следует, что они совпадают и по значениям атрибутов из множества Y, то говорят, что атрибуты множества Y функционально зависят от значений атрибутов из множества X.

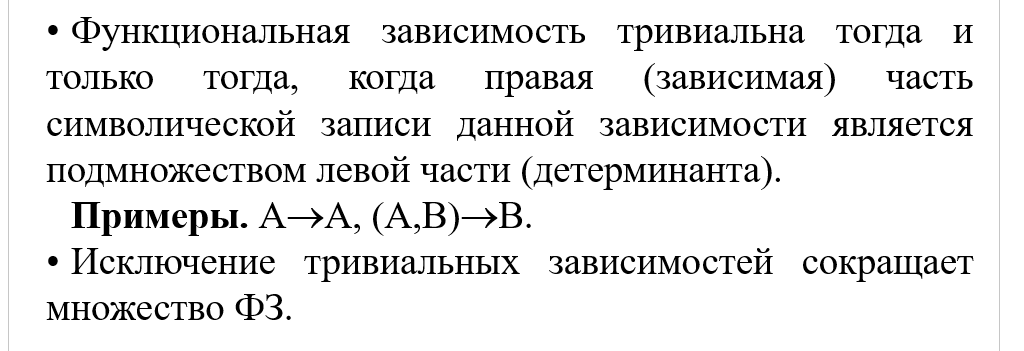


**Первичные ключи и функциональные зависимости. Пример.**

****

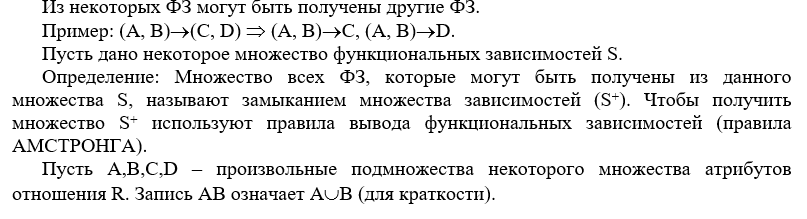
**Функциональные зависимости и целостность БД. Тривиальные ФЗ.**

****

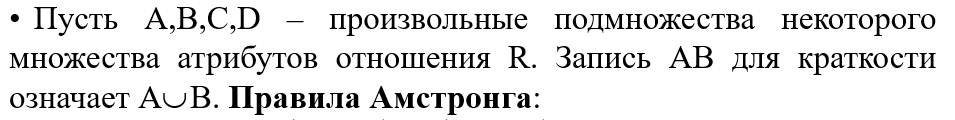


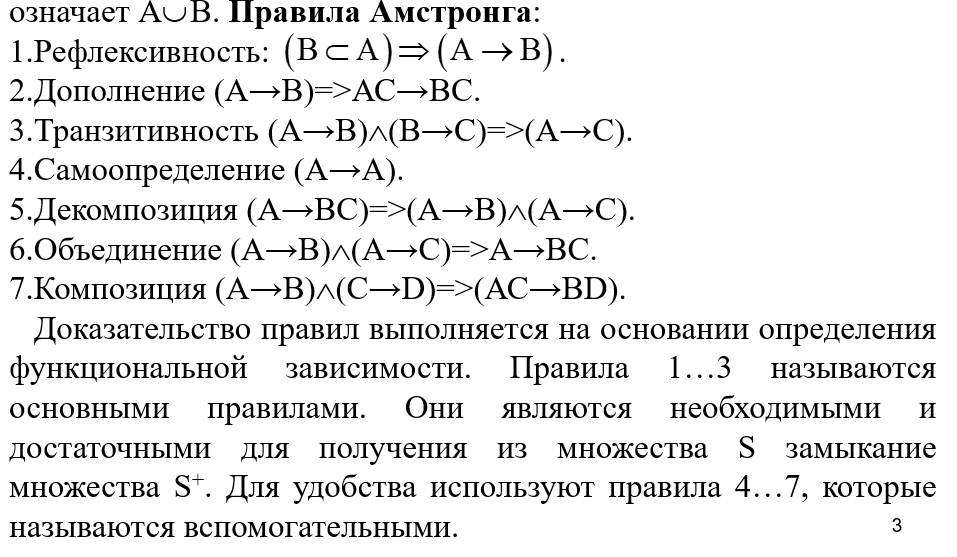
**Преобразование ФЗ. Замыкание множества ФЗ**

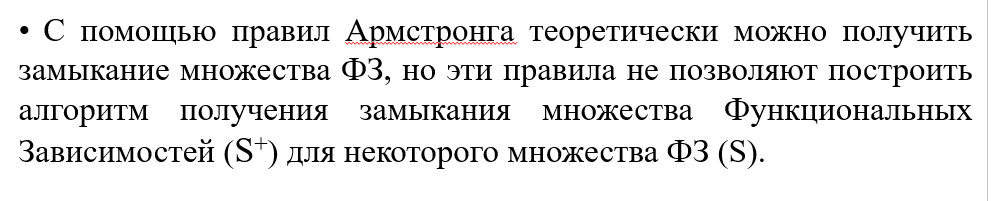
Преобразования фз - ???



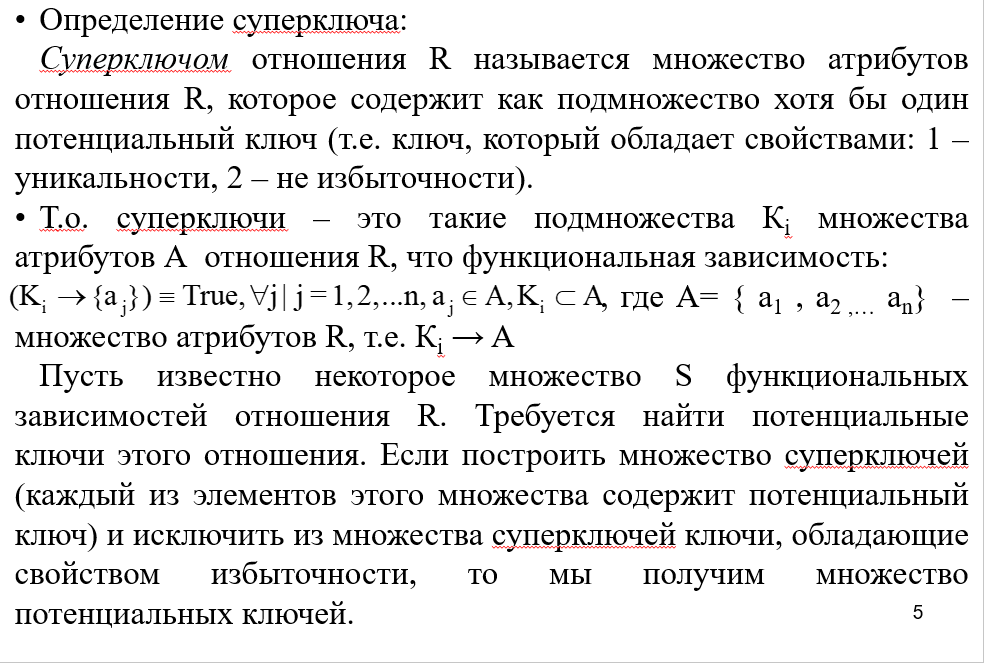
**Правила Армстронга. Определения.**

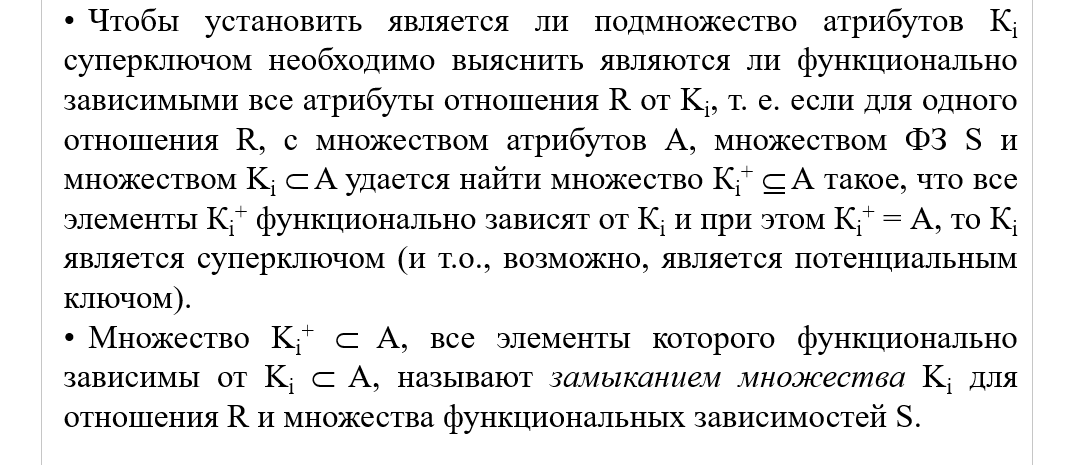
****

****

****

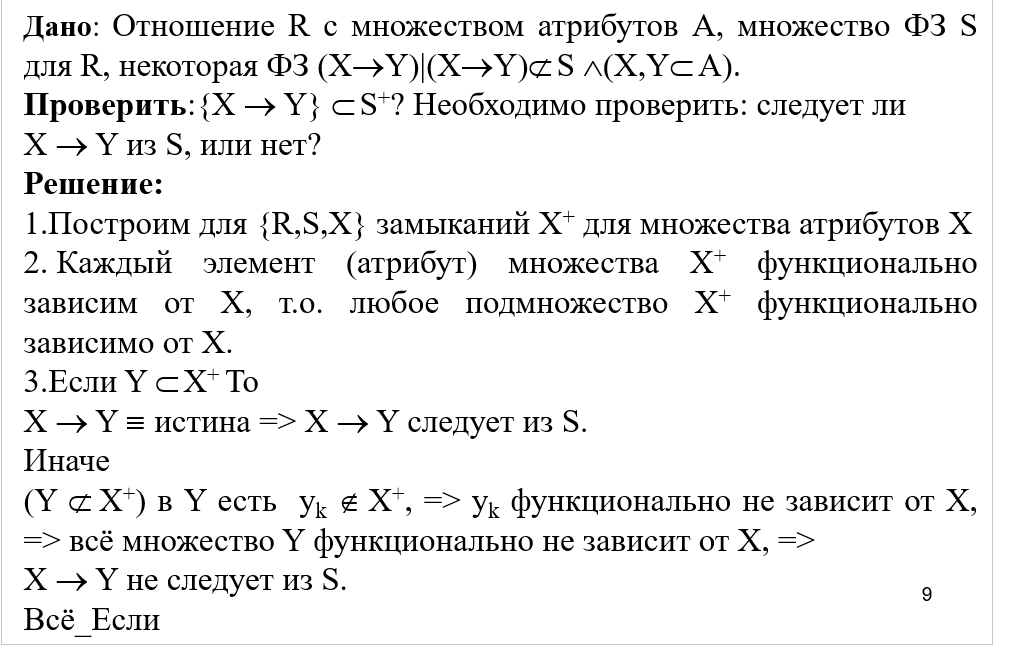
**Определение суперключа.**

****

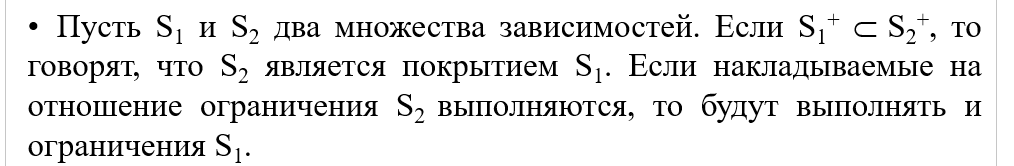
**Проблема построения потенциальных ключей отношения.  
 Определение замыкания множества атрибутов.**

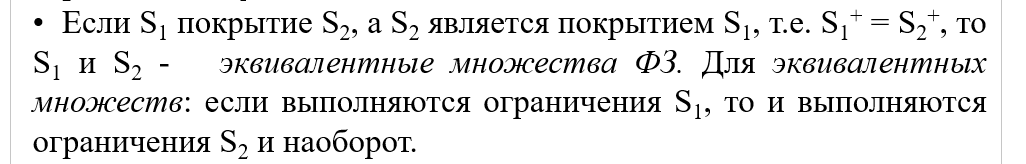
**Алгоритм построения замыкания множества атрибутов.**

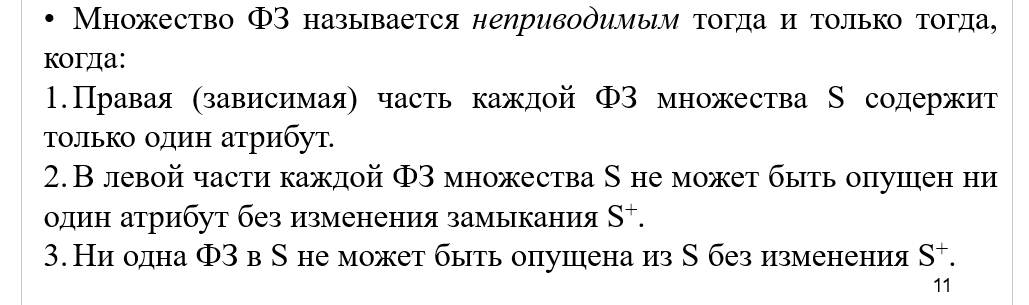
****

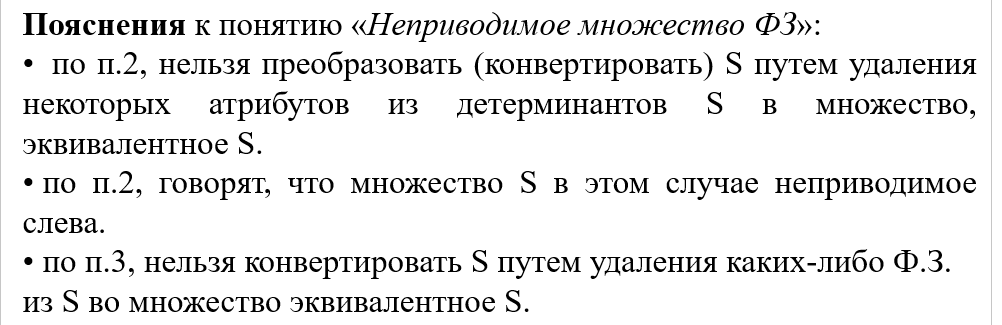
**Следствие из алгоритма построения замыкания Ki+.**

