

Министерство образования Республики Беларусь

Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Е.В. Калабухов

КУРС ЛЕКЦИЙ

по дисциплине «Базы данных, знаний и экспертные системы»

для студентов специальности I-40 02 01

«Вычислительные машины, системы и сети»

всех форм обучения

Минск 2007

## СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	6
1. СИСТЕМЫ БАЗ ДАННЫХ	7
1.1. Файловые системы	7
1.2. Системы баз данных	8
1.3. Трехуровневая архитектура ANSI-SPARC	11
1.4. Независимость от данных	15
2. СУБД	16
2.1. История развития СУБД (становление и поколения)	16
2.2. Функции СУБД	18
2.3. Компоненты СУБД	20
2.4. Архитектура многопользовательских СУБД	22
2.5. Преимущества и недостатки СУБД	24
3. ПРОЕКТИРОВАНИЕ БД	27
3.1. Задачи проектирования БД	27
3.2. Общая методология проектирования БД	28
4. МОДЕЛИ ДАННЫХ	34
4.1. Определение и классификация моделей данных	34
4.2. Концептуальные модели данных	35
4.2.1. Семантическое моделирование данных	35
4.2.2. ER-модель (модель типа «сущность-связь» или «объект/отношение»)	37
4.2.2.1. Концепции ER-модели	37
4.2.2.2. Структурные ограничения ER-модели	41
4.2.2.3. Проблемы ER-моделирования	42
4.2.3. EER-модель (Расширенная ER-модель)	43
4.3. Логические модели данных	46
4.3.1. Иерархическая модель данных	46
4.3.2. Сетевая модель данных	49

4.3.3. Реляционная модель данных	53
4.3.3.1. Реляционные объекты данных (структура)	55
4.3.3.1.1. Домены	56
4.3.3.1.2. Отношения	58
4.3.3.1.3. Представления	62
4.3.3.2. Целостность реляционных данных	63
4.3.3.2.1. Потенциальные ключи	64
4.3.3.2.2. Внешние ключи	65
4.3.3.2.3. Ссылочная целостность	66
4.3.3.2.4. NULL-значения	68
4.3.3.2.5. Третье средство обеспечения общей целостности в реляционной модели	69
4.3.3.3. Реляционные операторы	70
4.3.3.3.1. Реляционная алгебра	70
4.3.3.3.2. Реляционное исчисление	76
4.3.3.3.3. Связь реляционного исчисления и реляционной алгебры	79
4.3.3.4. Перевод ER-диаграммы в реляционную модель данных	80
4.3.3.5. Нормализация реляционных данных	83
4.3.3.5.1. Избыточность данных и аномалии обновления	83
4.3.3.5.2. Функциональные зависимости	85
4.3.3.5.3. Нормальные формы и схемы выполнения нормализации	87
4.3.3.6. Недостатки и пути развития реляционной модели	92
4.3.3.6.1. Недостатки традиционных реляционных систем	92
4.3.3.6.2. Объектно-реляционные СУБД	95
4.3.3.6.3. Нереляционные СУБД третьего поколения	97
4.4. Физические модели данных	100
4.4.1. Основные понятия физического хранения данных	101
4.4.2. Последовательные неупорядоченные и упорядоченные файлы	104
4.4.3. Хешированные файлы	106
4.4.4. Индексы	110

5. ЯЗЫКИ БАЗ ДАННЫХ	120
5.1. SQL	120
5.1.1. Общие сведения	120
5.1.2. Язык DDL SQL	122
5.1.2.1. Идентификаторы, типы данных, скалярные операторы	122
5.1.2.2. Создание структуры и объектов БД	124
5.1.3. Язык DML SQL	135
5.1.3.1. Оператор выборки данных SELECT	135
5.1.3.1.1. Общий формат оператора выборки	135
5.1.3.1.2. Секция WHERE - фильтр строк данных по условию	138
5.1.3.1.3. Обобщающие (агрегатные) функции	139
5.1.3.1.4. Группирующие запросы (секция GROUP BY)	141
5.1.3.1.5. Секция ORDER BY – сортировка результатов запроса	143
5.1.3.1.6. Подзапросы	144
5.1.3.1.7. Многотабличные запросы	147
5.1.3.2. Операторы UNION, INTERSECT и EXCEPT	149
5.1.3.3. Операторы изменения содержимого БД	150
5.1.3.3.1. Добавление (вставка) новых данных в таблицу	150
5.1.3.3.2. Изменение данных в таблице	152
5.1.3.3.3. Удаление данных из таблицы	153
5.2. QBE	154
6. ТРАНЗАКЦИИ	156
6.1. Основные определения	156
6.2. Параллельное выполнение транзакций	159
6.2.1. Проблемы параллельного выполнения транзакций	159
6.2.2. Пессимистические методы параллельного выполнения транзакций	164
6.2.2.1. Метод блокировок	164
6.2.2.2. Метод временных отметок	171
6.2.3. Оптимистические методы параллельного выполнения транзакций	172
6.2.4. Реализация изолированности транзакций средствами SQL	174

6.3. Восстановление данных	176
6.3.1. Влияние сбоев и отказов на БД	177
6.3.2. Средства защиты данных	179
6.3.3. Методы восстановления данных	181
7. ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ	186
7.1. Искусственный интеллект	186
7.2. Этапы и направления развития интеллектуальных систем	192
7.3. Экспертные системы	200
8. ЗНАНИЯ	222
8.1. Модели представления знаний	223
8.1.1. Модель на базе логики	225
8.1.2. Семантические сети	234
8.1.3. Фреймовые модели	239
8.1.4. Продукционные модели	245
8.2. Приобретение знаний	259
8.3. Работа с качественными знаниями	264
9. ЯЗЫКИ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ	274
9.1. РЕФАЛ	276
ЛИТЕРАТУРА	283

## ПРЕДИСЛОВИЕ

Предметом изучения курса «Базы данных, знаний и экспертные системы» являются два основных направления представления и обработки информации в современных компьютерных системах: системы баз данных и системы искусственного интеллекта.

Цель изучения данного курса – получение и систематизация знаний об основных идеях и методах работы с данными и знаниями.

Основу курса лекций составляют теоретические сведения, без знания которых невозможно практически грамотно использовать современные компьютерные системы данных направлений.

В рамках систем баз данных, предлагаются к изучению такие основные вопросы как:

- модели представления данных;
- методы проектирования баз данных;
- специализированные языки обработки данных SQL и QBE;
- методы параллельного выполнения транзакций и методы защиты данных;
- методы организации баз данных на низком уровне.

В рамках систем искусственного интеллекта, рассмотрены такие основные направления как:

- модели представления знаний;
- методы выделения знаний;
- проектирование и эксплуатация экспертных систем;
- языки систем искусственного интеллекта.

## 1. СИСТЕМЫ БАЗ ДАННЫХ

Базы данных (БД) – неотъемлемая часть современной жизни (наиболее значимые сферы применения: коммуникационные системы, транспорт, финансовые системы, наука). Дальнейшее расширение применения – практически везде, где есть необходимость хранения и поиска данных.

### 1.1. Файловые системы

Начало истории развития БД может характеризоваться следующими этапами: начальное накопление данных – книги; бумажные картотеки (карточки, папки) обычно индексированные по какому-либо признаку для ускорения поиска (например, картотека библиотеки) – ручные операции, эффективен только поиск по индексу (по фамилии авторов, названию книги) и практически невозможен по сложным критериям (найти среднее число книг написанных авторами с фамилией начинающейся на «А» и т.п.). Рост требований по поиску разнообразной информации (усложнение запросов, рост числа данных, требование быстрого ответа на запрос), а также рост мощностей и доступности вычислительной техники (особенно появление магнитных носителей (ленты, диски)) привело к появлению файловых систем.

Файловая система - набор программ, которые выполняют некоторые операции для пользователей (ввод данных, операции с файлами, генерация фиксированного набора специализированных отчетов), каждая программа определяет жестко свои собственные данные (структура и методы доступа) и управляет ими, все данные децентрализованы и хранятся в местах их обработки (например, по отделам предприятия). Данные в этих системах хранятся как наборы записей (record) в файлах, каждая запись содержит поля (field) хранящих определенные характеристики.

Ограничения файловых систем:

- разделение и изоляция данных – данные для обработки должны выбираться из нескольких файлов (сложность одновременной обработки данных из нескольких файлов);
- дублирование данных – (накопление файлов с данными по одному объекту в различных отделах) – неэкономное использование ресурсов (дисковое пространство) и риск нарушения целостности данных (ошибки, если данные в разных отделах различаются, требуются проверки данных);
- зависимость от данных – (физическая структура и способы доступа различны для разных приложений) сложно изменять файлы (добавить новое поле), для преобразования данных нужны специальные программы-конверторы и изменение приложений;
- несовместимость форматов файлов – (если приложения создаются с использованием различных языков программирования (COBOL, C, PASCAL)) необходима выработка общего формата (затраты времени)
- фиксированные запросы (рост количества приложений) – число запросов  $\sim$  числу приложений, рост запросов – рост числа приложений (снятие документирования и поддержки функциональности системы - усложнение сопровождения, снижение мер безопасности по защите).

Все ограничения файловых систем – следствие двух причин:

- определение данных содержится внутри приложений, а не отдельно от них,
- кроме приложений нет других инструментов для доступа к данным и их обработки.

## 1.2. Системы баз данных

Система баз данных (СБД) – компьютеризированная система для хранения информации в БД.

Компоненты СБД:



Г) Пользователи – делятся на четыре группы:

1) Администраторы:

- администраторы данных – отвечают за управление данными, включая планирование БД, разработку и сопровождение стандартов, бизнес правил (описывают основные характеристики данных с точки зрения организации) и деловых процедур, а также концептуальное и логическое проектирование БД;
- администраторы баз данных – отвечают за физическую реализацию БД, включая физическое проектирование, обеспечение безопасности и целостности данных, а также обеспечение максимальной производительности приложений и пользователей (более технический характер по сравнению с администратором данных);

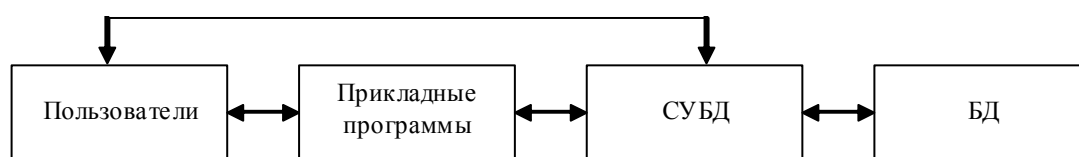


Рисунок 1. Компоненты СБД.

2) Разработчики баз данных:

- разработчики логической базы данных – занимаются идентификацией данных, связей между данными и устанавливают ограничения, накладываемые на хранимые данные - (ответ на вопрос ЧТО?);
- разработчики физической базы данных – по готовой логической модели создают физическую реализацию (формирование таблиц, выбор структур хранения, методов доступа, мер защиты) – (ответ на вопрос КАК?)

3) Прикладные программисты – создание приложений предоставляющих пользователям необходимые функциональные возможности (действия над базой данных);

4) Пользователи (клиенты БД):

- наивные пользователи – осуществляют доступ к БД через прикладные программы;
- опытные пользователи – могут осуществлять доступ к БД с использованием языков запросов или создавать собственные прикладные программы;

II) Прикладные программы – обеспечивают простой доступ к БД для пользователей, реализуются с использованием языков программирования 3-го (процедурные языки - C, COBOL, Fortran, Ada, Pascal) или 4-го поколения (SQL, QBE). 4-е поколение (“4GL”) - непроцедурные языки, возможно генерирование прикладного приложения по параметрам, заданных пользователем, делятся на: языки представления информации (языки запросов или генераторы форм и отчетов); специализированные языки (электронных таблиц и БД); генераторы приложений для определения, вставки, обновления или извлечения сведений из БД; языки очень высокого уровня для генерации кода приложений.

III) БД – совокупность логически связанных данных, хранящихся в компьютеризованной системе и отражающих некоторую предметную область человеческой деятельности. БД – единое, большое хранилище данных (набор интегрированных записей с самоописанием), содержит данные с минимальной долей избыточности, к которым может обращаться большое число пользователей. Описание данных называются системным каталогом или словарем данных, а сами элементы описания – метаданные (данные о данных) (обеспечивают независимость между программами и данными).