**\*Неприводимое множество зависимостей.**

**Определение покрытия множества ФЗ. **

**Определение эквивалентных множеств ФЗ. **

**Определение неприводимого множества ФЗ.**

****

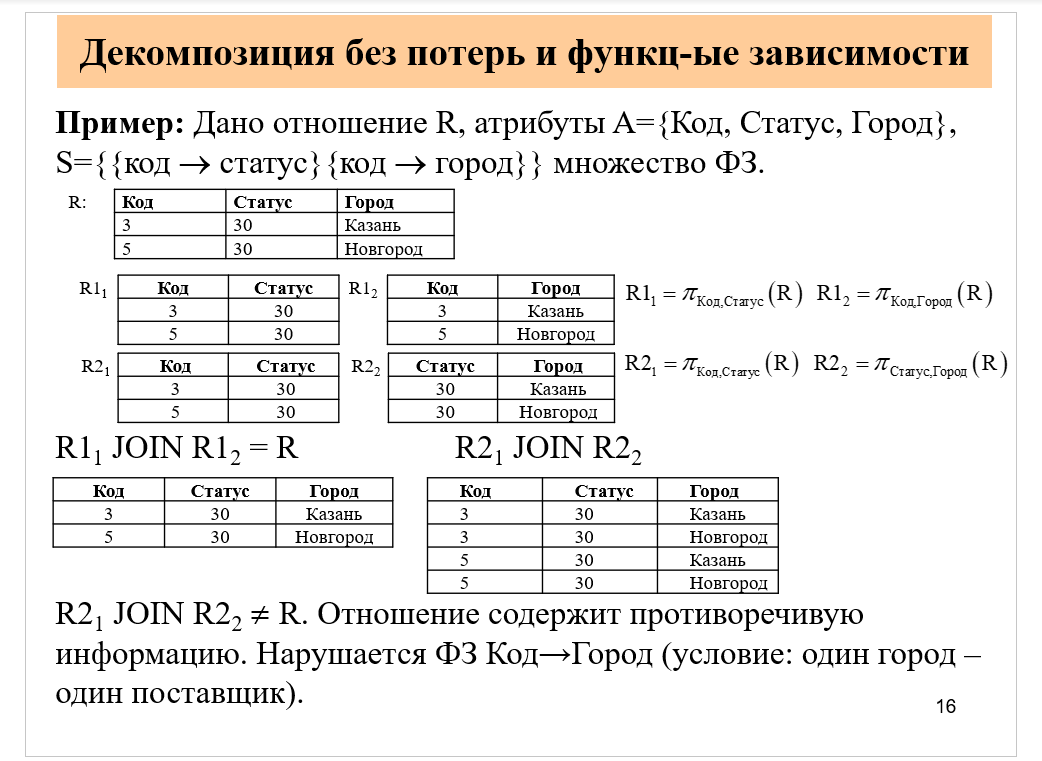
**Понятие о нормализации отношений.**

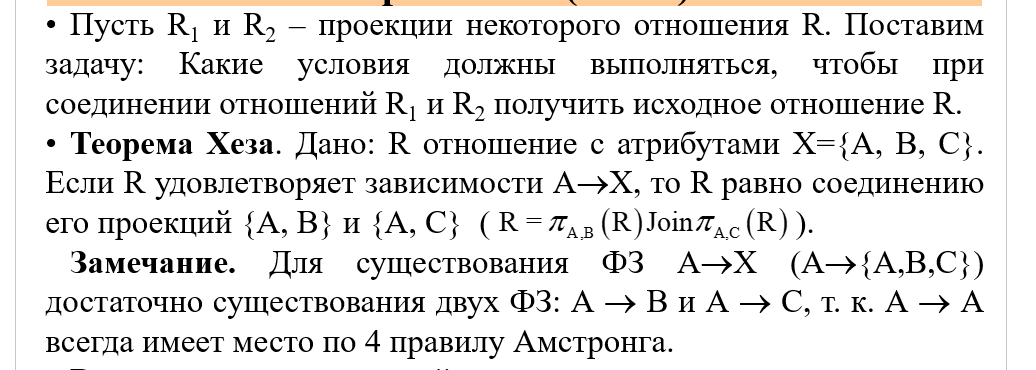
Понятие нормализованных форм ввел Кодд и определил 3 нормальных формы (1НФ, 2НФ, 3НФ). Введённая Коддом 3НФ приводила к некоторой неадекватности, была переработана Бойсом и названа НФБК (Нормальная Форма Бойса-Кодда). Позже Фейгин дал определение 4НФ и 5НФ, которые встречаются достаточно редко.

*Нормализация* – преобразование исходного отношения по определенным правилам и получение других отношений, совокупность которых эквивалентна исходному. Формы вложены друг в друга. Нахождение отношений в более старшей форме, в некотором смысле, более предпочтительно.

*Основной механизм нормализации* – декомпозиция исходного отношения (разделение исходного отношения на проекции).

Получение новых отношений не должно приводить к потере информации (возникновению противоречий), т.е. соединение полученных проекций должно дать исходное отношение.

**\*Декомпозиция без потерь и функциональные зависимости. Пример.**

**Теореме Хеза. Пример.**

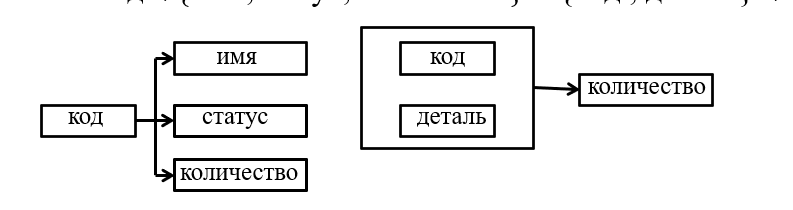
**Функциональная зависимость как семантическое понятие.**

**Диаграммы ФЗ.**

ФЗ – это особый вид ограничений целостности, т. е. это, семантическое понятие (отражающее смысл, логику организации предметной области).

Если отношение удовлетворяет ФЗ Код\_поставщика → Город, это значит, что каждый поставщик находится точно в одном городе. Такое ограничение существует в реальном мире (по крайней мере, в некоторой модели реального мира) и должно быть представлено в БД таким образом, чтобы оно могло быть приведено в действие СУБД. Для изображения ФЗ используется графическое изображение ФЗ диаграмма или схема ФЗ.

ФЗ: код→{имя,статус,количество} и {код , деталь}→количество



**Вспомогательные определения для процедур нормализации отношений.**

**ФЗ неприводимые слева**

Зависимости, отвечающие п.2. в определении неприводимых зависимостей, называются неприводимые слева. Это те зависимости, у которых нельзя слева опустить ни одного атрибута, чтобы не изменилось замыкание множества функциональных зависимостей (левая часть каждой ФЗ должна быть предельно простой). Неприводимые ФЗ и неприводимая слева ФЗ играют важную роль в нормализации.

Допущение: Для простоты изложения предполагаем, что каждое отношение имеет только один потенциальный ключ, который является первичным ключом.

**Неключевой атрибут отношения.** – это атрибут, который не входит в первичный ключ рассматриваемого отношения.

**Взаимно независимые атрибуты отношения.** Два или несколько атрибутов образующих множество А называются взаимно независимыми, если ни один из них не зависит функционально от какого-либо подмножества остальных атрибутов множества А.

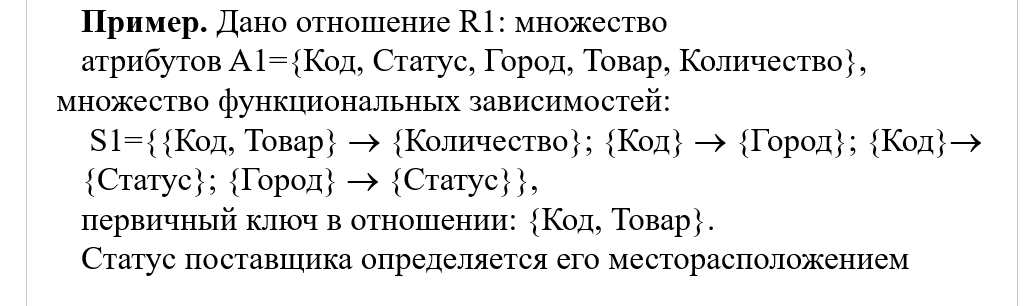
Физический смысл взаимно независимости: каждый атрибут из множества взаимно независимых атрибутов из множества А может быть обновлен независимо от остальных атрибутов этого множества.

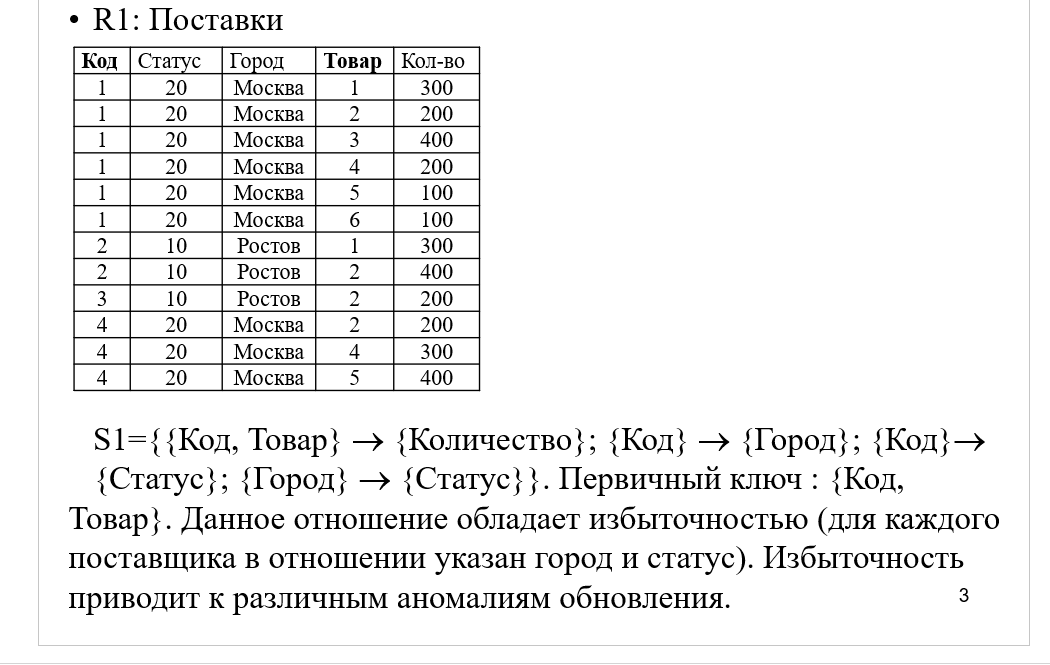
**Определение первой нормальной формы.**

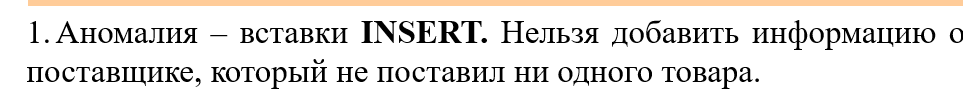
Отношение находится в первой нормальной форме тогда и только тогда, когда все используемые домены содержат только скалярные значения.

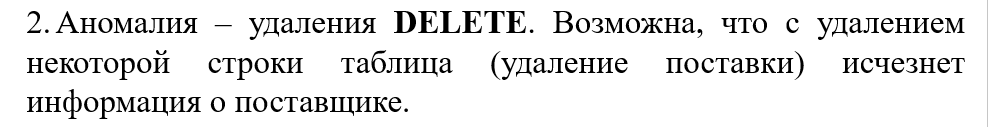
Любое нормализованное отношение находится в 1НФ.

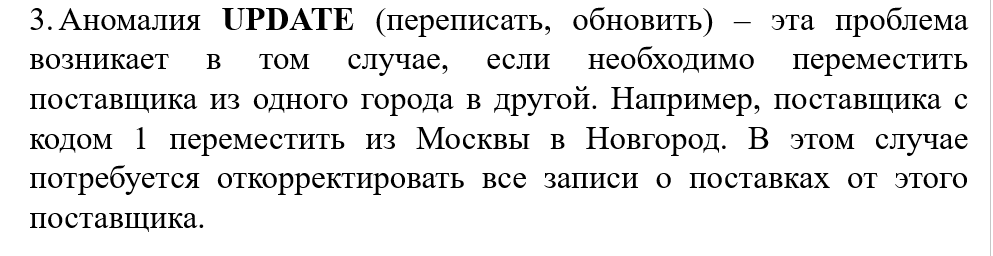
**\*Недостатки первой нормальной формы (1НФ) на примере.  
 Избыточность отношений в 1НФ.**

****

****

**аномалия – вставки в 1НФ.**

**аномалия – удаления в 1НФ.**

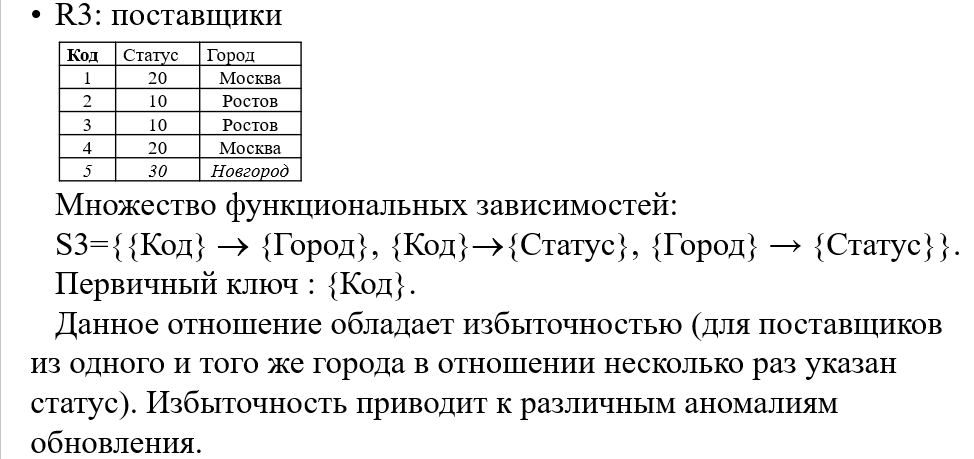
**аномалия обновления в 1НФ.**

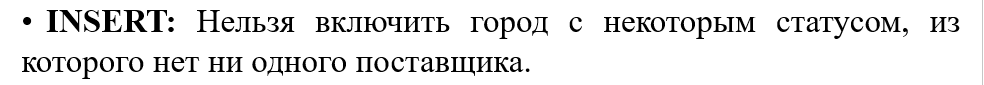
**Определение 2НФ и алгоритм перехода из 1НФ в 2НФ.**

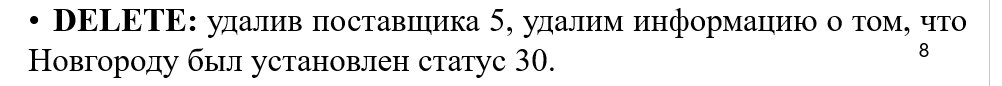
Определение 2НФ (при условии единственности потенциального ключа): Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в 1НФ и каждый его неключевой атрибут неприводимо зависим от первичного ключа.

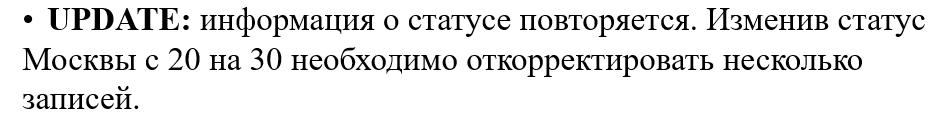


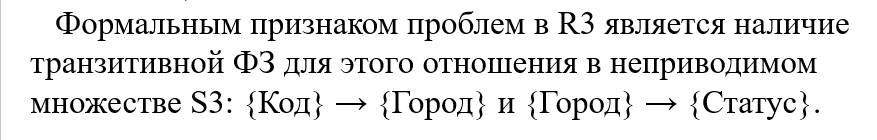
**\*Недостатки в 2НФ на примере.**

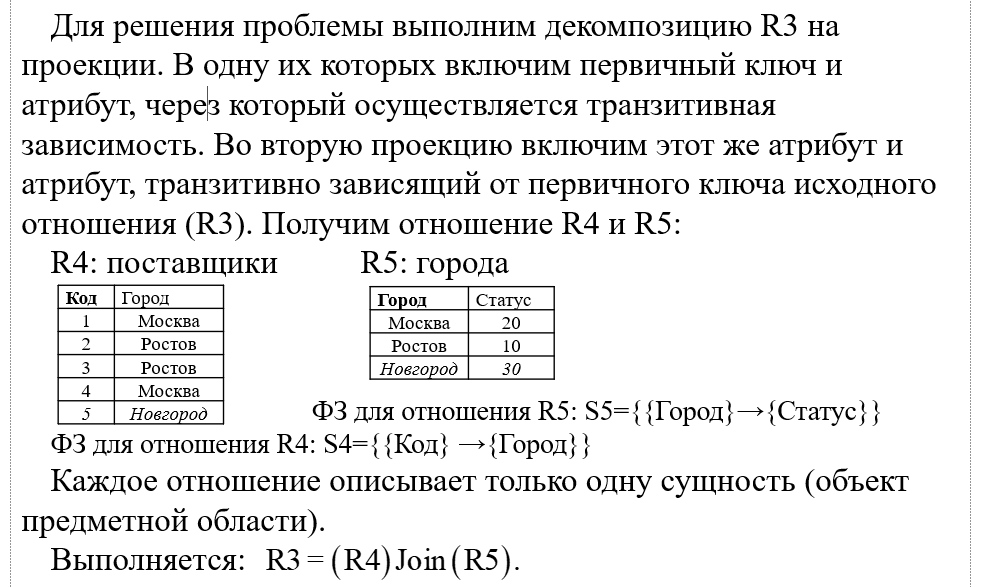
**Избыточность отношений в 2НФ.**

**аномалия – вставки в 2НФ.**

**аномалия – удаления в 2НФ.**

**аномалия обновления в 2НФ.**

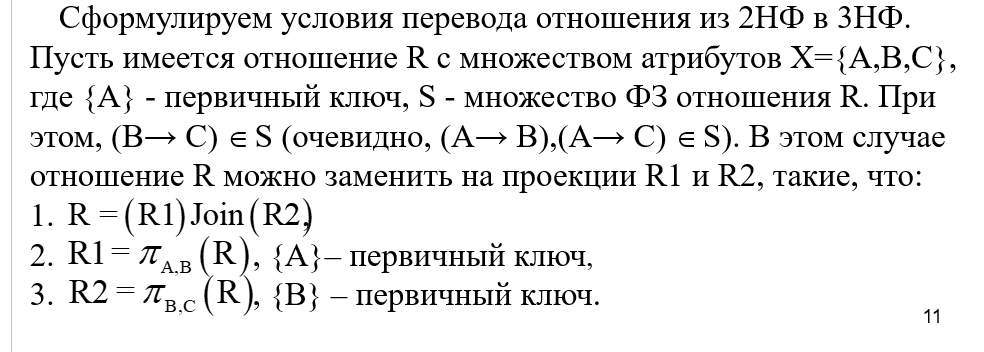
****

****

**Определение 3НФ и алгоритм перехода из 2НФ в 3НФ .**

Отношение находится в третьей нормальной форме тогда и только тогда, когда оно находится в 2НФ и каждый его неключевой атрибут не транзитивно зависим от первичного ключа.

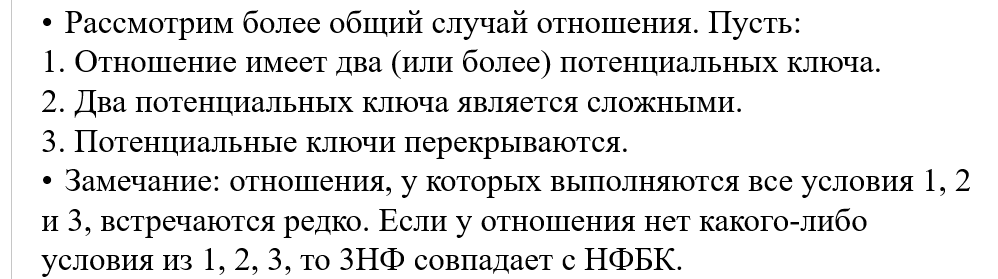
Нетранзитивная зависимость означает, что все неключевые атрибуты взаимно независимы. Т.е. неключевые атрибуты могут быть изменены не зависимо от изменения других неключевых атрибутов.



Замечание: Уровень нормализации данного отношения определяется семантикой, а не конкретными значениями данных в некоторый момент времени. Нельзя с первого взгляда на таблицу с данными для данного отношения определить, находится ли оно, например в 3НФ. Для этого также необходимо проанализировать существующие ФЗ. Даже при наличии всех ФЗ можно только высказать предположение, что данные не противоречат гипотезе о принадлежности отношения к 3НФ. Однако этот факт гарантирует, что предложенная гипотеза верна.

**\*Условия возникновения аномалий в третьей нормальной форме (3НФ). Пример возникновения аномалий в 3НФ. ?????**

**Нормальная форма Бойеса-Кодда. Преимущества НФБК.**

****

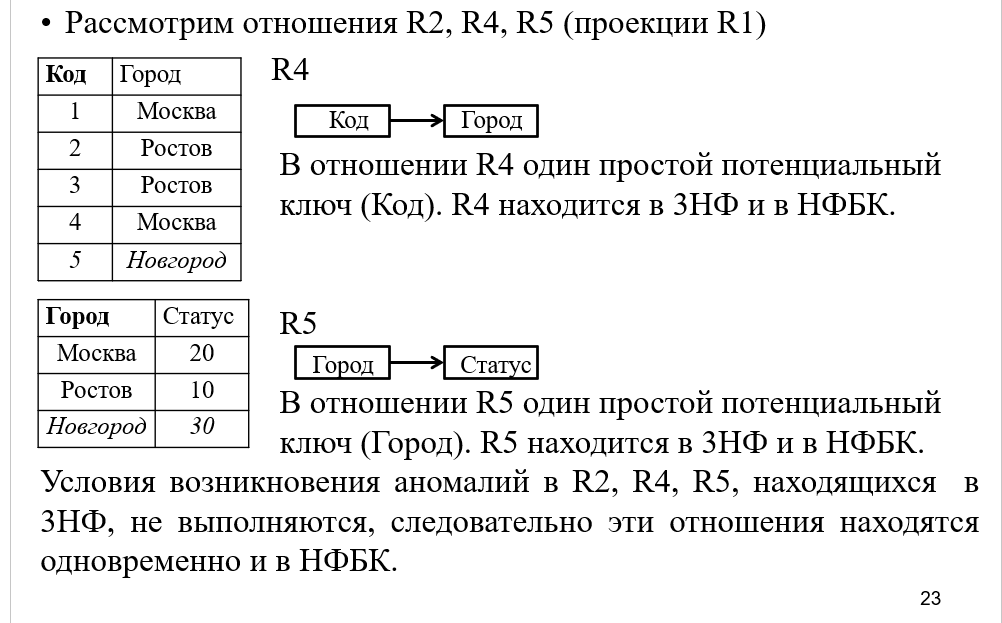
Определение. Отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда каждая нетривиальная и неприводимая слева ФЗ обладает потенциальным ключом в качестве детерминанта.

Менее формальное определение: Отношение находится в НФБК тогда и только тогда, когда детерминанты являются потенциальными ключами.

Преимущества НФБК:

Позволяет избавиться от некоторых проблем, присущих (хотя бы теоретически) форме ЗНФ

Определение НФБК концептуально проще ЗНФ. Отсутствует необходимость последовательно выполнять требований первой и второй нормальных форм (1НФ, 2НФ), единственности первичного ключа, использовать понятия «приводимость ФЗ слева», «транзитивная зависимость» и т.д

****

# 3 БЛОК

**Физическая организация данных. Формат записи**

Будем считать, что на физическом уровне информация хранится в файле, который содержит записи с идентичным форматом.

Формат записи – список имен полей, каждое поле занимает фиксированное число байт и имеет фиксированный тип. Запись состоит из значений каждого поля.

Над файлом требуется выполнить следующие типичные операции:

* включить запись;
* удалить запись;
* модифицировать запись;
* найти запись, удовлетворяющую заданным условиям;

**Эффективность физической организации** Эффективность физической организации определяется эффективностью реализации этих 4-х операций.

**Модель организации внешней памяти.**

Под внешней памятью чаще всего понимают магнитные диски, но возможны и другие устройства (CD–ROM, внешние накопители и др.).

Отвлекаясь от действительной организации и адресации внешней памяти (трехмерная адресация дисков H,T,B) предполагаем, что файловая система разделяет внешнюю память на логические блоки равного размера (кластеры). В каждом блоке могут храниться несколько записей (запись может занимать несколько блоков). Внутри блока может быть пространство, незанятое какими либо записями. Мы предполагаем, что файловая система устанавливает соответствие между именем файла и адресами блоков, которые образуют этот файл.

Можно считать, что запись так же имеет адрес, который можно рассматривать либо как абсолютный адрес её первого байта, либо как адрес блока, содержащего запись, + смещение записи внутри блока.

Отвлекаясь от действительной природы адресации, для ссылки на блок или запись будем использовать понятие «указатель», подразумевая – номер блока в некотором виртуальном линейном адресном пространстве, занимаемым некоторым файлом. Под указателем на запись иногда будем, так же, понимать её линейный адрес в некотором виртуальном линейном пространстве занимаемым файлом. Такая модель часто существенно искажает природу доступа к записи и под термином «указатель» на запись мы будем понимать «указатель» на блок, в котором находится запись.

Для того, чтобы найти нужную запись необходимо:

* прочитать блок, который её содержит;
* найти запись внутри блока.

При таком подходе запись внутри блока можно перемещать, при этом указатель на запись (фактически указатель на содержащий её блок) не изменится. Уменьшается вероятность «зависания указателя» – когда «указатель» указывает на место, где записи на самом деле нет.

Основная операция с внешней памятью – это передача блока из внешней памяти в оперативную память и наоборот. Поэтому, если говорить о быстродействии различных алгоритмов доступа к данным, то следует подсчитать число блоков, которые должны быть прочитаны в основную память или записаны из неё. Это число и будет представлять оценку быстродействия алгоритма.

Мы предполагаем, что записи блока (т.е. файла) имеют фиксированный размер и состоят из полей фиксированного размера и типа. В этом случае мы можем легко найти абсолютный адрес записи в файле.

**Закрепленные и незакрепленные записи.**

Для определения стратегии реализации файлов важно знать, являются ли записи файла «закрепленными» по некоторому фиксированному адресу, или нет.

Записи становятся закрепленными, если где-либо в базе данных могут существовать (хранится) указатели на них.

При закреплении записей мы не можем перемещать записи об этих объектах, иначе указатели на них «зависнут», т.е. не будут указывать на данные, на которые они указывали первоначально.

При закреплении записей не может быть реализован общий случай организации файла, при котором запись удаляется (например, при включении других записей или удалении записей). Так как, сложно найти, где именно в полной базе необходимо откорректировать указатели на перемещаемые записи.

**Организация файлов в виде «кучи».**

Наиболее очевидный подход к хранению записей файлов заключается в последовательном размещении их в необходимом числе блоков. Часто предполагается, что запись не может перекрывать границу блока (часть блока теряется). Эту организацию иногда называют «кучей».

Блоки, используемые для кучи, могут быть связаны в цепочку указателями (в конце блока указатель на следующий блок). При другой организации выделяют отдельный блок (блоки) в котором хранят список адресов блоков, образующих кучу.