IV) СУБД – система управления базами данных – программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ. Главные преимущества СБД – преодоление ограничений файловых систем (в основном из-за обеспечения централизованного управления данными).

### 1.3. Трехуровневая архитектура ANSI-SPARC

Основная цель СУБД – показ пользователям данных в абстрактном представлении, т.е. сокрытие от пользователей особенностей хранения и управления данными.

Попытки стандартизации терминологии и общей архитектуры СУБД предприняты в 1971 группой DBTG (DataBase Task Group). Был предложен двух уровневый подход к архитектуре СУБД, выработанный на основе системного представления): схема (уровень администратора) и подсхемы (уровень пользовательских представлений). В 1975 комитетом планирования стандартов и норм SPARC (Standarts Planning and Requeirements Committee) Национального института стандартизации США ANSI (American National Standard Institute) была предложена необходимость трехуровневого подхода. Эта модель не является стандартом, но представляет собой основу для понимания функциональных особенностей СУБД.

Цель трехуровневой архитектуры – отделение пользовательского представления базы данных от ее физического представления по ряду причин:

- каждый пользователь должен иметь возможность обращаться к данным, используя собственное представление о них (независимо от представлений других пользователей);
- взаимодействие пользователя с базой не должно зависеть от особенностей хранения данных в ней (например, индексирование, хеширование);
- администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления;
- внутренняя структура БД не должна зависеть от изменений физических аспектов хранения информации (например, использование нового устройства хранения).

Архитектура ANSI-SPARC имеет три уровня: внешний, концептуальный

и внутренний.



Рисунок 2. Трехуровневая архитектура ANSI-SPARC.

Внешний уровень – представление БД с точки зрения пользователей (для каждой группы пользователей – описываются только необходимая часть БД). Т.е. каждый пользователь имеет свое представление о «реальном мире», и не видит лишние (с его точки зрения) данные. Кроме того, разные представления могут по-разному отображать одни и те же данные (например, дату). Также часть данных при этом может не храниться в БД (так называемые производные и вычисляемые данные), а получаться по мере надобности (например, возраст сотрудников), что не потребует лишних обновлений БД и уменьшит ее объем.

Концептуальный уровень – обобщающее представление БД (описывает, какие данные хранятся в БД, а также связи между ними). Это промежуточный уровень в архитектуре. Он содержит логическую структуру всей БД (с точки зрения администратора базы данных), а именно: все сущности, их атрибуты и связи; накладываемые на данные ограничения; семантическая информация о данных; информация о мерах безопасности и поддержки целостности данных. Все внешние представления получают из данных этого уровня, но этот

уровень не содержит никаких сведений о методах хранения данных (например, может быть информация о длине полей их типе, названии, но нет данных об объеме в байтах и т.п.).

Внутренний уровень – физическое представление базы данных в компьютере (описывает, как информация хранится в БД). Описывает физическую реализацию, и предназначен для достижения оптимальной производительности и обеспечения экономного распределения дискового пространства (содержит описание структур данных и описание организации файлов). На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы для работы с данными. Уровень содержит информацию о распределении дискового пространства, описание подробностей хранимых записей (реальная длина - байты), сведения о размещении записей, сжатии и шифровании данных.

Ниже внутреннего уровня находится физический уровень, который контролируется операционной системой (под руководством СУБД, причем разделение функций ОС и СУБД варьируется от системы к системе). Физический уровень использует только известные операционной системе элементы (например, указатели).

Общее описание БД принято называть схемой базы данных. Для каждого уровня архитектуры ANSI-SPARC существуют свои схемы. На внешнем уровне имеется несколько внешних схем или подсхем (для различных представлений данных пользователей). Для концептуального и внешнего уровня имеются соответственно концептуальная и внешняя схемы (для каждой БД существуют в единственном экземпляре). Концептуальная схема описывает все элементы данных, связи между ними, ограничения целостности и т.п. Внутренняя схема описывает определения хранимых записей, методов представления, описания полей данных, индексов и т.п.

СУБД отвечает за установление соответствия между этими схемами (проверка их непротиворечивости – например, чтобы можно было каждую внешнюю схему вывести на основе концептуальной схемы (на основе данных

внутренней схемы), проверить соответствие имен и т.п.). Концептуальная схема связана с внутренней посредством концептуально-внутреннего отображения (для поиска фактической записи на физическом устройстве хранения, которая соответствует логической записи в концептуальной схеме с учетом ограничений). Каждая внешняя схема связана с концептуальной схемой с помощью внешне-концептуального отображения (отображение имен пользовательского представления на соответствующую часть концептуальной схемы). Например, пользователь хочет получить данные из БД, при этом его запрос преобразуется к виду принятому на концептуальном уровне (отображение внешний-концептуальный, например, изменение имен полей для получения логического описания), затем логическое описание отображается на внутренний уровень (отображение концептуальный-внутренний) для доступа к реальным данным, затем производится обратный процесс.

Следует различать описание БД и саму БД (описание БД (содержание БД) – схема базы данных, создается в процессе проектирования (редко), а сама БД (детализация) – информация содержащаяся в таблицах может меняться часто). Совокупность данных, хранящихся в БД на определенный момент времени – состояние БД (состояний может быть различное множество).

Основное назначение трехуровневой архитектуры – обеспечение независимости от данных (т.е. чтобы изменения на различных уровнях никак не влияли на другие уровни).

Принятое в архитектуре ANSI-SPARC двух уровневое отображение на практике снижает производительность системы, но при этом поддерживает более высокую независимость от данных (для повышения эффективности есть возможность прямого отображения внешних схем на внутреннюю, но при изменениях внутренней схемы необходимо будет вносить изменения во внешние схемы и прикладные программы).

## 1.4. Независимость от данных

Независимость от данных – основополагающий принцип построения СБД. В соответствии с этим принципом, в системе должны поддерживаться отдельные представления данных для пользователя («логическое представление») и для системных механизмов среды хранения БД («физическое представление»). Такое разделение избавляет пользователя от необходимости знания принятого способа хранения БД и позволяет динамически в процессе эксплуатации системы оптимизировать способ хранения БД для обеспечения более высокой производительности системы и (или) более рационального использования ресурсов памяти.

Различают два типа независимости от данных:

- логическая независимость от данных – защищенность внешних схем от изменений, вносимых в концептуальную схему (добавление и удаление новых сущностей, атрибутов и связей должны осуществляться без изменений уже существующих внешних схем или изменения прикладных программ)
- физическая независимость от данных – защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему (изменения в структурах хранения, модификация индексов и т.п. должны осуществляться без изменения концептуальной и внешней схем, пользователи могут заметить изменения только по изменению производительности).

## 2. СУБД

СУБД (DBMS) – программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ. Фактически – прослойка между БД и пользователем (прикладной программой) для скрывания особенностей хранения и управления данными (абстрагирование).

### 2.1. История развития СУБД (становление и поколения)

СУБД выросли из файловых систем. Примерное начало становления СУБД – 60-е годы 20 века (нет данных о разработках других стран (СССР, Европа)):

- для управления данными американского проекта Apollo в начале 60-х создано программное обеспечение GUAM (North American Aviation (теперь Rockwell International)), в середине 60-х на базе GUAM создана первая коммерческая СУБД IMS (Information Management System) (NAA + IBM) – носители магнитная лента, иерархическая структура данных (подходило для управления иерархией частей проекта (компоненты → узлы → детали));
- в середине 60-х фирма General Electric создала систему IDS (Integrated Data Store) – сетевая СУБД (более сложные взаимосвязи, чем у иерархических СУБД, попытка создания стандарта баз данных).

Формирование стандартов БД – в 1965 на конференции CODASYL (Conference on Data System Languages) создана группа List Processing Task Force, переименованная в 1967 в DBTG (Data Base Task Group) – предложен стандарт в отчетах 1969, 1971 на сетевые БД (логическая организация данных + язык управления данными) – стандарт не одобрен ANSI, но на его основе разработано большое число систем (CODASYL или DBTG-систем).



DBTG-системы + системы на основе иерархического подхода – СУБД первого поколения (будут рассмотрены при изучении иерархической и сетевой модели данных), имеют ряд недостатков:

- для выполнения простых запросов требуют написания достаточно сложных программ;
- независимость от данных реализована в минимальной степени;
- отсутствие теоретических основ для описания (только технические стандарты).

В 1970 опубликована работа (E.F. Codd, IBM) о реляционной модели данных, устраняющей недостатки иерархической и сетевой моделей. На базе этой модели появилось множество экспериментальных СУБД. Первые коммерческие реляционные СУБД – конец 70-х – начало 80-х (экспериментальная СУБД System R (IBM, Сан-Хосе, Калифорния) – создана для проверки реляционной модели, в ходе проекта создан язык SQL; СУБД DB2 (IBM); Oracle (Oracle Corporation)). Реляционные СУБД относятся к СУБД второго поколения.

Реляционная модель также имеет ряд недостатков, один из них – ограниченные возможности моделирования. Наиболее значимые работы по устранению этого недостатка реляционной модели (в области семантического моделирования данных – исследований о способах представления смыслового значения, о модели более точно описывающей реальный мир):

- 1976, Чен предложил модель «сущность-связь» (ER-модель) – технология проектирования баз данных (будем рассматривать);
- Кодд предложил расширенные версии реляционной модели (RM/T (1979) и RM/V2 (1990)).

В связи с возрастанием сложности приложений БД появились новые системы: объектно-ориентированные СУБД (OODBMS) и объектно-реляционные СУБД (ORDBMS). Они представляют собой СУБД третьего поколения, однако структура этих моделей пока еще определена не окончательно (не совсем ясно).

## 2.2. Функции СУБД

Для СУБД характерно наличие следующих функций и служб (сервисов) (первые восемь - Кодд, 1982):

- СУБД должна предоставлять пользователям возможность сохранять, изменять и обновлять данные в БД (основная функция СУБД, способ реализации этой функции должен скрывать от пользователя детали физической реализации системы (абстрагирование));
- СУБД должна иметь доступный конечным пользователям каталог, в котором хранится описание элементов данных (т.н. системный каталог – хранилище информации, описывающей данные БД (метаданных): имена, типы и размеры элементов данных; имена связей; накладываемые на данные ограничения поддержки целостности; имена санкционированных пользователей; описание схем архитектуры БД; статистические данные (счетчики событий)); преимущества использования системного каталога: централизованный контроль доступа к данным; определение смысла данных для понимания их предназначения; упрощается определение владельца данных или прав доступа к ним; облегчается протоколирование изменений БД; последствия изменений могут быть определены до их внесения в БД (усиление безопасности и целостности данных);
- СУБД должна иметь механизм, который гарантирует выполнение либо всех операций обновления данной транзакции, либо ни одной из них (транзакция – набор действий выполняемых пользователем (прикладной программой) с целью доступа или изменения БД (при этом в БД вносятся сразу несколько изменений)); если во время внесения изменений происходит сбой (например, вносимые данные вызывают нарушение целостности данных), то все изменения должны быть отменены (возвращение в непротиворечивое состояние БД));

- СУБД должна иметь механизм, который гарантирует корректное обновление БД при параллельном выполнении операций обновления многими пользователями (этот вопрос связан с поддержкой транзакций);
- СУБД должна предоставлять средства восстановления БД на случай ее повреждения или разрушения (этот вопрос связан с поддержкой транзакций);
- СУБД должна иметь механизм, гарантирующий возможность доступа к БД только санкционированных пользователей (скрытие части ненужных данных и защита от любого несанкционированного доступа);
- СУБД должна обладать способностью к интеграции с коммуникационным программным обеспечением (для поддержки связи с пользователями);
- СУБД должна обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам (целостность БД – корректность и непротиворечивость хранимых данных, один из типов защиты БД; целостность выражается в виде ограничений или правил сохранения непротиворечивости данных (если есть противоречия – изменения не вносятся в БД));
- СУБД должна обладать инструментами поддержки независимости программ от фактической структуры БД (независимость достигается за счет реализации механизма поддержки представлений (см. п. 1.3. Архитектура ANSI-SPARC), однако достичь полной логической независимости от данных очень сложно, т.к. обычно система легко адаптируется к добавлению нового объекта (атрибута, связи и т.п.), но не к их удалению (в некоторых системах вообще запрещено вносить изменения в существующие компоненты логической схемы));
- СУБД должна предоставлять некоторый набор различных вспомогательных служб (в основном для эффективного администрирования БД: импорт данных, мониторинг, стратегический анализ, реорганизация индексов, сборка мусора и перераспределение

памяти).