Чтобы включить запись, её необходимо поместить в последний блок, если в нём имеется место, или получить новый блок из OS, если места в последнем блоке больше нет.

Удаление может осуществляться установкой бита удаления (перед/после каждой записью или в начале блока – пояснить).

Повторное использование пространства удаленных записей опасно, и может быть осуществлено, если записи не закреплены. Для освобождения пространства с удаленными записями используют процедуру «сборки мусора».

Для поиска нужной записи (удовлетворяющей заданным условиям, например, при заданном значении ключа) в общем случае требуется просмотреть все записи файла (в «среднем» половину записей файла). Если число блоков, образующих файл велико, то это такой поиск требует большого количества доступов к внешней памяти и, следовательно, приводит к увеличению времени доступа.

В следующих разделах лекций рассматриваются методы ускорения доступа к информации при организации поиска.

**Основной принцип организации хешированных файлов.**

Решается задача: по ключу найти запись.

Основная идея, лежащая в основе организации файлов с хешированным доступом, состоит в разделении записей файла между участками, каждый из которых содержит один или более блоков памяти. Для любого файла, хранимого таким образом, существует хеш-функция (h), которая использует в качестве аргумента значение ключа, и вычисляет целое число то нуля до некоторого максимального значения N.

Пусть v есть значения ключа. Тогда значение h(v) есть номер участка, в котором должна находиться запись с этим значением ключа, если она вообще присутствует в файле.

**Выбор функции хеширования.** При выборе функции h желательно, чтобы h(v) принимала все её возможные значения (от 0 до N) примерно с равной вероятностью, когда v принимает всевозможные допустимые значения ключа.

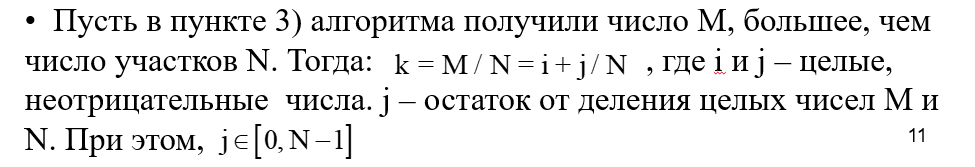
Выбор функции хеширования сложная проблема (есть много литературы). Во многих ситуациях полезна следующая стратегия:

Интерпретируем значение ключа как последовательность бит, сформированную путем конкатенации значений всех полей ключа. Эта последовательность имеет фиксированную длину, поскольку каждое поле имеет фиксированную длину.

Делим последовательность бит на группы, состоящие из фиксированного числа бит (например, из 16 бит). Последнюю группу при необходимости дополняем нулями.

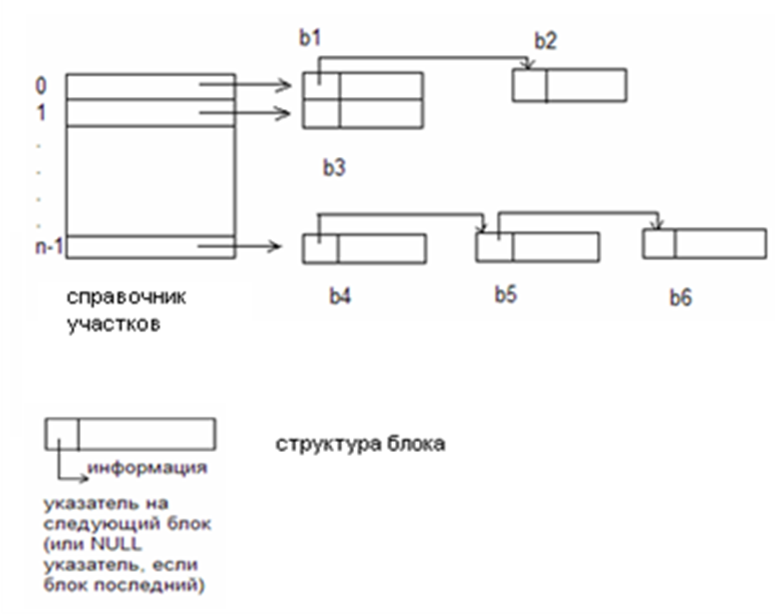
Складываем полученные группы бит как целые числа.

Делим полученную сумму на число участков и используем остаток от деления как номер участка.



**Схема организации хешированного файла.**

Если число участков мало, то справочник участков может находиться в основной памяти. В противном случае он сам (справочник) может быть разделен между некоторыми блоками и для организации поиска блока с нужной частью справочника участков необходим ещё один справочник (справочник справочника участков).



В каждом блоке предусмотрено место для размещения фиксированного числа записей. Если запись требует r байт, то для чтения, например, пятой записи внутри блока требуется сделать смещение 4r байт, от первого байта, следующего за заголовком блока.

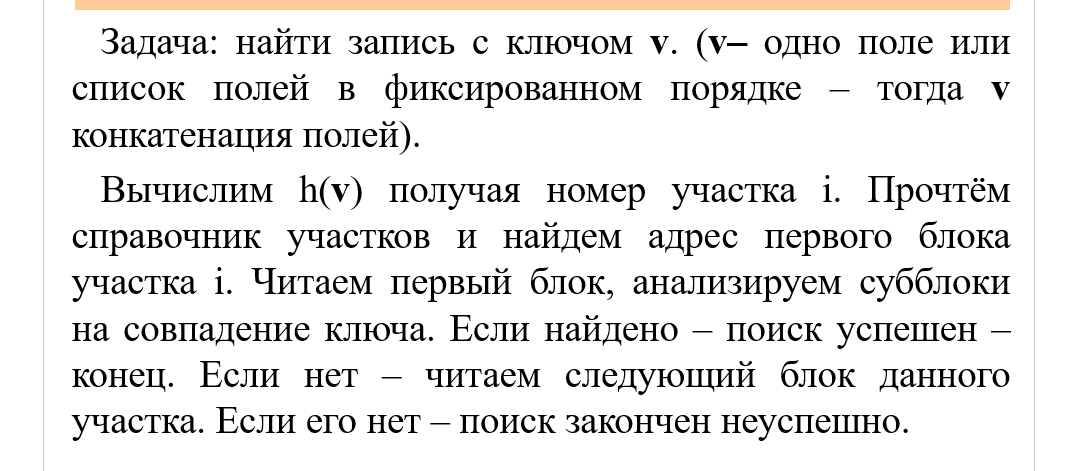
Пространство, используемое для хранения одной записи, будем называть субблоком.

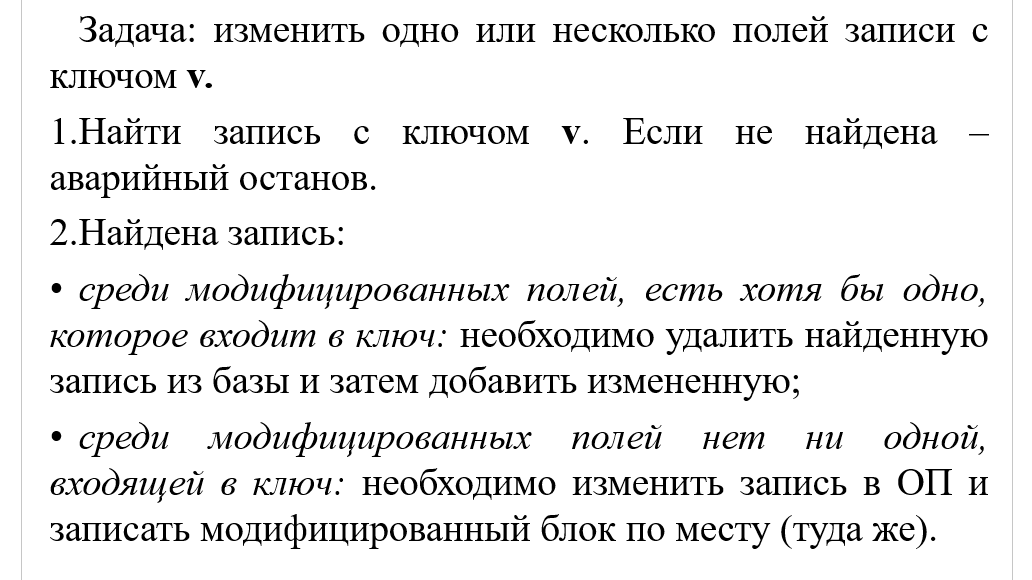
Во «время жизни» файла может оказаться, что некоторый субблок свободен (запись из него была удалена) в то время, как следующие за ним субблоки заняты. Для того, чтобы как-то различать записанные и свободные субблоки возможны два метода:

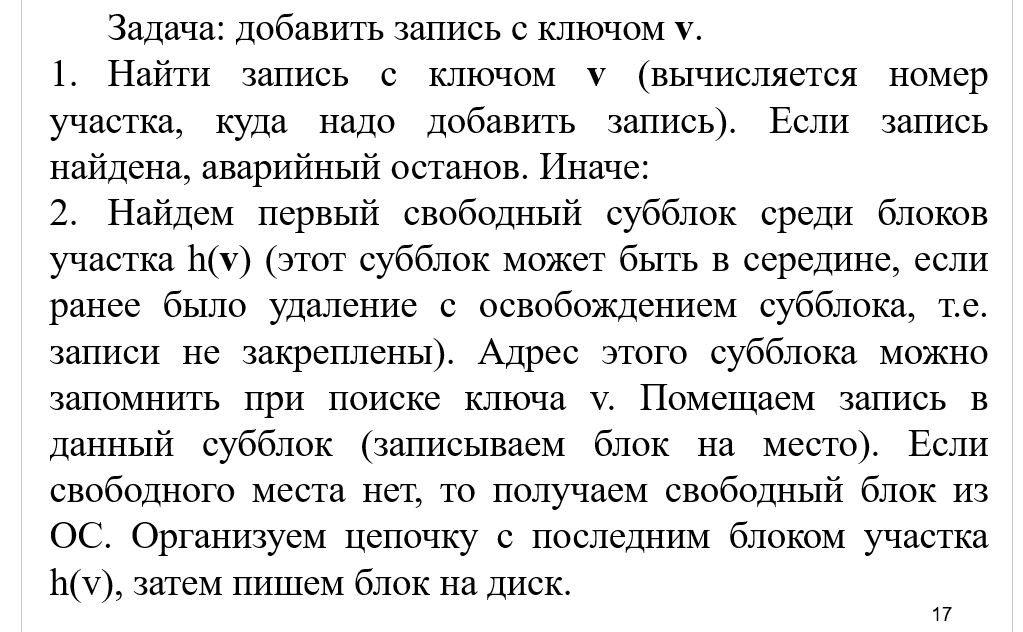
в освобождаемый субблок помещают последовательность бит, которая никогда не может появиться в записи;

в блоке отводят 1 бит на каждый субблок. Его нулевое значение – субблок свободен, 1– субблок занят.

Иногда отводят ещё один бит (в заголовке или в записи) который показывает, была ли удалена запись. Это позволяет избежать повторное использование субблоков, на которые была ссылка в случае закреплённых записей.

**Организация поиска в хешированном файле.**

**Модификация в хешированном файле. **

**Включение записи в хешированном файле. **

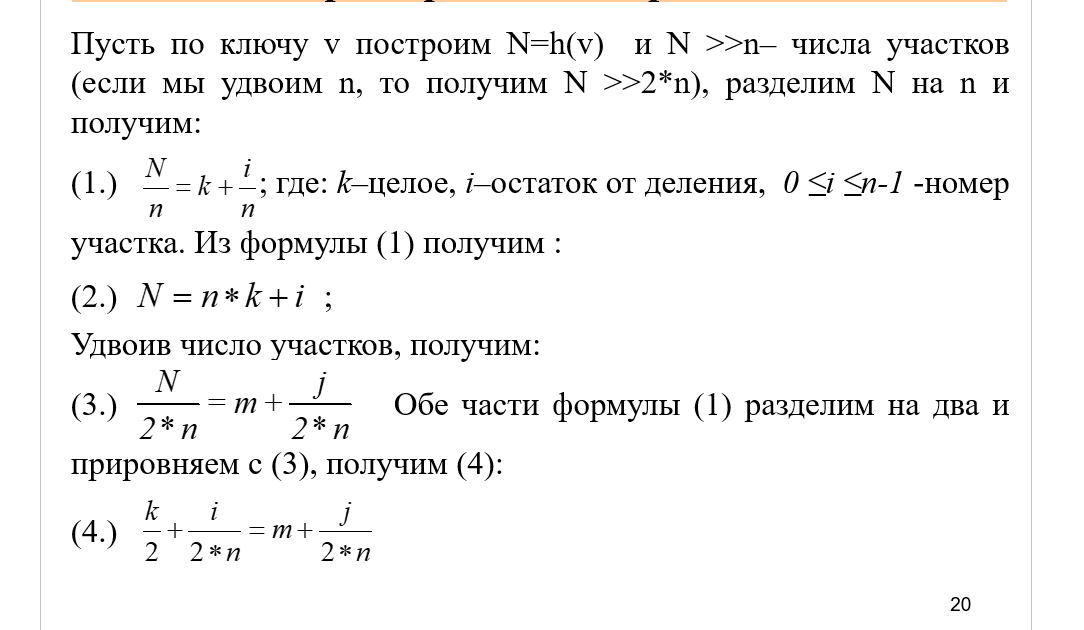
**Удаление в хешированном файле.** (в лекциях не описано, но частично похоже на алгоритм модификации, только теперь - просто удаление, без занесения измененных данных)

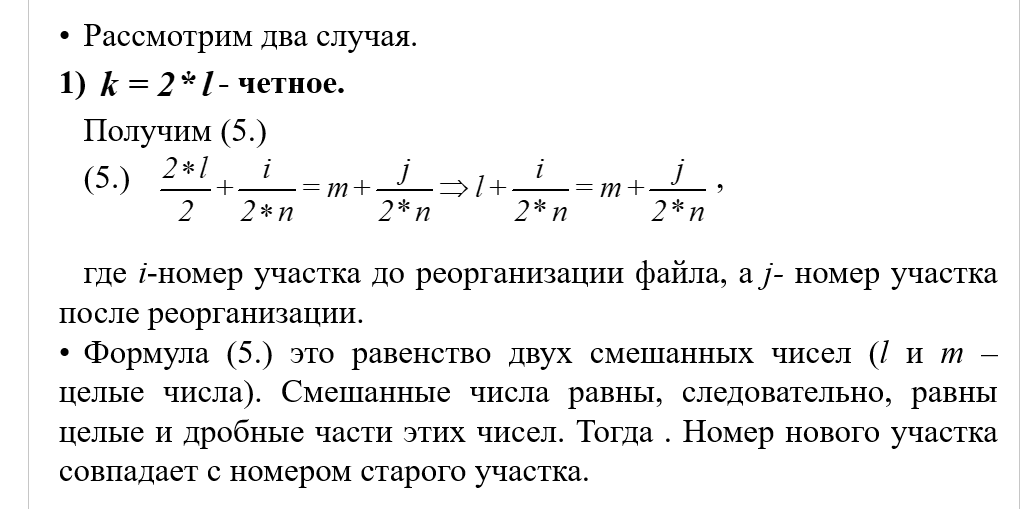
**Проблема реорганизации. Анализ временных характеристик хеширования.**

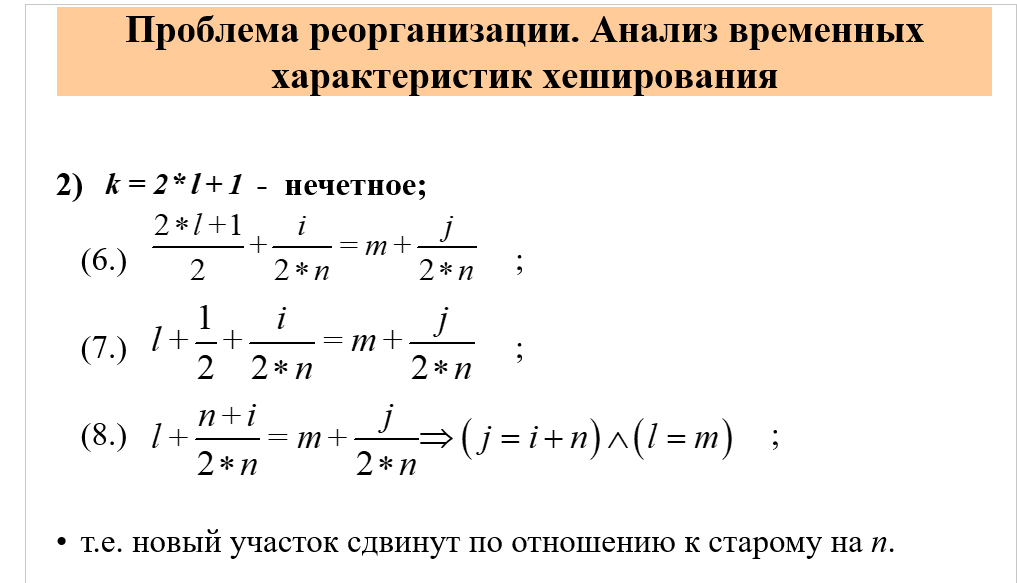
Для повышения скорости доступа, при росте числа блоков в участках, потребуется реорганизация файла. Её достаточно просто провести, если ввести два ограничения:

1. При вычислении функции хеширования от ключа v получают большое число, гораздо большее, чем число участков. Полученное число делят на число участков, остаток от деления – номер участка.
2. При реорганизации число участков n умножают на некоторое фиксированное с (обычно с = 2).

Если мы удвоим число участков, то все записи участка i будут попадать в участки i или i+n. И в эти участки не попадут никакие записи других участков.







**Индексированные файлы.**

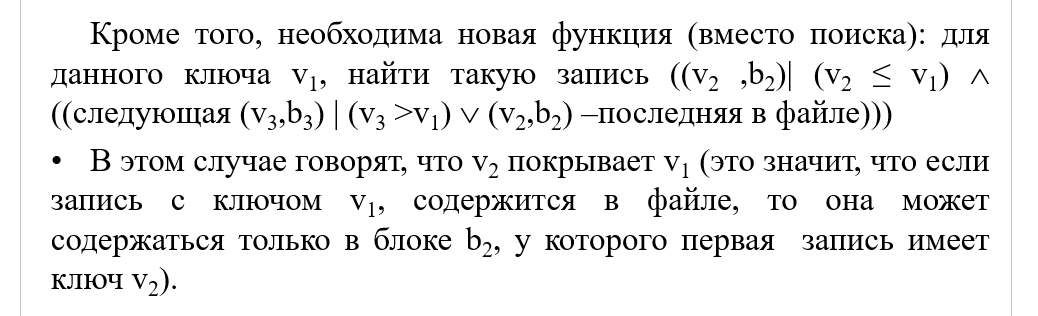
Решается та же задача: поиск записи по ключу v.

Допустим, что записи в файле отсортированы по значениям этого ключа (обратить внимание на сортировку дат dtos() и строк – лексикографическая сортировка). Для нескольких полей: сортировка по первому ключу, при равных первых – сортировка по второму и т.д.

В этом случае (файл отсортирован) для поиска можно использовать идею поиска в телефонной книге (словаре). Вместо просмотра всех записей будем анализировать только первую запись на каждой странице. Если искомый ключ превышает ключ этой записи, то возможно, что искомая запись находится на предыдущей странице. Если даже для последней страницы искомый ключ меньше, чем первый на этой странице, то следует проанализировать записи этой последней страницы.

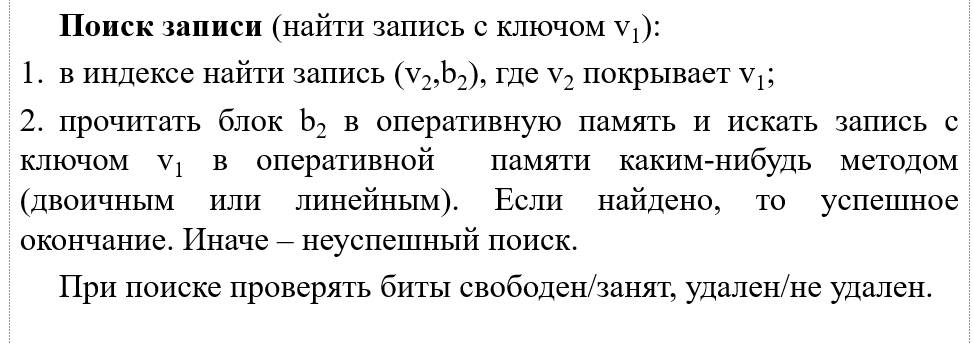
Аналогично организован доступ к файлу. Пусть мы имеем отсортированный файл с информацией, назовем его главным файлом. Создадим для главного файла второй файл– так называемый разреженный индекс, состоящий из пар (значение\_ключа, адрес\_блока).

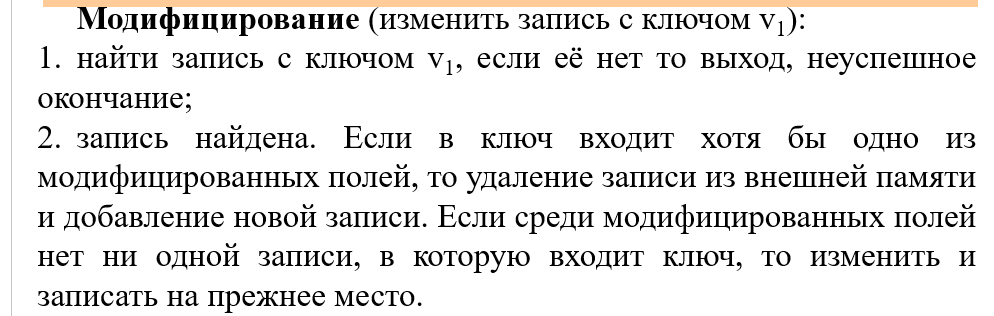
Пара (v,b) появляется в файле индекса, если первая запись в блоке главного файла с адресом b имеет значение ключа v. Записи файла индекса подобны обыкновенному (информационному) файлу с двумя полями: ключ и номер блока. Записи файла индекса отсортированы по значению ключа и не являются закрепленными, т.к. нигде в базе данных нет ссылки, ни на какую запись файла индекса. Вероятно, что с файлом индекса, как и с обычным (информационным) файлом следует выполнить операции включения, модификации, удаления.

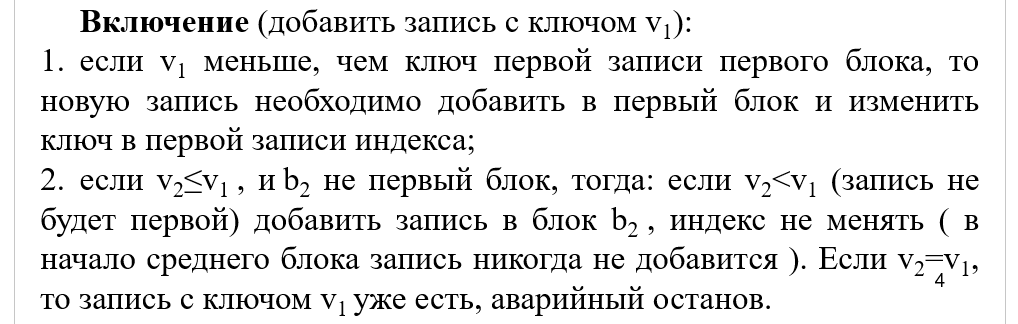


**Поиск в индексе.**

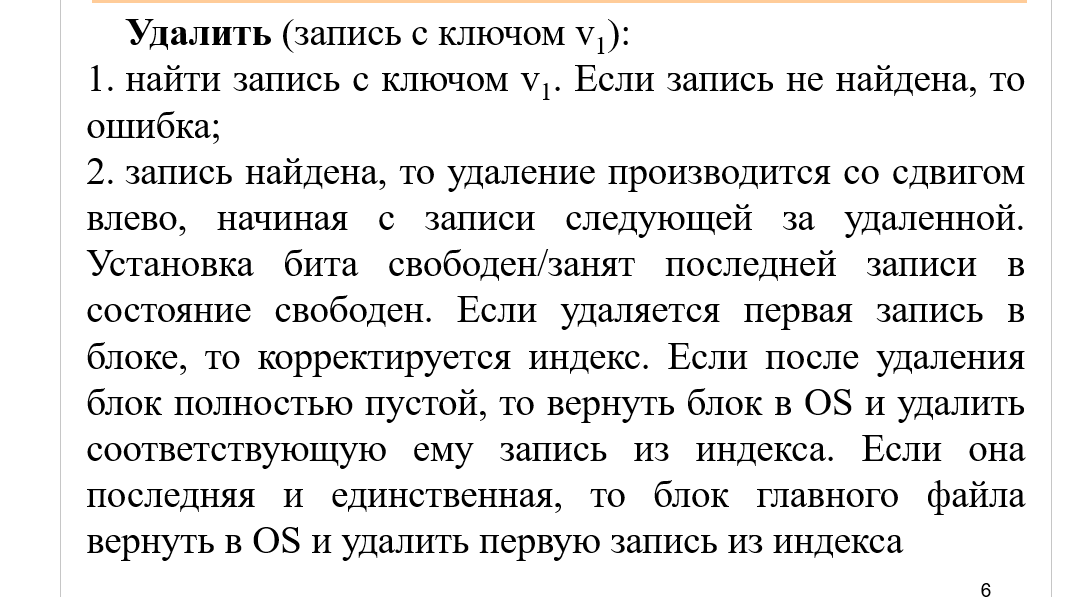
Операции (модификации, поиска, включения и удаления) над главным файлом требует таких же операций над файлами индекса. Записи файла индекса всегда не закреплены. Т.о. как выполняются операции над главным файле (у которого записи не закреплены), так и выполняются операции над файлом индекса.

**Поиск записи (сортированный файл с незакрепленными записями и индексом). **

**Модифицирование записи (сортированный файл с незакрепленными записями и индексом).**

**Включение записи (сортированный файл с незакрепленными записями и индексом).**

Т.о., если запись добавляется не в первый блок, то файл индекса никогда не изменяется. Если при добавлении в блоке есть место, запись занимает свободный субблок. Если места в блоке нет, тогда выполняют добавление новой записи в данный блок в ОП, записи сортируются в ОП. Далее, этот блок делят на две части, при этом последнюю запись выталкивают в следующий блок. Следовательно, индекс модифицируется. При обработке последнего блока (следующего блока нет), необходимо из OS получить новый блок и подключить его в конец файла.

**Удалить запись (сортированный файл с незакрепленными записями и индексом).**

**Организация сортированных файлов с закрепленными записями**

Если записи закреплены (пояснить), то невозможно (вообще говоря) поддерживать их внутри блока в отсортированном порядке. Нет возможности даже гарантировать, что записи предыдущего блока предшествуют записям последующего блока (относительно некоторого ключа). В этом случае каждый блок главного файла, на который имеется ссылка в индексе, играет роль первого блока в цепочке блоков (аналогично участку в хешированных файлах). При включении записей в участок добавляются новые блоки, которые объединяются в цепочку. Кроме того, создается еще один пустой участок, в который будут включаться записи, предшествующие первому участку. В этом случае индекс никогда не изменяется. Первые записи каждого блока первоначального файла определяют распределение записей по участку. Это распределение остается неизменным пока участки не достигнут больших размеров и потребуется реорганизация файла.

**Инициализация (сортированный файл с закрепленными записями и индексом).** процедура формирования файла индекса по первоначальному главному файлу называется – инициализацией.

Для инициализации отсортируем файл и распределим его записи по блокам. При этом в каждом блоке выделим некоторое свободное пространство (несколько свободных субблоков), которые при добавлении записи исключат на некоторое время рост цепочек в участках главного файла. Кроме того, предусмотрим один пустой блок в участке, предшествующем первому, куда будем помещать записи при расширении файла, и которые будут предшествовать записям первоначального файла. Создадим индекс для полученного файла, в том числе и для пустого блока. Запись в индексе для этого блока должна содержать номер «пустого» блока и не содержать ключа.