### 2.3. Компоненты СУБД

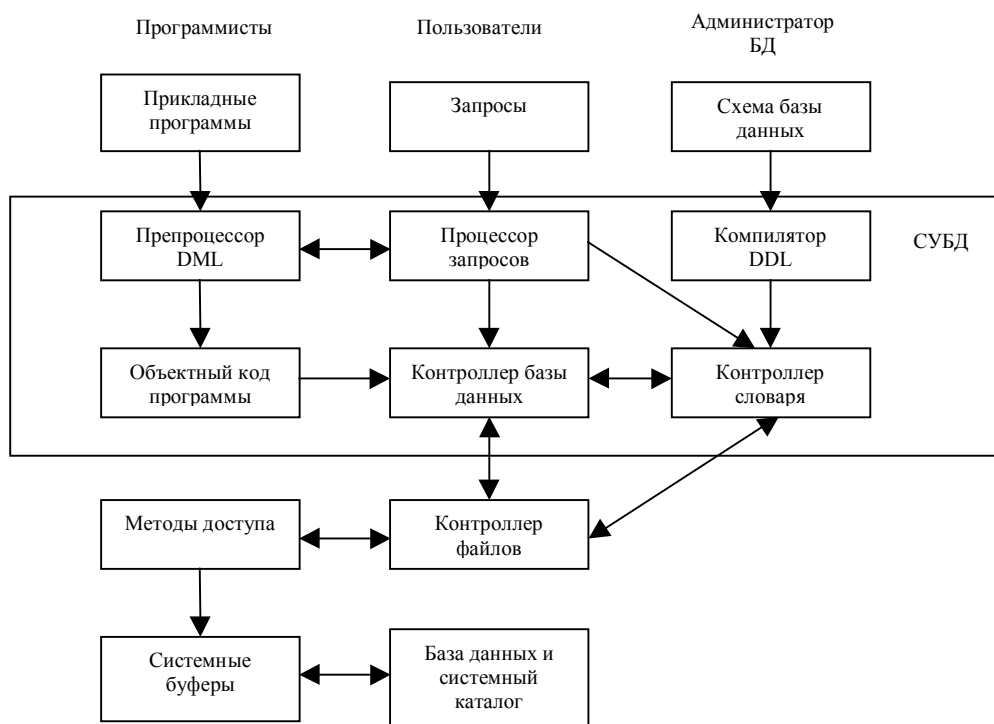


Рисунок 3. Основные компоненты типичной СУБД.

СУБД состоит из программных компонентов (модулей), каждый модуль выполняет одну специфическую операцию (часть операций может поддерживаться операционной системой, но только базовые и контролируемые СУБД (как надстройкой над ОС)).

Компоненты типичной СУБД:

- процессор запросов – основной компонент СУБД – преобразует запросы в последовательность низкоуровневых инструкций для контроллера базы данных;
- контроллер базы данных – принимает запросы, проверяет внешние и

концептуальные схемы для определения тех концептуальных записей, которые необходимы для выполнения запроса, в него входят обычно следующие основные компоненты: контроль прав доступа, процессор команд, средства контроля целостности, оптимизатор запросов, контроллер транзакций, планировщик, контроллер восстановления, контроллер буферов (память-диск);

- контроллер файлов – манипулирует файлами БД (создает и поддерживает список структур и индексов, определенных во внутренней схеме, но не управляет вводом-выводом данных непосредственно, а лишь передает запросы соответствующим методам доступа, которые обмениваются данными между системными буферами и диском);
- препроцессор языка DML – преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка (взаимодействуя с процессором запросов); DML (Data Manipulation Language) – язык, содержащий набор операторов для поддержки основных операций манипулирования содержащимися в базе данными (вставки, модификации, извлечения и удаления данных); DML может быть двух типов: процедурный – указывает, как можно получить результат оператора DML (обычно оперирует данными по одной записи), непроцедурный – описывает, какой результат будет получен (обычно оперирует наборами записей - SQL);
- компилятор языка DDL – преобразует DDL-команды в набор таблиц, содержащих метаданные (сохраняются в системном каталоге и частично в заголовках файлов с данными); DDL (Data Definition Language) - описательный язык, который позволяет описывать сущности (объекты), необходимые для работы некоторого приложения, а также связи, имеющиеся между различными сущностями (теоретически, в архитектуре ANSI-SPARC, можно выделить языки DDL для каждой схемы, но на практике есть один, позволяющий задавать спецификации как минимум для внешней и концептуальной схем);

- контроллер словаря – управляет доступом к системному каталогу и обеспечивает работу с ним (почти все компоненты СУБД имеют доступ к системному каталогу).

## 2.4. Архитектура многопользовательских СУБД

Различают следующие архитектуры (по мере развития):

### 1) Телеобработка

В данном случае используется один компьютер, соединенный с терминалами пользователей (терминал – неинтеллектуальное устройство, предназначенное только для ввода-отображения информации), пользователь через терминал обращается к пользовательскому приложению, а то в свою очередь – к СУБД, затем результат идет в обратном порядке к пользователю. В последнее время наблюдается тенденция к децентрализации и замене дорогих мейнфреймов сетями персональных компьютеров (более эффективно).

### 2) Файловый сервер

Обработка данных распределена в сети (обычно – в локальной вычислительной сети (ЛВС)). Пользовательские приложения и сама СУБД размещены и функционируют на рабочих станциях и обращаются к файловому серверу только при необходимости доступа к данным (нужным файлам).

Недостатки:

- большой объем сетевого трафика;
- необходимость в наличии СУБД на каждой рабочей станции;
- управление параллельностью, восстановлением и целостностью данных усложняется (доступ к данным от нескольких экземпляров СУБД).

### 3) Технология «клиент/сервер»

Существует единая двухуровневая система, состоящая из клиентского процесса (толстый (интеллектуальный) клиент) (запрос некоторого ресурса) и серверного процесса (предоставление некоторого ресурса). Процессы совсем необязательно должны быть размещены на одном компьютере (обычно сервер располагается на одном узле ЛВС, а клиенты – на других узлах).

Клиент управляет пользовательским интерфейсом и логикой приложения (рабочая станция с пользовательским приложением): принимает от пользователя запрос, проверяет синтаксис и генерирует запрос к базе данных, передает сообщение серверу, ожидает поступление ответа и форматирует полученные данные для показа пользователю.

Сервер принимает и обрабатывает запросы к базе данных (включая проверку полномочий клиента, обеспечение требований целостности, поддержку системного каталога, поддержку параллельного доступа, выполнение запроса и обновление данных), а затем передает полученные результаты обработки обратно клиенту.

Преимущества технологии «клиент/сервер»:

- более широкий доступ к существующим базам данных;
- повышение производительности системы (по сравнению с выше перечисленными подходами);
- снижение стоимости аппаратного обеспечения (серверная машина – более мощная, чем клиентские);
- снижение сетевого трафика (по сети идут только необходимые данные);
- повышается уровень непротиворечивости данных (проверка целостности данных выполняется централизованно на сервере, а не у клиентов).

Расширение двухуровневой технологии «клиент/сервер» - переход к трехуровневой структуре (эффективна в Internet):

- тонкий (неинтеллектуальный) клиент – управление только пользовательским интерфейсом (например, web browser);
- средний уровень (клиент) – управление логикой пользовательского приложения;

- сервер базы данных.

## 2.5. Преимущества и недостатки СУБД

Преимущества СУБД:

- контроль за избыточностью данных (вводится контролируемая избыточность – хранятся только те данные, что необходимы для описания модели, теперь все данные хранятся централизованно);
- непротиворечивость данных (легче отслеживать возникновение противоречивых состояний в БД, чем в файлах разбросанных по отделам, кроме того, меньше избыточность данных (см. выше));
- больше полезной информации при том же объеме хранимых данных (т.к. в БД хранится вся информация, а не только данные одного отдела (как в файловых системах), то возможно использование совместных данных для анализа некоторых ситуаций и подготовке отчетов);
- совместное использование данных (фактически учет и ведение данных на новом уровне – не для себя одного (отдела), а в многопользовательском режиме – вся организация (большой объем данных для сотрудника));
- поддержка целостности данных (корректность и непротиворечивость хранимых данных – в одном месте легче настраивать, проверять и изменять);
- повышенная безопасность (несмотря на то, что централизованные данные более уязвимы, чем распределенные, в интегрированной системе проще выработать систему безопасности (прав доступа к данным) и заставить ее работать);
- применение стандартов (установка требуемых форматов данных для обмена, отчетов, обновления и правил доступа);
- повышение эффективности с ростом масштабов системы (снижение затрат на создание прикладных программ (т.к. приложения работают с



одним источником данных (доступ упрощен), перенос бюджета в область обновления оборудования без потерь в обновлении прикладных программ);

- возможность нахождения компромисса для противоречивых требований (потребности одних пользователей (отделов) могут противоречить потребностям другим пользователям (отделов), имея общий взгляд на всю структуру БД, администратор базы данных может контролировать такие противоречия и оперативно их устранять (обычно производительность отдается важным приложениям за счет менее важных));
- повышение доступности данных и их готовности к работе (т.е. устраняется время на распространение общих данных по отделам, данные становятся доступными сразу по их внесению в БД);
- улучшение показателей производительности (в СУБД программист освобожден от реализации стандартных действий (т.к. такие функции уже созданы и оптимизированы), поэтому его время используется более продуктивно);
- упрощение сопровождения системы за счет независимости от данных (т.к. в СУБД описания данных отделены от приложений, поэтому приложения защищены от изменений в описании данных, что упрощает обслуживание и сопровождение приложений);
- улучшенное управление параллельностью (в файловых системах при одновременной записи в файл возможна потеря данных или их целостности, СУБД позволяет вести параллельный доступ к БД);
- развитые службы резервного копирования и восстановления (в файловых системах резервное копирование должен делать пользователь, в СУБД – данные лежат централизованно, и данные операции поддерживаются автоматически (программно и аппаратно)).

Недостатки СУБД:

- сложность и размер (при проектировании БД, разработчики должны хорошо представлять функциональные возможности СУБД, непонимание

может приводить к неудачным результатам проектирования, более сложные СУБД занимают значительный объем дискового пространства (при равных качествах кода));

- стоимость СУБД и дополнительные затраты на аппаратное обеспечение (чем больше возможностей у СУБД (и число пользователей БД при равной производительности), тем выше ее стоимость, стоимость ее сопровождения и аппаратных затрат);
- затраты на преобразование (при переносе БД с одной СУБД на другую важную роль представляют не только затраты на аппаратные и программные средства, но и затраты на перенос данных, переучивание специалистов, запуск новой системы (т.е. правильный выбор СУБД – тонкая вещь));
- производительность (т.к. приложения файловых систем оптимизируются каждое к конкретному случаю, то их производительность может быть очень высока, СУБД же предназначена для решения общих задач и соответственно «притормаживать»);
- серьезные последствия при выходе системы из строя (централизация повышает уязвимость системы, выход из строя компонента СУБД – прекращение работы организации).

### 3. ПРОЕКТИРОВАНИЕ БД

Проектирование БД – процесс, который для заданного набора данных, относящихся к некоторой предметной области, позволяет выбрать и построить соответствующую оптимальную структуру БД.

Все вопросы и термины, связанные с ниже описанными задачами и общей методологией проектирования БД, будут раскрыты подробнее в последующих главах.

#### 3.1. Задачи проектирования БД

При проектировании базы данных решаются три основные задачи:

1. «Как адекватно отразить предметную область и информационные потребности пользователей в семантическую модель БД?». Эту проблему называют проблемой инфологического (концептуального) проектирования баз данных.

2. «Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области, и было по возможности эффективным?» Эту проблему называют проблемой логического проектирования баз данных.

3. «Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создать дополнительные структуры данных (например, индексы)?» Эту проблему называют проблемой физического проектирования баз данных.