**Поиск записи (сортированный файл с закрепленными записями и индексом).**(найти запись с ключом v1 ):

Найдем запись индекса, значение ключа которой покрывает требуемое значение ключа v1. Если v1 меньше, чем первое значение ключа в файле индекса (заметим, что первичный ключ находится только начиная со второй записи индекса), то требуемой записью индекса является первая запись. По указателю на блок (из индекса) читаем первый блок цепочки (участка). Чтобы найти ключ v1 последовательно читаем блоки данного участка, соединенные в цепочку.

**Модификация записи (сортированный файл с закрепленными записями и индексом).**( изменить запись с ключом v1 ):

Выполняется аналогично предыдущей организации файла.

**Включение записи (сортированный файл с закрепленными записями и индексом).** (добавить запись с ключом v1 ):

Поиск участка (v2,b2) по ключу v1. Последовательно прочитать блоки участка с целью найти запись с ключом v1. Если такая запись найдена, то прервать добавления записи.

Параллельно с поиском ключа v1 искать свободное место. Если свободного субблока нет, то получить из OS пустой блок и добавить его в конец участка (пояснить). Включим новую запись в новый блок.

**Удаление записи (сортированный файл с закрепленными записями и индексом).** (удалить запись с ключом v1):

Найти запись с ключом v1 . Если запись не найдена, то ошибка, аварийное завершение. Если запись найдена, то установка бита удален в состояние «1» (запись удалена). Бит свободен/занят оставить в состоянии «занят».

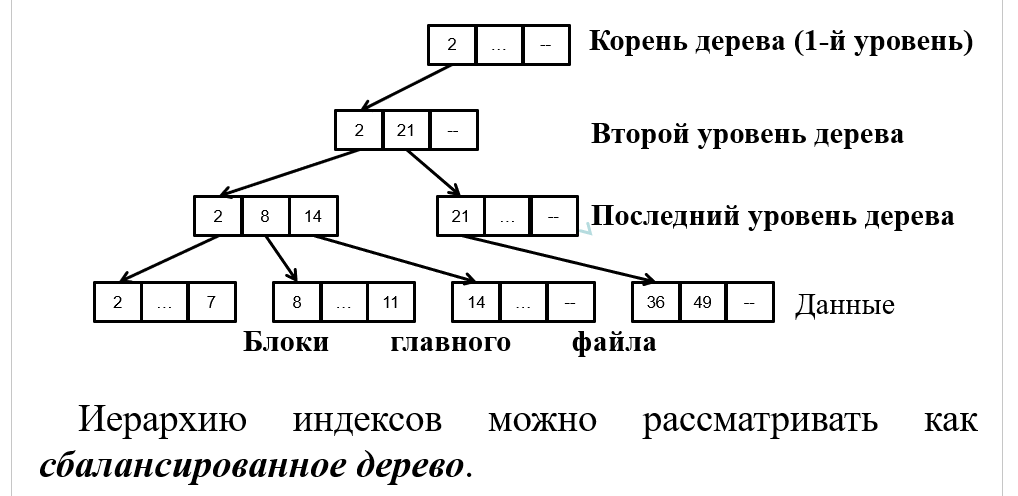
**\***

**Иерархия индексов в виде сбалансированного В–дерева.**

*Дать определения: «Дерево», «Сбалансированное дерево».*

Рассмотрим случай незакрепленных записей главного файла. Индекс – это файл с незакрепленными записями. Таким образом, для этого файла можно построить индекс индекса. Далее индекс этого индекса и т.д. пока индекс не поместится в один блок. Такая схема может быть более эффективна одноуровневого индекса и называется иерархией индексов.

Общая схема для очень больших файлов основана на иерархии индексов, соответствующей иерархической природе устройств внешней памяти. При этом, иерархию индексов можно организовать и независимо от иерархии структуры внешних устройств.



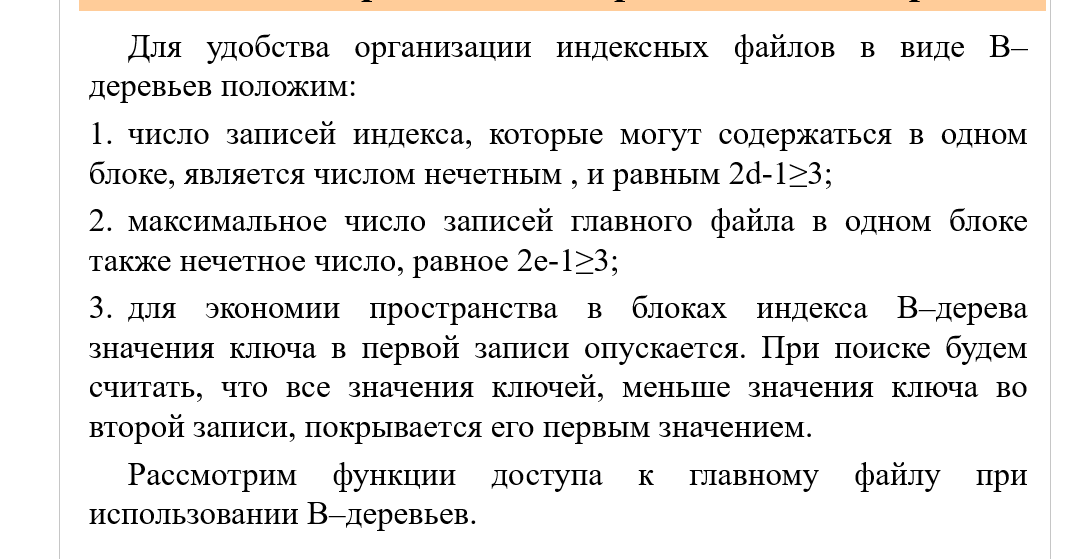
Операции поиска, модификации, включения, и удаления над многоуровневыми индексами являются простым обобщением ранее рассмотренных методов. Проблема – если индекс первого уровня превышает размер 1-го блока. Очевидно, что можно добавить ещё один уровень, что не меняет алгоритмы доступа. Таким образом, мы получим сбалансированное дерево, с неопределённым (переменным) числом уровней.

Напоминаем, что дерево сбалансированно, если длина пути от корня до любого листа одинакова.

Предполагаем, что записи главного файла не закреплены. Если записи закреплены, то можно, как и в предыдущем случае, считать, что листья в дереве являются первыми блоками участков.

Как и раньше будем считать, что в листе дерева стоит указатель на блок главного файла (случай, когда в листе стоит ссылка на запись главного файла, а не на блок, называется плотным индексом, название от того, что в блоках главного файла нет свободных мест (записи плотно распределены)).

Для операций включения и удаления над В–деревьями можно было бы использовать ту же стратегию, что и в предыдущем разделе, применяя операции включения и удаления к узлам (блокам) дерева на всех уровнях. Тогда узлы содержали бы от одной записи до максимально возможного их числа. Для В–деревьев обычно применяется другая стратегия включения и удаления, которая обеспечивает наполняемость всех узлов (за исключением корня) не менее чем на половину

**Соглашения, принятые для организации В–деревьев.**

**Поиск в B- деревьях.**(найти запись со значением ключа v):

Алгоритм поиска пути от корня В–дерева к некоторому листу, в котором будет находится требуемая запись, если она существует (рекурсивный алгоритм).

1. начало поиска от корня дерева. Устанавливаем текущий уровень дерева =1.

2. пусть рассматривается узел дерева с номером «i». Возможны две ситуации:

* i – лист (это легко проверяется подсчетом текущего уровня дерева от корня и сравнением этой величины полной длинной дерева). В этом случае проверим находится ли запись с ключом v в данном блоке (главного файла). Если да – успех, если нет – записи нет, конец алгоритма;
* i – не лист, следовательно, i → блок индекса. Определим, какое значение ключа в блоке покрывает v (учитывая, что первая запись блока не содержит ключа и первое значение покрывает все значения ключей, меньших значения ключа во второй записи). В блоке «i» для значения ключа, которое покрывает ключ v находим указатель на следующий блок.

3. увеличиваем текущий уровень дерева на 1 и повторяем данный алгоритм с пункта 2 для нового блока.

**Модификация файла с В-деревом.**

Модификация (изменить запись с ключом v):

1. Найти запись с ключом v. Если запись не найдена, то ошибка, завершение обработки;
2. Запись найдена. Если ни одно модифицируемое поле не входит в ключ, то откорректировать запись в ОП и записать изменения в тот же субблок внешней памяти. Иначе: существует хотя бы одно поле, которое входит в ключ: запись корректируется в ОП, удаляется из внешней памяти, а потом снова записывается в внешнюю память из оперативной.

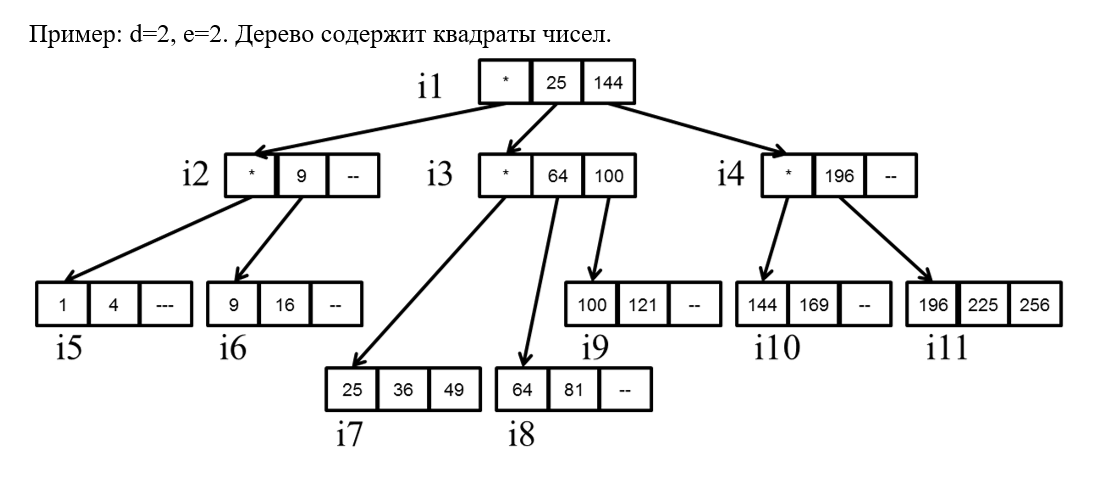
**Включение записи в файл с В-деревом** (добавить запись с ключом v):

1. Найти блок «i» , к которому эта запись относится. Если запись есть в блоке «i», то ошибка, завершение обработки.
2. В блоке i нет записи с ключом v:

* если в блоке меньше, чем 2е-1 запись, то запись просто включается в блок в отсортированном порядке. Эта запись никогда не может быть самой левой в блоке, если только блок «i» не является самым левым листом, т.е. нет необходимости изменять предшественника блока «i» (в индексе). Но и в этом случае (i – самый левый лист) предшественник по уровню не корректируется, т.к. первая запись блока индекса не имеет ключа;
* если в блоке 2е-1 записи (нет места), то создадим новый блок «ik». Разделим записи блока i с добавляемой записью между двумя блоками «i» и «ik», по «е» записей в каждый (было 2е-1. Добавили одну запись. Получим всего 2е записей);
* Пусть теперь «j» - отец блока «i» (j- известен, т.к. при поиске мы построили путь от корня к «i»). Применим данную процедуру включения к блоку «j» рекурсивно, но с константой d (в блок «j» добавлена запись о блоке «ik»). Если многие предки блока «i» заполнены полностью (по 2d-1 записей), то процедура расширения распространяется на несколько уровней и может достигнуть корня дерева. Если и он заполнен полностью, то его необходимо расщепить на два блока и получив у OS еще один блок создать новый корень, в котором будет две записи (т.о. размер дерева увеличится на единицу). При этом в корне может быть меньше d записей (если 2d-1>3).

**Удаление записи из файла с В-деревом** (удалить запись со значением ключа v:

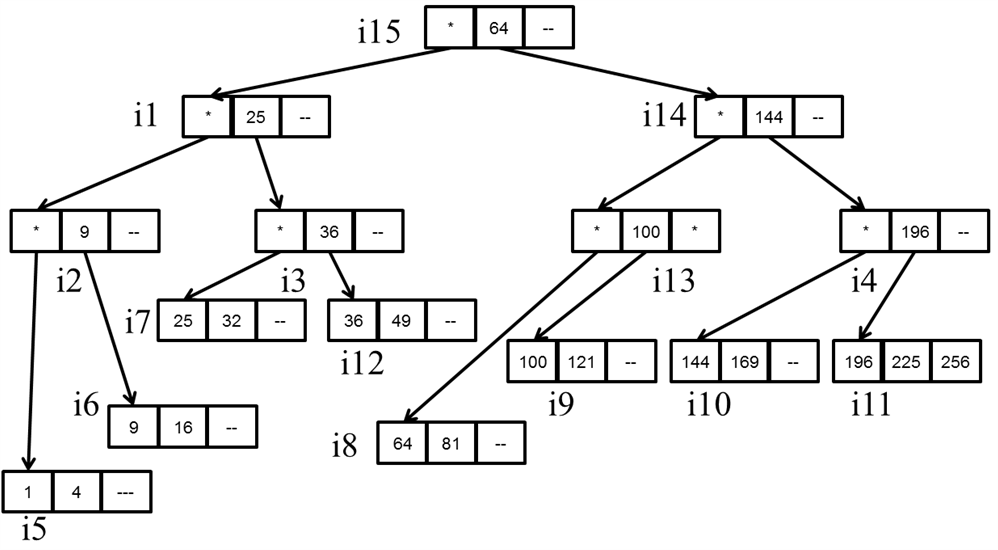
1. найдем путь от корня дерева к блоку i, содержащую запись с ключом v;
2. если записи в блоке i нет, то ошибка. Если запись есть, то удалим эту запись и пересортируем записи блока. Если после удаления в блоке не менее «е» записей, то, если эта запись первая, тогда возможно требуется изменить индексный файл на предыдущих уровнях. Однако коррекции не будет подвергаться отец блока «i» (т.к. у него первая запись не имеет ключа). Но возможно другие предки имеют запись с ключом v, которая не является первой. Т.о. от блока «i» следует двигаться в верх к корню, и в случае, если обнаружится такая запись, то её следует удалить, применив данную процедуру рекурсивно с константой d. (Хотя удалять и не обязательно, всё равно дерево будет работать как индекс).
3. Если после удаления в блоке «i» менее «е» записей (т.е. е-1 записей), то рассмотрим соседних братьев (в том же уровне), имеющих того же отца, что и блок «i». Если среди братьев блока «i» есть блок «j», с числом записей больше чем «е», тогда общее число k записей в блоках i и j принадлежит интервалу [2e,3e-2]. Минимум k=e-1+e+1=2e. Максимум k=e-1+2e-1= 3e-2. Распределим k записей между двумя блоками. В случае минимального k, в каждом блоке будет по e записей. В случае максимального k максимальное число записей в каждом блока будет не больше, чем (3e)/2-1<2e-1. Полученные от деления две группы записей поместятся в каждый из блоков. И их число в каждом новом блоке будет не меньше, чем e.Такое удаление потребует модификации и рекурсивного применения процедуры удаления к предкам «i» , заменив константу е на константу d. При этом, возможно, необходимо удалить корень и объединённые блоки на предыдущем корню уровне образуют новый корень – размер дерева уменьшается на единицу.