В процессе проектирования, первичным этапом является решение первых двух задач – создания логического макета (логической модели) БД, и только потом этот логический макет отображают на некоторые физические структуры,

поддерживаемые конкретной СУБД. Таким образом, физический макет (физическая модель) является специфическим для каждой СУБД, а логический макет наоборот совершенно независим от СУБД и для его реализации могут быть использованы строгие теоретические принципы. Данный вид проектирования и его методики называются нисходящими, т.к. они позволяют выполнять преобразование от общего к частному.

На практике реализация физического макета может оказывать существенное влияние на логический макет, поэтому конечная структура БД будет некоторым компромиссом в связке «логический макет – физический макет».

Следует учесть, что проектирование БД больше искусство, чем просто наука. Научные принципы составления БД будут изложены далее, но при проектировании возникает множество проблем, которые нельзя охватить этими принципами. В области проектирования БД разработано большое количество методик проектирования, но все они являются более или менее специализированными (т.е. позволяют построить такой логический макет, который был бы лучшим в некоторой конкретной ситуации). Кроме того, в большинстве случаев построение макета БД рассматривается независимо от приложений, которые будут работать с этими данными. Это происходит по причине невозможности учета на начальной стадии проектирования всех возможных способов использования данных, и соответственно полученный макет не будет отвечать требованиям повышенной производительности, т.к. не будет зависить от аппаратной платформы, операционной системы, СУБД, языка программирования и т.п.

### 3.2. Общая методология проектирования БД

Под методологией проектирования понимают структурированный подход, предусматривающий использование специализированных процедур,

технических приемов, инструментов, документации и нацеленный на поддержку и упрощение процесса проектирования. При этом процесс проектирования разбивается на фазы и этапы, которые предлагают к выполнению набор технических приемов и, следуя которым, разработчик может оптимально решать задачи проектирования, стоящие перед ним на данной стадии разработки. Методология проектирования фактически разбивает всю огромную проблему на ряд небольших формализованных подзадач, которые по отдельности решаются проще основной проблемы, однако, следуя путем, заданным конкретной методологией, разработчик может получить в ряде случаев неоптимальный или неадекватный макет (т.е. методология обычно является специализированной и подходит только для частных случаев). Ниже приведена общая методология проектирования БД, которая наименее специализирована.

Общая методология проектирования состоит из трех фаз: концептуальной, логической и физической.

Концептуальное проектирование БД – это процедура конструирования информационной модели, не зависящей от каких-либо физических условий реализации. Фаза концептуального проектирования БД начинается с создания локальных концептуальных моделей данных на основе представлений о предметной области каждого отдельного типа пользователей. Формально этот этап можно разделить на следующие подзадачи:

- интервьюирование (опрос) будущих пользователей БД (уточнение задач решаемых пользователем и данных необходимых для решения этих задач, уточнение связей между пользователями (например, иерархия подчиненности) и т.п.);
- идентификация объектов с учетом особенностей интервью (анализ названий объектов на синонимы и омонимы, с учетом разных вариантов описания и отнесения объекта к предметной области);
- идентификация связей, исключая те, что не входят в рамки проекта;
- идентификация атрибутов, кроме названий должны учитываться

особенности использования атрибутов (в частности, производные, необязательные, составные и т.п. атрибуты);

- создание графического представления локальных концептуальных моделей;
- проверка и обсуждение локальных концептуальных моделей с конечными пользователями (для понимания отражения моделью реального мира пользователя).

Особенность этапа концептуального проектирования для данной методологии является то, что число получаемых моделей на данном этапе зависит от числа опрошенных групп пользователей и на данном этапе модели для этих групп представлены обособленно друг от друга – это позволяет декомпозировать задачу опроса и упростить вхождение в проектирование БД.

Логическое проектирование БД – процесс конструирования информационной модели на основе выбора существующих логических моделей данных, но не зависимой от конкретной СУБД и прочих физических условий реализации. Фаза логического проектирования состоит из двух этапов. На первом этапе для каждой концептуальной модели данных, полученной на первой фазе проектирования, строится адекватная логическая модель, при этом выполняются следующие подзадачи:

- преобразование локальной концептуальной модели данных в локальную логическую модель (что предусматривает преобразование сложных связей и атрибутов (например, связей типа M:N, рекурсивных и связей с атрибутами), перепроверка связей типа 1:1, удаление избыточных связей);
- определение набора отношений исходя из структуры локальной логической модели;
- проверка модели с помощью правил нормализации (актуально для реляционной модели данных, проверяет группировку атрибутов для описания объектов и связей, устранение неоднозначностей и избыточности в описании данных, повышение гибкости модели для

возможного расширения);

- проверка модели в отношении транзакций пользователей (убедиться что модель позволяет выполнять все действия пользователя с данными);
- определение требований поддержки целостности данных;
- создание графического представления локальных логических моделей;
- проверка и обсуждение локальных логических моделей с конечными пользователями.

На втором этапе логического проектирования проводится создание и проверка глобальной логической модели данных на основе локальных логических моделей данных, при этом выполняются следующие подзадачи:

- слияние локальных логических моделей данных в единую глобальную логическую модель данных (устранение противоречий объединения – анализ имен объектов, связей и атрибутов на пропуски и дубликаты, проверка корректности ссылок, соблюдения ограничений целостности, приведения в порядок бизнес-правил);
- проверка глобальной логической модели данных;
- проверка возможностей расширения модели в будущем (определение частей модели, которые будут расширяться в будущем, согласование расширений с пользователями);
- создание графического представления глобальной логической модели;
- проверка и обсуждение глобальной логической модели с конечными пользователями.

Концептуальное и логическое проектирование в общей методологии проектирования – это итеративные процессы. Они могут циклически продолжаться (т.к. понимание данных пользователей у разработчиков будет расти в ходе проектирования и потребуются улучшения уже созданного, для выполнения этого очень важна роль опросов и обсуждений хода разработки с конечными пользователями) до тех пор, пока не закончится внесение уточнений и улучшений в составляемую схему БД. Плохо выполненное концептуальное и логическое проектирование – ведет к небольшой

производительности физической реализации, к плохой адаптации к развитию и изменениям и даже к провалу физической реализации БД. Однако не стоит при проектировании впадать в другую крайность – число циклов пересмотра концептуальной и логической моделей ограничены временем разработки и не должны выполняться за счет сокращения времени на физическое проектирование БД.

Физическое проектирование БД – процесс создания конкретной реализации БД, размещаемой во вторичной памяти (например, накопители типа винчестер) вычислительной машины. В процессе физического проектирования выполняется отображение созданной глобальной логической модели на особенности конкретной СУБД. Для наиболее распространенных сейчас реляционных СУБД этот процесс можно разбить на следующие этапы и подзадачи:

Этап 1. Перенос глобальной логической модели в среду целевой СУБД:

- проектирование базовых таблиц (с учетом наиболее полного соответствия выбранной логической модели (например, реализация ключей), добавление необходимых структур обслуживания – триггеры, первичные индексы);
- реализация бизнес-правил (зависит от СУБД, лучший вариант – полное использование возможностей СУБД, не переносить бизнес-правила в приложения).

Этап 2. Проектирование физического представления БД:

- анализ транзакций (выполнение анализа на пропускную способность (число транзакций, выполненных за определенный интервал времени), анализ времени ответа на запрос, отнесение транзакции к важным);
- выбор файловой структуры (для оптимальной файловой организации);
- определение вторичных индексов (для ускорения выполнения транзакций по не ключевым атрибутам и ссылкам);
- анализ необходимости введения контролируемой избыточности данных (процесс обратный нормализации, применяется для повышения



производительности системы, только если исчерпаны другие возможности, может привести к снижению гибкости и расширяемости БД, а также усложняет реализацию и обновление данных);

- определение требований к дисковой памяти (в том числе учет требований для обоснования приобретения нового оборудования).

Этап 3. Разработка механизмов защиты:

- разработка пользовательских представлений (видов);
- определение прав доступа.

Этап 4. Организация мониторинга и настройка функционирования системы (требуется для устранения ошибочных проектных решений и изменения требований к системе (например, отказ от более дорогого оборудования, улучшение психологического комфорта пользователей по работе с системой); настройка БД производится фактически постоянно, а не только при первом запуске системы, что позволяет оперативно реагировать на изменения в состоянии системы и требования пользователей; внесение любых изменений должно производиться обдуманно и осторожно, с учетом глобального влияния изменений, для оценки влияния изменений применяют дубликат системы).

Общая методология проектирования позволяет повысить общую эффективность работы по проектированию БД. В основу данной методологии положен циклический процесс проектирования, а также существенная роль отводится конечным пользователям, привлекаемым для ознакомления и проверки моделей данных и проектной документации. Выполнение той или иной подзадачи в методологии не регламентировано жестко и порядок их выполнения в рамках этапов может меняться по усмотрению разработчиков.

Более подробно модели данных и технические приемы, заложенные в методологию, будут рассмотрены далее.

## 4. МОДЕЛИ ДАННЫХ

Для определения схемы БД таким образом, чтобы она могла быть понятной пользователям разных категорий, описание на языке DDL конкретной СУБД не подходит (оно слишком низкого уровня) и требуется описывать схему БД на более высоком уровне – уровне модели данных.

Модель данных – интегрированный набор понятий для описания данных, связей между ними и ограничений, накладываемых на данные.

### 4.1. Определение и классификация моделей данных

Модель данных является представлением «реального мира» (т.е. реальных объектов и событий, и связей между ними), это некоторая абстракция, в которой остаются только те части реального мира, которые важны для разработчиков БД, а все второстепенные – игнорируются.

Цель построения модели данных – представление данных в понятном виде, который легко можно применить при проектировании БД (модель должна точно и недвусмысленно описывать части реального мира в таком виде, который позволяет разработчикам и пользователям (заказчикам) БД обмениваться мнениями при разработке и поддержке БД).

Модель данных включает в себя следующие части:

- структурная часть – набор правил, по которым может быть построена БД;
- управляющая часть – определяет типы допустимых операций с данными (обновление и извлечение данных, изменение структуры);
- ограничения поддержки целостности данных (необязательная часть).

Модели данных можно разделить на три категории:

- 1) концептуальные (объектные) модели данных – описание данных высокого уровня (уровня объектов): модель типа «сущность-связь» или ER-модель (Entity-Relationship), объектно-ориентированная модель

- (состояние + поведение объектов), функциональная модель;
- 2) логические модели данных (модели данных на основе записей) – БД состоит из логических записей фиксированного формата: сетевая модель данных (network), иерархическая модель данных (hierarchical) и реляционная модель данных (relational) (хотя и не в такой степени, как две предыдущие – табличный подход (как в представлении данных для пользователя, так и в обработке (декларативный подход (какие данные надо извлечь), а не навигационный (по одной записи))) и неявные связи между записями – ближе к объектным моделям);
  - 3) физические модели данных – описывают, как данные хранятся в компьютере (информация о структуре записей, порядке расположения записей и путях доступа к ним): обобщающая модель (unifying), модель памяти кадров (frame memory).

Для отображения ANSI-SPARC архитектуры существует три модели данных:

- внешняя модель данных (предметная область пользователей),
- концептуальная модель данных (логическое представление о данных),
- внутренняя модель данных (представление концептуальной модели средствами конкретной СУБД).

Первые две категории моделей данных используются для описания концептуального и внешнего уровней, а последняя – для внутреннего уровня.

## 4.2. Концептуальные модели данных

### 4.2.1. Семантическое моделирование данных

Семантическое моделирование данных – область исследований о способах представления смыслового значения данных. Это направление получило интенсивное развитие в конце 1970-х, мотив – для расширения

понимания смыслового значения данных (хотелось бы, чтобы СУБД была более «разумной» и могла понимать что «вес товара» и «количество поставок» являются хотя и числовыми, но все же семантически разными величинами).

Общий подход к проблеме семантического моделирования состоит из четырех этапов:

- 1) Задание множества семантических концепций (понятий), которые полезны при обсуждении реального мира: мир состоит из объектов (определяются интуитивно); для объектов есть разделение на типы объектов (обобщенное определение группы однотипных объектов) и экземпляры объектов (конкретный объект из группы); все объекты определенного типа обладают общими свойствами, и существует такое свойство (свойства), которое позволяет различать объекты в группе (т.е. каждый объект обладает идентичностью); группы объектов могут связываться с другими группами объектов с помощью отношений.
- 2) Определение множества соответствующих символических (т.е. формальных) объектов, которые могут быть использованы для представления определенных выше семантических концепций (однако разные пользователи могут рассматривать один и тот же предмет реального мира по-разному (как объект, как свойство или как отношение) – гибкость интерпретации данных в семантическом моделировании).
- 3) Вывод множества формальных правил целостности для работы с формальными объектами.
- 4) Задание множества формальных операторов для манипулирования формальными объектами (однако, с точки зрения проектировщика БД, операторы являются менее важной частью модели по сравнению с объектами и правилами целостности).

Идеи семантического моделирования могут быть полезны при проектировании БД даже при отсутствии непосредственной поддержки этих идей в СУБД – при проектировании макета БД. Методологии проектирования, основанные на идеях семантического моделирования, называются

нисходящими методологиями, т.к. они начинаются на высшем уровне абстракции с конструкций «реального мира» (объектов), а заканчиваются на сравнительно низком уровне абстракции, представленном конкретной структурой БД.

#### 4.2.2. ER-модель (модель типа «сущность-связь» или «объект/отношение»)

ER-модель была предложена Ченом (Chen P.P.) в 1976 г и представляет собой высокоуровневую концептуальную модель данных (основана на идеях семантического моделирования, содержит набор концепций, которые описывают структуру данных и связанные с ней транзакции обновления и извлечения данных, не зависящие от конкретной СУБД или аппаратной платформы).

Основа популярности модели – наличие диаграммной техники (ER-диаграмм) – графического представления логической структуры БД (используются в качестве основы для методики проектирования БД, причем диаграмма может не включать в себя все идеи ER-модели).

##### 4.2.2.1. Концепции ER-модели

В ER-модели выделены следующие три концепции:

###### 1) Объекты (типы сущностей):

Типы объектов (типы сущностей) – множество объектов реального мира с одинаковыми свойствами, характеризуются независимым существованием и могут быть объектом как с реальным (физическим) существованием (например, «работник», «деталь», «поставщик»), так и объектом с абстрактным (концептуальным) существованием (например, «рабочий стаж», «осмотр

объекта»). Разные разработчики могут выделять разные типы объектов. Каждый тип объекта идентифицируется именем и списком свойств.

Объект (экземпляр типа объекта или сущность) – экземпляр типа сущности, «предмет, который может быть четко идентифицирован» на основе свойств (т.к. обладает уникальным набором свойств среди объектов одного типа).

Типы объектов классифицируются как сильные и слабые:

- слабый тип объекта (дочерний, зависимый или подчиненный) – тип объекта, существование которого зависит от какого-то другого типа объекта;
- сильный тип объекта (родительский, владелец или доминантный) – тип объекта, существование которого не зависит от какого-то другого типа объекта.

Представление объектов на диаграмме: сильный тип объекта – прямоугольник, с именем внутри него; слабый тип объекта – прямоугольник с двойным контуром (деление объектов на сильные и слабые – по желанию проектировщика).

## 2) Свойства (атрибуты):

Свойства (атрибуты) – служат для описания типов объектов или отношений. Значения свойств каждого типа извлекаются из соответствующего множества значений (в этом множестве определяются все потенциальные значения свойства, различные свойства могут использовать одно множество значений).

Свойства делят по характеристикам:

- простые и составные: простое свойство состоит из одного компонента с независимым существованием (такое свойство – атомарное (не может быть разделено на более мелкие компоненты); например «зарплата», «пол»); составное свойство – состоит из нескольких компонентов, каждый из которых характеризуется независимым существованием

(могут быть разделены на более мелкие части (например, «адрес»), но решение о представлении такого атрибута (как простой или составной) зависит от проектировщика);

- однозначные и многозначные: однозначное свойство – свойство, которое может содержать только одно значение для одного объекта; многозначное свойство – может содержать несколько значений для одного объекта (например, «телефон компании»);
- производные и базовые: производное свойство - представляет значение, производное от значения связанного с ним свойства или некоторого множества свойств, принадлежащих некоторому типу объектов (не обязательно одному), например, «сумма деталей», «возраст сотрудника»; базовое – не зависит от других свойств.
- Ключевое и не ключевое: ключ – свойство (набор свойств), которое однозначно определяет объект из всех объектов данного типа (например, «номер паспорта»); ключи делят на: потенциальные – содержит значения, которые позволяют уникально идентифицировать каждый отдельный экземпляр типа объекта (тип объекта может иметь несколько потенциальных ключей); первичные – один из потенциальных ключей для типа объекта, который выбран для идентификации объектов (должен гарантировать уникальность значений не только в текущий период, но и в обозримом будущем, лучше чтобы он содержал меньшее число свойств (например, «номер сотрудника», «номер паспорта»)); альтернативные – все потенциальные ключи, кроме первичного; составной ключ – потенциальный ключ, который состоит из 2-х и более свойств (позволяет определить уникальность объекта, не вводя лишних уникальных свойств).

Представление свойств на диаграмме: в виде эллипсов с именем внутри него, присоединенных линией к типу объекта; для производных свойств – эллипс окружен пунктирным контуром, для многозначных – двойным; имя свойства, которое является первичным ключом – подчеркивается.

### 3) Отношения (типы связей):

Типы отношений (типы связи) – осмысленная ассоциация (связь) между типами объектов, каждое отношение имеет имя, описывающее его функцию.

Экземпляр отношения (отношение) – ассоциация (связь) между экземплярами объектов, включающая по одному экземпляру объекта с каждой стороны связи.

Объекты, включенные в отношение, называются участниками этого отношения (при этом в связях для определения функций каждого участника могут присваиваться ролевые имена). Количество участников данного отношения называется степенью этого отношения (два участника – бинарная (наиболее часто применима), три – тернарная, четыре – кватернарная, n-участников – n-арная). Существуют унарные (рекурсивные) отношения - в них одни и те же типы объектов участвуют несколько раз и в разных ролях.

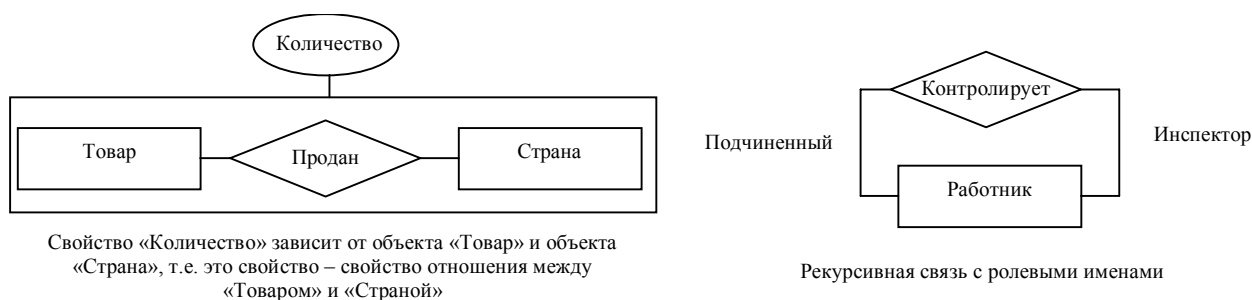


Рисунок 4. Свойство отношения (составного объекта). Унарное отношение.

Отношения можно рассматривать как своего рода объекты (объектные множества или составные объекты), т.е. разделение объектов и отношений достаточно зыбко и зависит от представлений проектировщика (при этом для отношений возможно наличие свойств – это обычно означает, что отношение скрывает некоторую неопределенную сущность).

Представление отношений на диаграмме: в виде ромба (при связи слабого типа объекта с сильным, ромб отношения имеет двойной контур) с указанным в нем именем связи и соединенного линиями с участниками отношения.



#### 4.2.2.2. Структурные ограничения ER-модели

Структурные ограничения, накладываемые на участников отношения, являются отражением требований реального мира (частично это правила целостности).

Показатель кардинальности (мощность отношения) – максимальное количество элементов одного типа объекта, связанных с одним элементом другого типа объекта. Обычно рассматриваются следующие виды связей:

- «один к одному» – максимальная мощность отношения в обоих направлениях равна одному;
- «один ко многим» - максимальная мощность отношения в одном направлении равна одному, а в другом – многим;
- «многие ко многим» - максимальная мощность отношения в обоих направлениях равна многим.

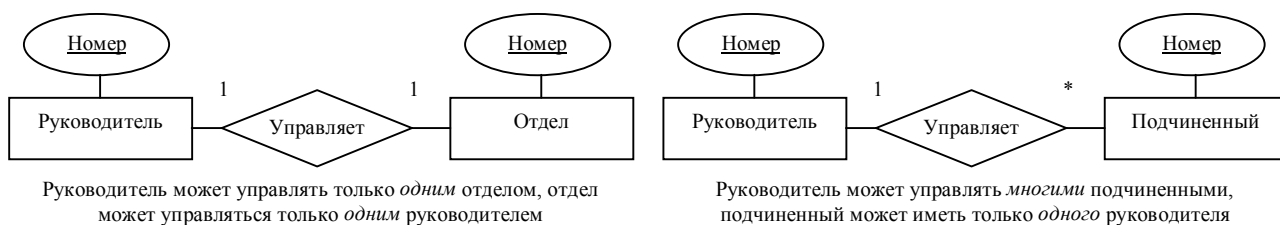


Рисунок 5. Отношение «один к одному». Отношение «один ко многим».



Рисунок 6. Отношение «многие ко многим».

Показатель кардинальности для каждого отношения (обычно рассматриваются бинарные отношения) определяется обычно бизнес-

правилами (производственными правилами) организации (не все бизнес-правила могут быть представлены с помощью ER-модели). На диаграммах возможно указание не только максимальной мощности отношения, но и минимального числа допустимых связей для каждого участника отношения (более информативно).

По степени участия объектов в отношении выделяют:

- полное (обязательное) участие объекта в связи – для существования некоторого объекта требуется существование другого объекта, связанного с первым связью (на диаграмме соединение с отношения с таким объектом выполняется двойной линией);
- частичное (необязательное) участие объекта в связи – для существования некоторого объекта не требуется существования другого объекта, связанного с первым связью.



Рисунок 7. Степень участия объектов в отношении.

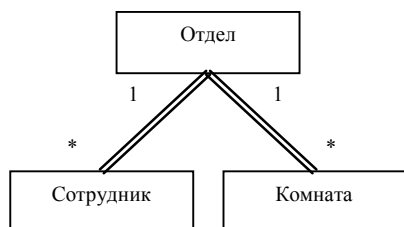
#### 4.2.2.3. Проблемы ER-моделирования

Проблемы, которые обычно возникают при разработке модели данных, принято называть ловушками соединения (возникают при неправильной интерпретации смысла некоторых связей).

Проверка на наличие таких ловушек важна, т.к. их устранение может потребовать перестройки всей модели. Два основных типа ловушек

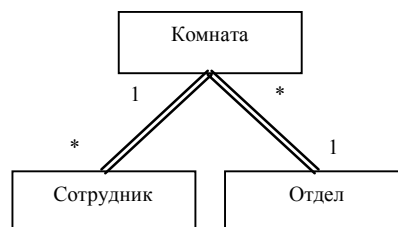
соединения: ловушка разветвления и ловушка разрыва.

Ловушка разветвления – имеет место в том случае, когда два или больше отношений типа «один ко многим» разветвляются из одного объекта.



Один отдел может быть расположен в нескольких комнатах, в комнате может быть расположен только один отдел, в отделе может работать много сотрудников, сотрудник может работать только в одном отделе.

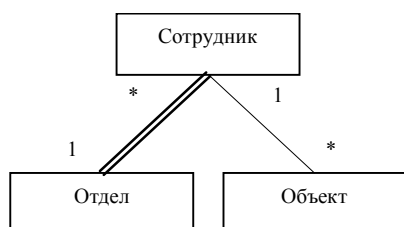
Ловушка: «В какой комнате рабочее место каждого сотрудника?»