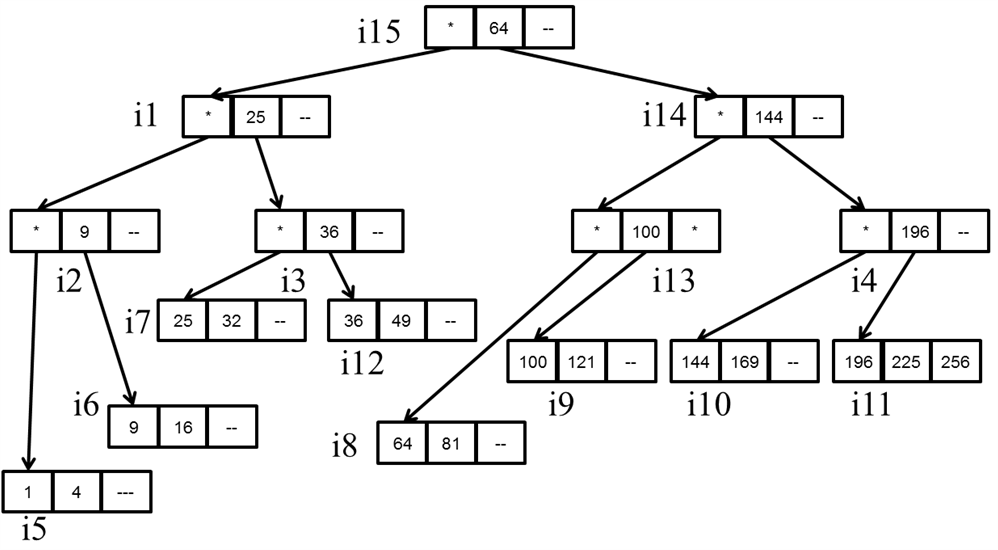
**Пример добавления записи в файл с В–деревом**

Добавим запись с ключом 32.

Ищем блок, в который необходимо добавить – это i7, но в нем нет места, значит делим его на 2: i7={25,32,-} и i12={36,49,-}. В i3 необходимо добавить 36. Делим i3 на два блока:

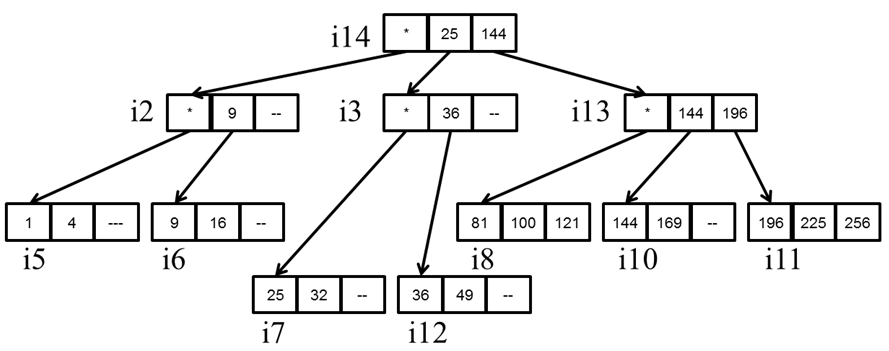
i3={\*,36,-} и i13={\*,100,-}. На втором уровне дерева появился еще один (четвертый) блок. Необходимо добавить еще один указатель в блок i1. Но там нет свободного субблока. Разделим i1 на два блока: i1={\*,25.-} и i14={\*,144,-}. Добавляем новый корень дерева (i15), увеличивая размер дерева не единицу. i15={\*,64,-}. Получаем:

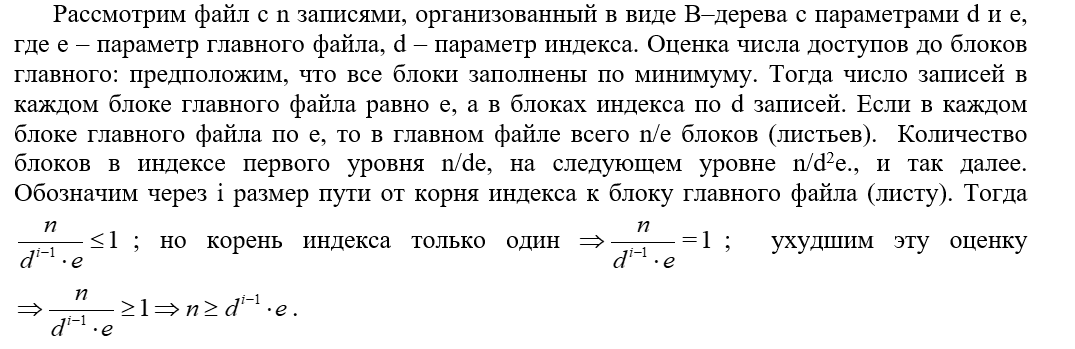


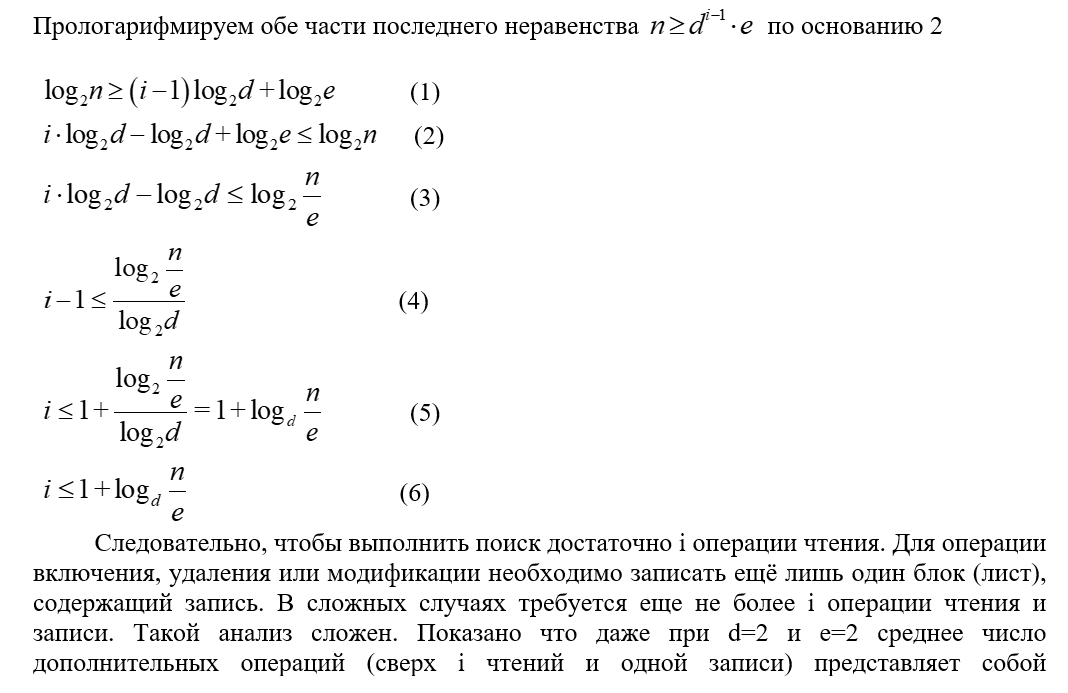
**Пример удаления записи из файла с В–деревом**

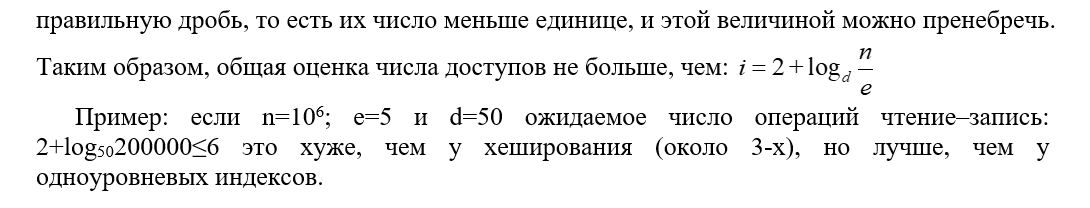
Удалим запись с ключом 64.

Найти блок с ключом 64. Это блок i8. Удаляем 64. Новый ключ – 81. Распространяем изменения вверх. В i15 ключа 64 заменяем на 81 (всегда изменяем ровно 1 ключ).Но в блоке i8 остаётся одна запись. Блок i8 заполнен меньше, чем на половину и имеется блок i9 (брат i8), в котором меньше, чем 2е-1 запись, т.е. 2. Соединяем его с блоком i8. У блока i13 единственный сын и у блока i13 есть брат, с числом записей меньше, чем 2е-1 запись. Это блок i4. Соединяем запись с ключом 100 из блока i13 с блоком i4. Теперь вторая ссылка в этом блоке должна быть 144. Блок i13 обладает указателями на i10 и i11. Значение ключа, сопровождающее указатель на i11 находится в i4, тогда как значение ключа 144 для блока i10 в i14. Вообще, если мы объединяем блоки при удалении, необходимое значение ключей содержатся либо в объединяемых блоках, либо в блоке, являющемся их общим отцом. При объединении i13 и i4 оказывается, что у i14 существует единственный сын, поэтому объединяем i14 с i1. теперь у i15 один сын. Т.к. он корень, то удаляем его. И т.д. Получаем:



**Анализ временных характеристик над В–деревьями**





**Файлы с плотным индексом**

Пусть нет необходимости поддерживать файл в отсортированном порядке. Допуская возможность хранения записей в произвольном порядке, можно избегать появления в главном файле большого числа частично заполненных блоков. При этом облегчается операция включения, для чего необходимо хранить только адрес последнего блока в файле и добавлять в него новую запись (при его заполнении получить адрес нового блока из OS). Если удаления производятся часто, то в файле появляются незаполненные места. Можно просто игнорировать тот факт, что определенные субблоки становятся при удалении свободными. (Напомнить технологию DELETE–PACK). Другой подход– использование отдельного файла с записями, содержащими одно поле, которое является указателем на блок с одним или более свободными субблоками (или хранить указатель на свободный субблок (используется в Oracle. Условные индексы).

При использовании несортированного файла проблема заключается в организации поиска записей по заданному значению ключа.

Для организации такого поиска используют еще один файл, который называется плотным индексом.

Одна запись плотного индекса соответствует одной записи главного файла. Структура записи индекса: (v,p); где v–ключ записи главного файла, р – указатель на запись главного файла с ключом v.

Для плотных индексов возможна многоуровневая организация в виде В–дерева. Отличие от разреженного индекса в том, что при том же размере дерева в плотном индексе листья – ссылки на записи главного файла, а в разряженном – блоки главного файла. Т.о.: при модификации, включении и удалении требуется на два доступа больше, чем в разряженном индексе (после чтения индекса надо читать еще блок главного файла и + еще записать при модификации - это недостаток).

Внимание! При использовании плотного индекса записи главного файла закреплены.

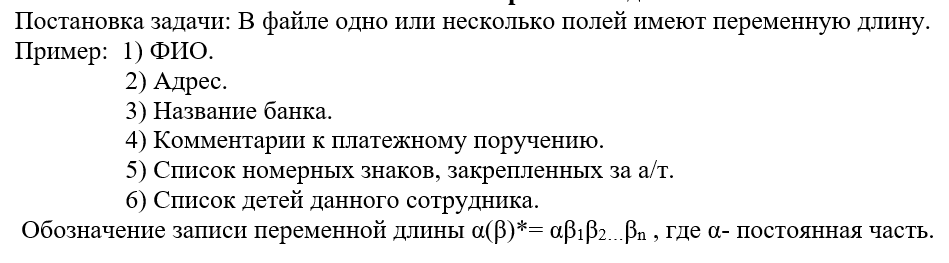
**Преимущества плотного индекса**

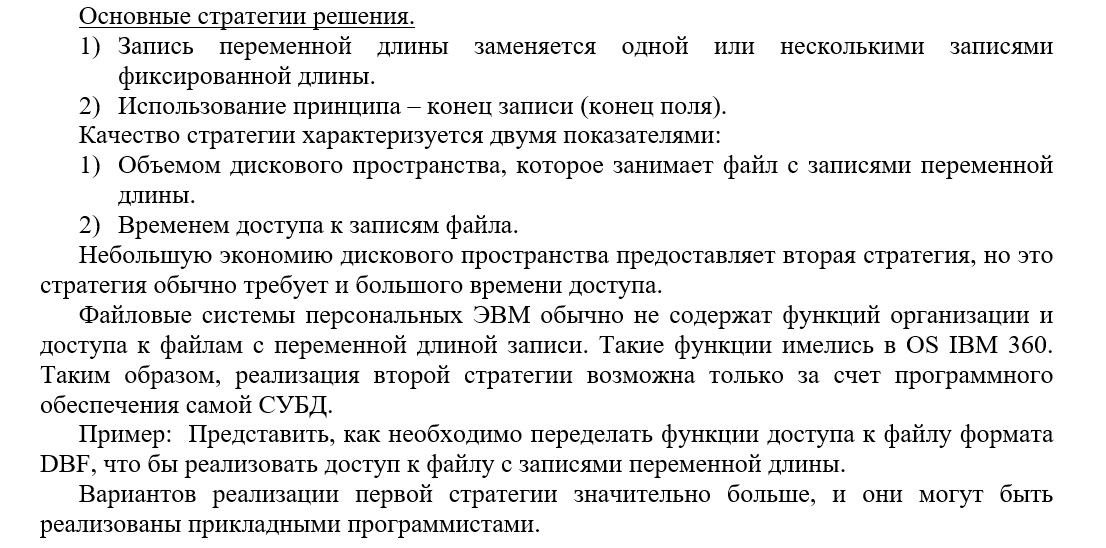
Преимущества плотного индекса.