Возможный выход из ловушки: в одной комнате расположен один отдел, сотрудник может работать только в одной комнате (новые отношения не вводятся)

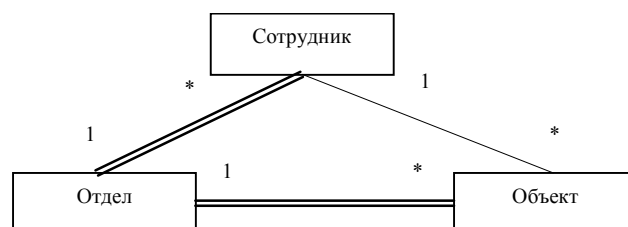
Рисунок 8. Ловушка разветвления и ее решение (все отношения с полным участием).

Ловушка разрыва – может возникнуть при наличии отношения с частичным участием, образующего часть пути между связанными объектами.



Отдел содержит множество сотрудников, сотрудник может работать только в одном отделе, сотрудник может контролировать от 0 до n объектов, объект контролируется не более чем одним сотрудником (т.к. отношение «сотрудник-объект» - с *частичным* участием).

Ловушка: «Какой объект принадлежит отделу?»



Возможный выход из ловушки: ввод дополнительного отношения «отдел-объект» (с *полным* участием)

Рисунок 9. Ловушка разрыва и ее решение.

#### 4.2.3. EER-модель (Расширенная ER-модель)

С помощью ER-модели можно представлять большинство схем БД в административно-управленческих приложениях. Однако для создания БД в

приложениях автоматизированного проектирования (CAD), автоматизированного создания программ (CASE) и мультимедийных приложениях необходимо введение новых понятий (понятий ER-модели недостаточно). В результате дополнения ER-модели новыми семантическими концепциями была создана EER-модель (Enhanced ER – расширенная ER-модель).

EER-модель включает все концепции ER-модели плюс дополнительные концепции специализации, генерализации и категоризации.

Дополнительные концепции связаны с новыми понятиями типов объектов (суперклассами и подклассами), а также с процессом наследования атрибутов.

Суперкласс – тип объекта, включающий различные подклассы.

Подкласс – член суперкласса, который представляет собой некоторый тип объекта имеющий общие черты с суперклассом, но играющий отдельную роль (например, суперкласс – сотрудник, подклассы – менеджер, секретарь, бухгалтер и т.п.). Связь между суперклассом и любым его подклассом называется отношением «суперкласс/подкласс» (тип «один к одному»). Некоторые суперклассы могут содержать перекрывающиеся подклассы (например, сотрудник может быть менеджером и инспектором одновременно), а также не все члены суперкласса обязаны входить в какой либо его подкласс.

Причины введения понятий суперклассов и подклассов:

- исключение описания внутри одного типа объекта его различных видов с возможно разными атрибутами – уменьшение числа неопределенных атрибутов для типа объекта (а также возможных отношений характерных только для определенного вида объекта);
- позволяет избежать повторного определения сходных понятий (за счет наследования атрибутов - экономия времени и лучшая читаемость диаграмм);
- больше семантической информации включено в модель (причем в более привычной для человека форме (например, «менеджер является сотрудником»)).

Объект подкласса обладает как специфическими атрибутами (характерными только для него), так и атрибутами, связанными с суперклассом (наследование атрибутов). Так как подкласс также является типом объекта, то он в свою очередь может иметь собственные подклассы (иерархии типа).

Специализация – процесс увеличения различий между отдельными типами объектов за счет выделения их отличительных характеристик (нисходящий подход к определению множеств суперклассов и связанных с ними подклассов).

Генерализация – процесс сведения различий между объектами к минимуму путем выделения их общих характеристик (восходящий подход, позволяет создать обобщенный суперкласс на основе различных исходных подклассов (противоположность подходу специализации) – поиск сходств (общих атрибутов и связей)).

На специализацию могут накладываться следующие ограничения (ограничения пересечения и участия для специализации и генерализации отличаются):

- ограничение непересечения - если подклассы некоторой специализации не пересекаются, то каждый отдельный объект может быть членом только одного из подклассов данной специализации (для представления непересекающейся специализации используется символ «d», расположенный в центре кружка, соединяющего подклассы с суперклассом; для пересекающейся – «o»);
- ограничение участия – полное (означает, что каждый объект суперкласса должен быть членом подкласса этой специализации (между суперклассом и кружком специализации проводят двойную линию)) или частичное (означает, что объект не обязательно должен быть членом любого подкласса этой специализации (ординарная линия)).

Категоризация – процесс моделирования единственного подкласса (категории) со связью, которая охватывает несколько суперклассов.

Операцию категоризации можно детализировать с учетом полного

(каждый экземпляр всех суперклассов должен быть представлен в данной категории (обозначается двойной линией, соединяющей категорию с кружком категоризации)) или частичного (не все экземпляры суперклассов могут присутствовать в данной категории) участия сторон.

### 4.3. Логические модели данных

#### 4.3.1. Иерархическая модель данных

Иерархическая модель данных возникла из практики работы с данными и создавалась по сути дела программистами. Стандарт на иерархическую модель не был разработан. Наиболее распространенной СУБД реализующей иерархическую модель являлась СУБД IMS, в рамках которой будет рассмотрена иерархическая модель данных.

СУБД IMS – Information Management System была создана в начале 1960-х годов (совместная разработка IBM и North American Aviation) для поддержки проекта «Аполлон» (Apollo - американская лунная программа). Ключевым фактором при создании IMS была необходимость управления большим количеством деталей, связанных друг с другом иерархическим образом (детали -> узлы -> модули и т.п.).

Иерархическая модель – модель данных на основе записей, в которой все отношения между объектами структурированы в виде деревьев. Все записи хранятся при этом в общей древовидной структуре, имеющей одну корневую запись. Каждый узел дерева – запись, описывающая экземпляр объекта. Соединения между записями в дереве определяют связи между экземплярами объектов. В иерархической модели связи называются отношениями «предок-потомок», а записи – сегментами. В иерархической модели потомок может быть связан только с одним предком (для реализации связи с разными предками

требуется построение разных деревьев). Отношение «предок-потомок» реализуют связи один-ко-многим. Иерархическая БД – совокупность деревьев. Все отношения, которые могут быть представлены в иерархической модели данных – бинарные.

Алгоритм перевода ER-диаграммы в иерархическую модель данных представлен следующими шагами:

1) Для каждого объекта ER-диаграммы создается тип сегмента в иерархической модели. Название типа сегмента соответствует названию объекта. Атрибуты объекта задают поля записи сегмента (название, тип данных).

2) В диаграммах деревьев отношения один-ко-многим задаются с помощью отношений «предок-потомок», причем объект со стороны много в связи становится потомком, а объект со стороны один – предком.

3) Для реализации отношений многие-ко-многим создается два разных отношения «предок-потомок», в этих отношениях один и тот же объект выступает в качестве предка (первое отношение «предок-потомок») и в качестве потомка (второе отношение «предок-потомок»). В одном из таких отношений «предок-потомок» реальные данные для экономии места заменяют на ссылки (указатели) на данные из второго отношения.

4) Если отношение мощности многие-ко-многим на ER-диаграмме обладает атрибутом, то в диаграммах деревьев создается сегмент пересечения, который располагается между предком и потомком и содержит значение атрибута.

В IMS дерево может состоять из 15 уровней, количество сегментов – 255. Концептуальный уровень ANSI/SPARC архитектуры представлен описанием базы данных, в котором определяется формат, длина и адрес каждого сегмента данных и положение каждого сегмента в структуре дерева:

; определение базы ОТДЕЛ

DBD NAME=BD\_OTDEL

; определение сегмента ОТДЕЛ (корневой сегмент)

SEGM NAME=OTDEL,PARENT=0,BYTES=20

; задание поля сегмента (дополнительные опции: SEQ – сортировка по значению поля, U – уникальность значения)

FIELD NAME=(NAME,SEQ,U),BYTES=10,START=1,TYPE=C

FIELD NAME=DIRECTOR,BYTES=10,START=11,TYPE=C

; определение сегмента СОТРУДНИК (потомок сегмента ОТДЕЛ)

SEGM NAME=SOTRUDNIK,PARENT=OTDEL,BYTES=22

FIELD NAME=(NAME,SEQ),BYTES=20,START=1,TYPE=C

FIELD NAME=NUMBER,BYTES=2,START=21,TYPE=P

; определение сегмента СПЕЦИАЛЬНОСТЬ (потомок сегмента СОТРУДНИК)

SEGM NAME=SPEC,PARENT= SOTRUDNIK,BYTES=17

FIELD NAME=(CODE,SEQ),BYTES=2,START=1,TYPE=P

FIELD NAME=NAME,BYTES=15,START=3,TYPE=C

Внешний уровень архитектуры ANSI/SPARC в IMS представлен блоком программной спецификации – определяет представление данных, которым будет пользоваться прикладная программа, делится на блоки программной коммуникации, в которых определяются сегменты к которым возможно обращение:

; определение блока программной коммуникации (PROCORT определяет опции доступа (Get, Insert, Replace, Delete, All))

PCB TYPE=DB,DBNAME=DB\_OTDEL,PROCORT=A,KEYLEN=18

SENSEG NAME=OTDEL,PARENT=0

SENFLD NAME=NAME,START=1

SENSEG NAME= SOTRUDNIK,PARENT=OTDEL

SENFLD NAME=NAME,START=1

Для обработки данных в IMS был создан DML DL/I. Программная рабочая область СУБД содержала следующие переменные: шаблоны сегментов, указатели текущего положения и флаги состояния. Доступные команды позволяли выполнить поиск и извлечение данных в соответствующий шаблон



сегмента, блокировать, заменить, удалить и вставить данные. Обработка данных осуществляется навигационным подходом.

Основное достоинство иерархической модели данных - представление данных связанных иерархически.

К основным недостаткам иерархической модели данных относятся:

- нет разделения логической и физической модели данных;
- непредвиденные запросы требуют реорганизации БД;
- для не иерархических связей описание сильно усложняется и требует много памяти;
- сложность составления программ обработки данных;
- нет стандарта на построение подобных систем.

#### 4.3.2. Сетевая модель данных

Сетевая модель данных – модель данных на основе записей, в которой данные представлены сетевыми структурами типа направленного графа. Узлы в такой модели представляют объекты и называются – типы записей данных. Ребра графа представляют отношения (связи) между объектами и называются наборами. Наборы выражают отношения один-ко-многим или один-к-одному между двумя типами записей, при этом выделяют тип записи владельца – со стороны один связи, и тип записи члена набора – со стороны много связи. Сетевая БД состоит из наборов отношений. Все отношения, которые могут быть представлены в сетевой модели данных – бинарные.

Сетевая модель данных была разработана в рамках CODASYL (Conference on Data Systems Languages) группой DBTG (Database Task Group) в конце 1960-х годов. Наиболее известные реализации: СУБД IDS (Integrated Data Store) и СУБД IDMS.

Алгоритм перевода ER-диаграммы в сетевую модель данных представлен следующими шагами:

1) Для каждого объекта ER-диаграммы создается тип записей в сетевой модели. Название типа записей соответствует названию объекта. Атрибуты объекта задают поля типа записи (название, тип данных).

2) Для отношения один-ко-многим тип записи со стороны один отношения становится владельцем набора, а тип записи со стороны много отношения – членом набора. Если мощность отношения один-к-одному, то типы записей в наборе выбираются произвольно.

3) Для каждого n-арного отношения, где  $n > 2$ , создается связывающая запись, которая становится типом записи члена для n наборов. Владельцами наборов назначаются типы записей со стороны один, полученных в результате преобразований отношений один-ко-многим.

4) Для реализации отношений многие-ко-многим создается связывающая запись, которая становится типом записи члена для двух наборов, владельцами которых являются типы записей, соответствующие объектам связи

5) Если отношение мощности многие-ко-многим на ER-диаграмме обладает атрибутом, то этот атрибут добавляется в связывающую запись.

В сетевой модели DBTG концептуальный уровень ANSI/SPARC архитектуры представлен схемой данных, в которой задается имя схемы, типы записей и наборы схемы:

```
; имя схемы ЗАПИСИСЧЕТОВ
SCHEMA NAME IS RECORDSCHET
; тип записи КЛИЕНТ
RECORD NAME IS KLIENT
NUMBER TYPE IS NUMERIC INTEGER
NAME TYPE IS CHARACTER 15
ADRESS TYPE IS CHARACTER 40
BALANS TYPE IS NUMERIC(5,2)
; тип записи СЧЕТ
RECORD NAME IS SCHET
NUMBER TYPE IS NUMERIC INTEGER
```

DATA TYPE IS CHARACTER 9  
 STATUS TYPE IS CHARACTER 9  
 DATA TYPE IS NUMERIC(5,2)  
 ; тип записи ЭЛЕМЕНТ  
 RECORD NAME IS ELEMENT  
 NUMBER TYPE IS NUMERIC INTEGER  
 DESCR TYPE IS CHARACTER 20  
 PRICE TYPE IS NUMERIC(4,2)  
 ; набор КЛИЕНТ-СЧЕТ  
 KLIENT\_SCHET  
 OWNER IS KLIENT  
 MEMBER IS SCHET  
 ; набор СЧЕТ-ЭЛЕМЕНТ  
 SCHET\_ELEMENT  
 OWNER IS SCHET  
 MEMBER IS ELEMENT

Внешний уровень архитектуры ANSI/SPARC в IMS представлен подсхемой – определяет представление данных, которым будет пользоваться программа, делится на отдел заголовка, отдел преобразований (для преобразования имен схемы в необходимые) и структурного отдела (описывает раздел записей и наборов доступных в подсхеме):

; подсхема СТАТУССЧЕТА  
 SS STATUSSCHET WITHIN RECORDSCHET  
 ; отдел преобразования  
 MAPPING DIVISION  
 ALIAS SECTION  
 AD RECORD KLIENT IS PAYMAN  
 AD SET KLIENT\_SCHET IS PAYMAN\_SCHET  
 ; структурный отдел  
 RECORD SECTION

0I PAYMAN

0S NUMBER

0S NAME

0S BALANS

0I SCHEM ALL

SET SECTION

SD PAYMAN\_SCHEM

Для обработки данных в сетевой модели DBTG был создан DML.

Термины DML DBTG:

- пользовательская рабочая область – память, в которой хранятся переменные программы – указатели на записи и наборы разных типов;
- индикаторы текущего состояния – метки записей;
- флаги состояния – переменные применяемые для обозначения результата выполнения последней операции;
- шаблоны записей.

Операции над данными выполняются навигационным подходом.

Основными командами DML DBTG являются:

- команды передвижения (FIND);
- команды извлечения (GET);
- команды обновления записей (ERASE, STORE, MODIFY);
- команды обновления наборов (CONNECT, DISCONNECT, RECONNECT).

Данные могут быть помещены в шаблон записи и все дальнейшие операции проводятся для заполненного шаблона:

; удалить клиентов с балансом равным нулю

MOVE 0 TO BALANS IN KLIENT

FIND FOR UPDATE ANY KLIENT USING BALANS

DOWHILE OS\_STAUS = 0

ERASE KLIENT

FIND FOR UPDATE DUBLICATE KLIENT USING BALANS

ENDDO

; удалить данные из набора

MOVE 254 TO NUMBER IN SCHET

FIND ANY SCHET USING NUMBER

DISCONNECT SCHET FROM KLIENT\_SCHET

Основные достоинства сетевой модели данных:

- высокая производительность в статичных системах с простой структурой;
- наличие ограничений целостности (опции удержания);
- наличие стандарта DBTG.

К основным недостаткам сетевой модели данных относятся:

- структура БД определяется запросами пользователей – смена запросов ведет к реорганизации всей БД;
- непредвиденные приложения работают непроизводительно – нет возможности расширения;
- определение схемы влияет на физический уровень БД (формат записей);
- сложность составления программ обработки данных.

#### 4.3.3. Реляционная модель данных

Базы данных, построенные на реляционной модели в настоящее время, наиболее востребованы. Популярность реляционной модели связана, прежде всего, с однородностью структурной организации данных.

Реляционная модель была предложена Э.Ф. Коддом (E.F. Codd, IBM, математик) в 1970 в статье «Реляционная модель данных для больших совместно используемых банков данных» - поворотный пункт в истории развития СБД (однако еще раньше была предложена модель основанная на множествах (Childs, 1968)).

Реляционная модель – формальная теория данных, основанная на некоторых положениях математики (теории множеств и предикативной

логике). Кодд впервые осознал, что математические дисциплины можно использовать, чтобы внести в область управления базами данных строгие принципы и точность (в то время не было устоявшихся терминов (однозначных определений) в этой области).

Идеи лежащие в основе реляционной модели и развитие ее оказали влияние на:

- обеспечение более высокой независимости от данных (чем в более ранних моделях данных);
- расширение языков управления данными (за счет включения операций над множествами)
- решение ряда вопросов семантики, проблем непротиворечивости и избыточности данных;
- развитие других областей (искусственного интеллекта, обработки естественных языков, разработки аппаратных систем).

Наиболее значительные исследования реляционной модели были проведены в рамках трех проектов:

- «System R» (конец 1970-х, г. Сан-Хосе, Калифорния, лаборатория IBM (руководитель проекта Astrahan)) – прототип истинной реляционной СУБД (был создан для проверки практичности реляционной модели), в ходе выполнения проекта получена информация о проблемах управления параллельностью, оптимизации запросов, управления транзакциями, обеспечения безопасности и целостности данных, восстановления, влияния человеческого фактора и разработки пользовательского интерфейса; проект стимулировал создание языка структурированных запросов (SQL) и ряда коммерческих СУБД (DB/2, ORACLE);
- «INGRES» (конец 1970-х, Калифорнийский университет, г. Беркли), практически те же цели разработки, получена академическая версия и коммерческие продукты;
- «Peterlee Relational Test Vehicle» (1976, г. Петерли, Великобритания, научный центр IBM) – более теоретический проект, основные цели –

решение задач оптимизации и обработки запросов, а также функциональные расширения системы.

Коммерческие системы на основе реляционной модели начали появляться в начале 1980-х (на сегодня - более нескольких сотен типов, хотя они и не удовлетворяют точному определению реляционной модели данных – нет необходимости следовать теории только «ради теории», главное назначение теории обеспечить базу для построения систем, практичных на 100%, кроме того некоторые вопросы реляционной теории до сих пор спорны). Также, многие нереляционные системы обеспечиваются реляционным пользовательским интерфейсом (независимым от базовой модели СУБД).

Были предложены некоторые расширения реляционной модели данных, предназначенные для наиболее полного и точного выражения смысла данных (Codd, 1979 и 1990), для поддержки объектно-ориентированных понятий (Stonebraker и Rowe, 1986), а также для поддержки дедуктивных возможностей (Gardarin и Valduriez, 1989).

Реляционная модель связана с тремя аспектами данных: структурой (представлением данных в БД), целостностью (ключи) и обработкой (операторы обработки данных). Для всех этих частей существуют свои специальные термины.

#### 4.3.3.1. Реляционные объекты данных (структура)

Реляционная модель данных основана на математическом понятии отношения (relation), физическим представлением которого является таблица. Все данные (описания объектов) в реляционной БД пользователь воспринимает как набор таблиц (множество отношений).

Краткое (неформальное, т.е. нестрогое) описание терминов:

- отношение – плоская таблица;
- кортеж – строка таблицы (не включая заголовков);

- кардинальное число – количество строк таблицы (без заголовка);
- атрибут – столбец таблицы (или поле строки);
- степень – количество столбцов таблицы;
- первичный ключ – уникальный идентификатор для таблицы;
- домен – общая совокупность допустимых значений.

Формальное описание терминов (теория) приведено ниже.

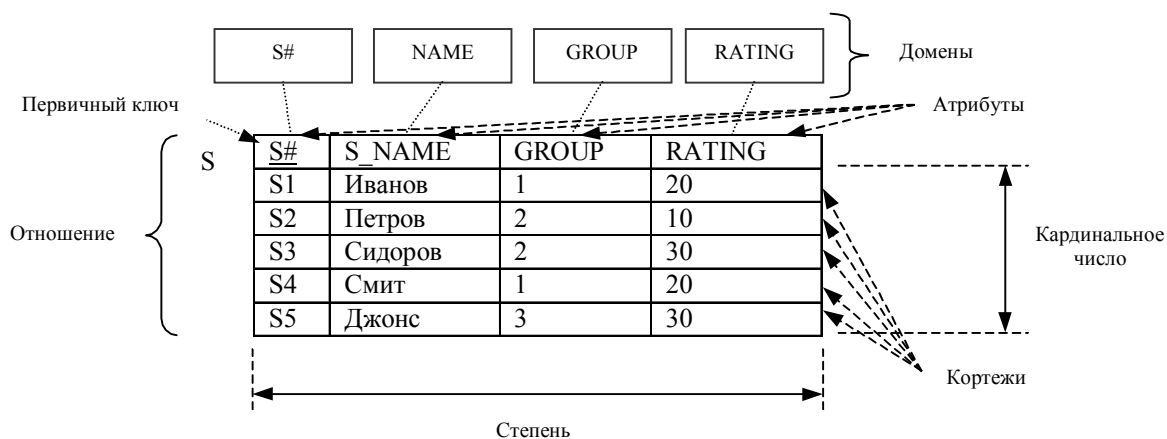


Рисунок 10. Реляционные объекты данных (отношение S).

#### 4.3.3.1.1. Домены

Домен – именованное множество скалярных значений (скаляр – наименьшая (атомарная) семантическая единица данных (например, «номер детали»)) одного типа (например, в домене «города» будут представлены все возможные названия городов, которыми может оперировать модель).

Значения атрибута (атрибутов) выбираются из множества значений домена, при этом каждый атрибут должен быть определен на единственном домене (т.е. в любой момент времени набор значений атрибута (в отношении) является некоторым подмножеством множества значений его домена, также в любой момент времени в домене могут существовать такие значения, которые



не являются значениями ни одного из атрибутов, соответствующих этому домену).

Значение доменов в реляционной модели:

- домены позволяют централизованно определять смысл и источник значений, которые могут получать атрибуты (в системе не будет значений связанных с ошибками написания или непонимания семантики значения (вместо «да» или «нет» пользователь может поставить прочерк (т.е. ситуация типа «sex - регулярно» ☺)));
- домены позволяют ограничивать сравнения (важнейшее свойство, позволяет избегать семантически некорректных операций), т.е. если значения атрибутов взяты из одного и того же домена, тогда операции над этими атрибутами (например, сравнение) будут иметь смысл, т.к. операция будет проводиться над между подобными по семантике атрибутами; если значения атрибутов взяты из различных доменов, то операции над ними, скорее всего, не будут иметь смысла (в основном предназначается для предотвращения грубых ошибок (механических опечаток или недостаточного знания модели)).

Домены имеют концептуальную природу. Для наличия возможности ограничения сравнений в системе нужно, чтобы домены, по крайней мере, были описаны в рамках определений БД, имели уникальные имена, и все атрибуты отношений имели бы ссылку на домен (при этом для упрощения системы наборы значений доменов могут в ней не храниться). На практике – большинство программных продуктов поддержки сравнений не обеспечивают, а смысл атрибутов и источник их значений определяется (иногда и только в случае небольшого количества значений) в других таблицах – не домены, а их замена (например, названия всех дней недели определяются в одной таблице (содержит только один атрибут), а в другой таблице пользователю разрешен выбор значений атрибута только из значений первой таблицы).

#### 4.3.3.1.2. Отношения

Отношение  $R$ , определенное на множестве доменов  $D_1, D_2, \dots, D_n$  (не обязательно различных), содержит две части: заголовок (строка заголовков столбцов в таблице) и тело (строки таблицы).

Заголовок содержит фиксированное множество атрибутов или пар вида  $\langle \text{имя\_атрибута} : \text{имя\_домена} \rangle$ :

$$\{ \langle A_1:D_1 \rangle, \langle A_2:D_2 \rangle, \dots, \langle A_n:D_n \rangle \},$$

причем каждый атрибут  $A_j$  соответствует одному и только одному из лежащих в основе доменов  $D_j$  ( $j=1,2,\dots,n$ ). Все имена атрибутов  $A_1, A_2, \dots, A_n$  - разные.