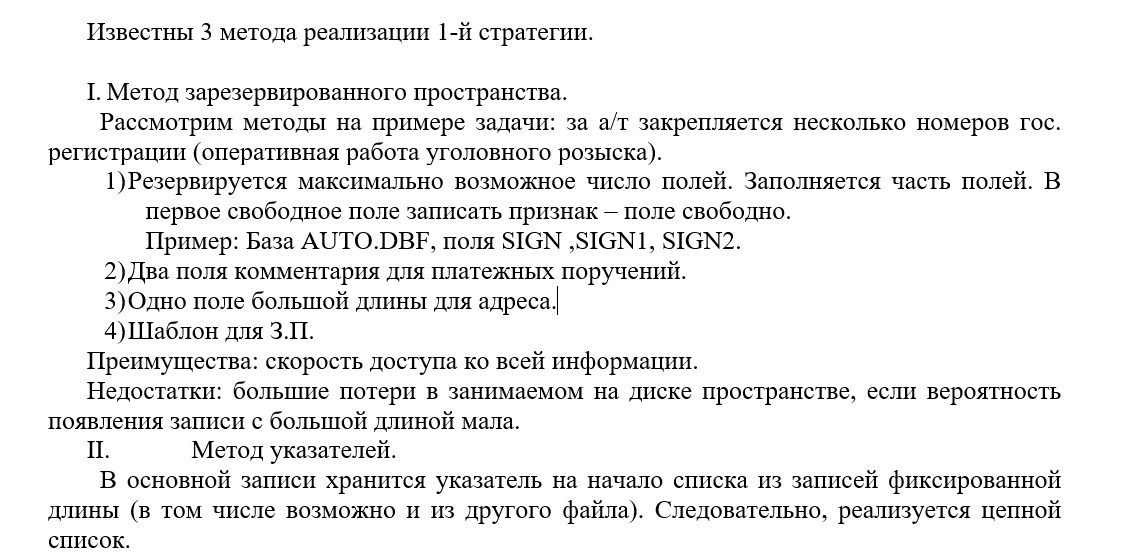
1) Записи главного файла могут быть закреплены, но записи индекса всегда не закреплены, следовательно, можно использовать более простую организацию и функции доступа к индексу. Нет понятия «участков» как в разряженном индексе.

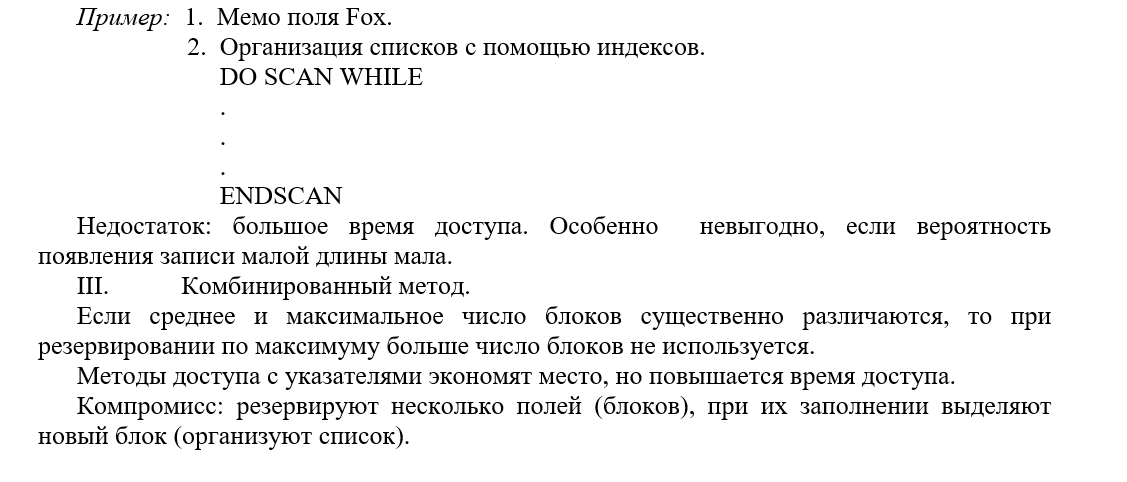
2) Если записи главного файла длинные, то общее число блоков меньше, чем у разряженного индекса, т.к. все блоки (кроме последнего) заполнены плотно.

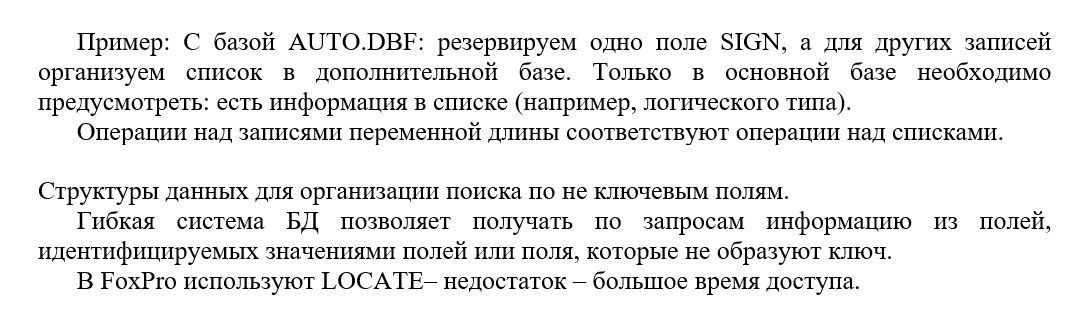
**Файлы с записями переменной длины**

****

****

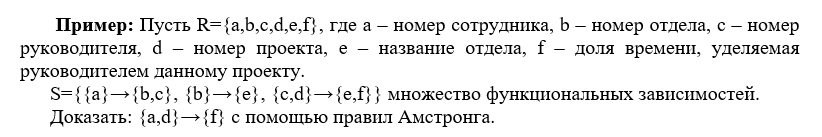
****

****

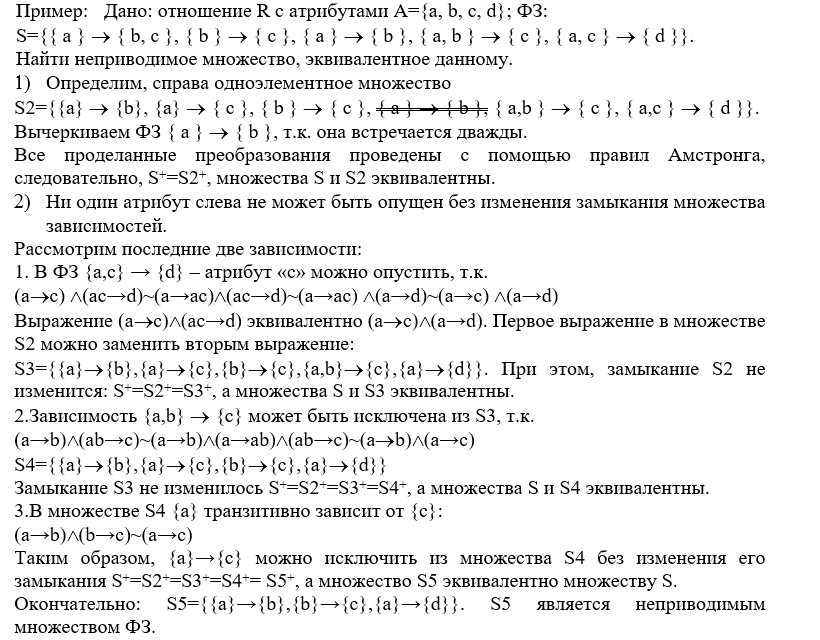
****

# ПРАКТИКА

**Доказать с помощью правил Армстронга**

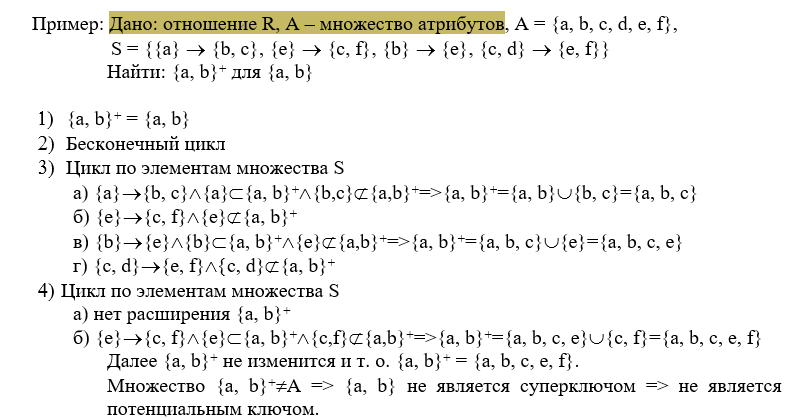
****

**(есть такое же задание, но с другим условием. Решение ниже)**

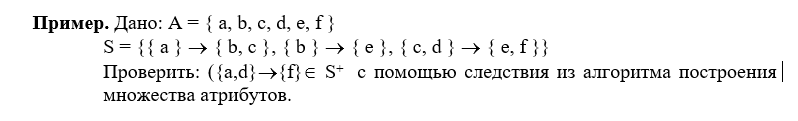
****

**Найти {a,b}+ для {a,b}**

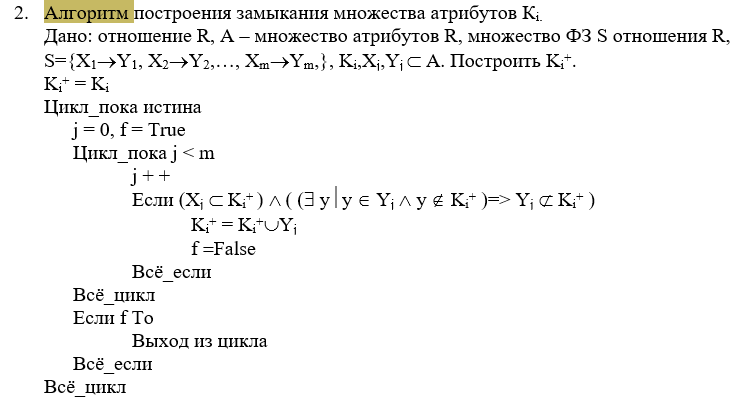
****

****

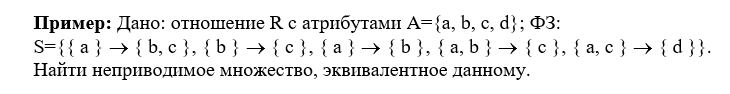
**Проверить с помощью следствия из алгоритма построения множества атрибутов**

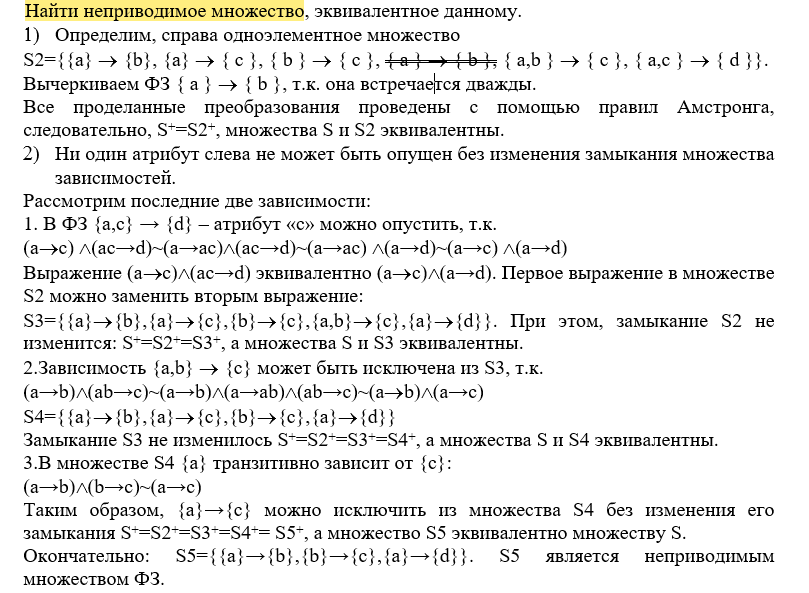
****

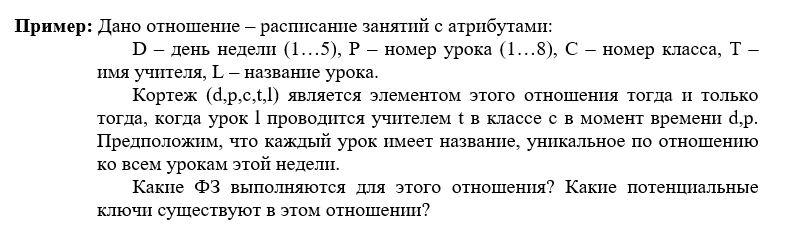
**(решения нет, есть алгоритм. Он такой же, что и у задания выше. Сам алгоритм ниже)**

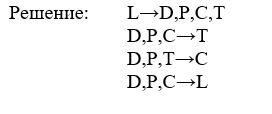
****

**Найти неприводимое множество**

****

****

**Найти ФЗ и потенциальные ключи**

****