Тело содержит множество кортежей. Каждый кортеж, в свою очередь, содержит множество пар  $\langle \text{имя\_атрибута} : \text{значение\_атрибута} \rangle$ :

$$\{ \langle A_1:v_{i1} \rangle, \langle A_2:v_{i2} \rangle, \dots, \langle A_n:v_{in} \rangle \},$$

( $i=1,2,\dots,m$ , где  $m$  – количество кортежей в этом множестве). В каждом таком кортеже есть одна пара такая пара  $\langle \text{имя\_атрибута} : \text{значение\_атрибута} \rangle$ , т.е.  $\langle A_j:v_{ij} \rangle$ , для каждого атрибута  $A_j$  в заголовке. Для любой такой пары  $\langle A_j:v_{ij} \rangle$   $v_{ij}$  является значением из уникального домена  $D_j$ , который связан с атрибутом  $A_j$ .

Значения  $m$  и  $n$  называются соответственно кардинальным числом и степенью отношения  $R$ .

Для выше приведенного рисунка 10:

- домены –  $S\#, \text{NAME}, \text{GROUP}, \text{RATING}$ ;
- заголовок –  $\{ \langle S\#:S\# \rangle, \langle S\_NAME:NAME \rangle, \langle GROUP:GROUP \rangle, \langle RATING:RATING \rangle \}$ ;
- кортеж отношения (элемент тела) –  $\{ \langle S\#:S1 \rangle, \langle S\_NAME:'Иванов' \rangle, \langle GROUP:1 \rangle, \langle RATING:20 \rangle \}$ ;
- тело отношения (без указания имен доменов, порядок перечисления значений в кортежах должен совпадать с порядком перечисления атрибутов в заголовке) –  $\{ (S1,'Иванов',1,20), (S2,'Петров',2,10), \dots \}$

(S5, 'Джонс', 3, 30)}.

Свойства отношений:

- отношение имеет уникальное имя (т.е. отличное от имен других отношений в БД);
- в любом отношении нет одинаковых кортежей (это свойство следует из того факта, что тело отношения – математическое множество (кортежей), а множества в математике по определению не содержат одинаковых элементов); важное следствие из этого свойства – т.к. отношение не содержит одинаковых кортежей, то всегда существует первичный ключ отношения;
- в любом отношении кортежи не упорядочены (т.е. нет упорядоченности «сверху-вниз») (это свойство следует из того, что тело отношения – математическое множество, а простые множества в математике неупорядочены); следствие – нет понятий позиции кортежа и последовательности кортежей в отношении, кортеж в отношении всегда определяется по первичному ключу;
- в любом отношении атрибуты не упорядочены (т.е. нет упорядоченности «слева-направо») (это свойство следует из того, что заголовок отношения также определен как множество атрибутов); следствие – атрибут всегда определяется по имени, нет понятий позиции атрибута и последовательности атрибутов в отношении;
- в отношении все значения атрибутов атомарные (неделимые) (это следует из того, что все лежащие в основе домены содержат только атомарные значения); следствие – в каждой ячейке на пересечении столбца и строки в таблице расположено только одно значение, отношения при этом представлены в первой нормальной форме (не содержат групп повторения) – простота структуры отношения приводит к упрощению все системы (и облегчает операции с отношениями).

Как видно из свойств отношения, отношение и таблица на самом деле не одно и то же: таблица – конкретное представление (упорядоченное, обычно

графическое) абстрактного объекта отношение.

Для каждого отношения существует связанная с ним интерпретация (например, отношение  $S$  (см. рисунок 10) можно интерпретировать (не совсем формально) так: «студент с определенным номером ( $S\#$ ) имеет определенное имя ( $S\_NAME$ ), принадлежит определенной группе ( $GROUP$ ) и имеет определенную оценку ( $RATING$ ); не может быть нескольких студентов с одинаковыми номерами»). Эта интерпретация (или предикат) – функция значения истинности (после подстановки значений в эту функцию, она может принимать только два значения: истина или ложь). Отношение может содержать только те кортежи, для которых предикат является истиной.

Тогда для того чтобы добавить (вставить) новый кортеж в отношение, необходимо подставить значения из этого кортежа в функцию истинности для данного отношения и если значение этой функции - истина, то вставка кортежа (или возможность обновления данного отношения) будет допустима (при этом предикат составляет критерий возможности обновления (по крайней мере «правдоподобного»)). Для приведенного примера, СУБД будет считать кортеж приемлемым, если выполнены условия: значение  $S\#$  принадлежит домену  $S\#$ , значение  $S\_NAME$  принадлежит домену  $NAME$ , значение  $GROUP$  принадлежит домену  $GROUP$ , значение  $RATING$  принадлежит домену  $RATING$  и значение  $S\#$  является уникальным среди всех значений отношения - т.е. можно определить как логическое умножение (логическое «AND») всех правил целостности для отношения  $S$ .

Переменная отношения – именованное отношение, которое может иметь разные значения в разное время (однако все возможные значения этой переменной будут иметь одинаковые заголовки; этот термин предназначен для подчеркивания возможности изменения тела отношения, т.е. его динамичности (при этом термин «отношение» считается неудачным), при статичности описания отношения).

Различают множество видов отношений (теория):

- именованное отношение – переменная отношения, определенная с

помощью DDL в СУБД;

- базовое отношение – именованное отношение, которое является автономным (на практике – логически важные отношения, хранимые в БД (обычно соответствует некоторому объекту в концептуальной схеме));
- производное отношение – отношение, определенное посредством реляционного выражения через другие именованные отношения;
- выражаемое отношение – отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения (множество выражаемых отношений охватывает множество всех базовых и всех производных отношений);
- представление – именованное производное отношение (виртуально – представлено в системе только через определение в терминах других именованных отношений);
- снимок – именованное производное отношение, в отличие от представлений, снимок реален (не виртуален), т.е. имеет свои данные (отдельные от других именованных отношений);
- результат запроса – неименованное производное отношение, содержащее результат запроса (временное существование);
- промежуточный результат – неименованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение;
- хранимое отношение – отношение, которое поддерживается в физической памяти.

Между понятиями домена и унарного отношения (степень отношения равна единице (один атрибут)) есть значительная разница (теория): домены статичны (содержат все возможные значения и не меняют их описание), а отношения динамичны (изменяют содержимое со временем).

Реляционная БД – БД, воспринимаемая пользователем как набор нормализованных отношений (идея реляционной модели применима к внешнему и концептуальному уровню системы баз данных – т.е. на уровне

абстракции (логическом уровне)).

#### 4.3.3.1.3. Представления

Представление – динамический результат одной или нескольких реляционных операций над базовыми отношениями, является виртуальным отношением (т.е. реально в базе не существует, а содержит части реально существующих (базовых) отношений). Содержимое представления изменяется, если изменяется содержимое частей базовых таблиц, связанных с данным представлением. Если пользователи вносят допустимые изменения в содержимое представления, то эти изменения «одновременно» вносятся и в базовые отношения. Представление описывает не всю внешнюю модель пользователя (представление пользователя), а только его некоторую часть.

Представления предназначены для:

- скрывания некоторых частей базы данных от определенных пользователей (групп) (например, инспектор получит доступ только к объектам, которые он контролирует) – часть механизма защиты;
- организации доступа пользователей к данным наиболее удобным для них образом (например, переименование групп атрибутов для придания им определенной специфики – реализация поддержки логической независимости пользователя от данных);
- упрощения сложных операций с базовыми отношениями (для пользователя обращение к представлению будет выглядеть достаточно просто (т.е. он будет работать с ним как с одним отношением, реальные же операции над базовыми отношениями будут поддерживаться автоматически));
- изоляции пользователей от реорганизации концептуальной схемы (например, если базовое отношение при необходимости будет разбито на несколько других, то это будет скрыто от пользователя с помощью

представления).

По возможности обновления, представления делят на теоретически необновляемые, теоретически обновляемые и частично обновляемые (т.к. изменения в представлении могут быть отражены на базовые отношения только при определенных условиях (например, если представление построено только простого запроса к единственному базовому отношению и содержит ключ базового отношения – то оно допускает обновление)).

#### 4.3.3.2. Целостность реляционных данных

Целостность данных предназначена для сохранения в БД «отражения действительности реального мира», т.е. устранения недопустимых конфигураций (состояний) значений и связей (например, вес детали может быть только положительным), которые не имеют смысла в реальном мире (не следует путать с «безопасностью» - хоть и есть некоторое сходство, но безопасность относится к защите от несанкционированного доступа, а целостность – защита БД от санкционированных пользователей). Целостность данных – наиболее изменчивая часть, т.к. эта часть реляционной модели имеет наименее твердую научную основу (по сравнению с другими частями модели).

Правила целостности данных можно разделить на:

- специфические или корпоративные ограничения целостности – правила, определяемые пользователями или администратором БД, которые указывают дополнительные ограничения, специфические для конкретных БД (например, максимальные и минимальные значения некоторых атрибутов (диапазон оценок));
- общие правила целостности – правила, которые применимы к любой реляционной БД (относятся к потенциальным (первичным) и к внешним ключам).

#### 4.3.3.2.1. Потенциальные ключи

Пусть  $R$  – некоторое отношение. Тогда потенциальный ключ  $K$  для  $R$  – это подмножество множества атрибутов  $R$ , обладающее следующими свойствами:

- свойством уникальности – нет двух различных кортежей в отношении  $R$  (в текущем значении) с одинаковым значением  $K$  (каждое отношение имеет как минимум один потенциальный ключ, т.к. не содержит одинаковых кортежей, важно чтобы уникальность соблюдалась не только для конкретного содержания отношения (на данный момент времени), а всегда (т.е. для множества всех возможных значений отношения в любой момент времени) – для этого при определении потенциального ключа важно знать смысл атрибутов отношения);
- свойством избыточности (неприводимости) – никакое из подмножеств  $K$  не обладает свойством уникальности (если определенный «потенциальный ключ» на самом деле не является избыточным, то система не сможет обеспечить должным образом соответствующее ограничение целостности (например, если вместо номера студента  $\{S\# \}$ , потенциальный ключ будет определен как номер студента и номер группы  $\{S\#, GROUP\}$ , то уникальность номера студента будет поддерживаться системой в более узком (локальном) смысле – только в пределах одной группы)).

Простой потенциальный ключ – потенциальный ключ, состоящий из одного атрибута.

Составной потенциальный ключ – потенциальный ключ, состоящий более чем из одного атрибута.

Логическое понятие потенциального ключа не следует путать с физическим понятием уникального индекса (это разные вещи с точки зрения реляционной модели данных, хотя на практике они обычно совмещаются).



Потенциальные ключи обеспечивают основной механизм адресации на уровне кортежей в реляционной системе (единственный гарантируемый системой способ точно указать некоторый кортеж – это указать значение некоторого потенциального ключа).

Отношение может иметь несколько (хотя это довольно редко применяется) потенциальных ключей. По традиции (исторически) один из потенциальных ключей должен быть выбран в качестве первичного ключа отношения (отношение всегда имеет первичный ключ), а остальные потенциальные ключи будут называться альтернативными ключами. Выбор одного из потенциальных ключей в качестве первичного – вопрос философский и фактически выходит за рамки реляционной модели данных.

#### 4.3.3.2.2. Внешние ключи

Пусть  $R_2$  – базовое отношение. Тогда внешний ключ  $FK$  в отношении  $R_2$  – это подмножество множества атрибутов  $R_2$ , такое что:

- существует базовое отношение  $R_1$  ( $R_1$  и  $R_2$  не обязательно различны) с потенциальным ключом  $K$ ;
- каждое значение  $FK$  в текущем значении  $R_2$  всегда совпадает со значением  $K$  некоторого кортежа в текущем значении  $R_1$  (обратное не требуется, т.е. потенциальный ключ, соответствующий данному внешнему ключу, может содержать значение, которое в данный момент не является значением внешнего ключа).

Внешний ключ будет составным (т.е. будет состоять из более чем одного атрибута), тогда и только тогда, когда соответствующий потенциальный ключ также будет составным. Соответственно, внешний ключ будет простым тогда и только тогда, когда соответствующий потенциальный ключ также будет простым.

Каждый атрибут, входящий в внешний ключ, должен быть определен на

том же домене, что и соответствующий атрибут соответствующего потенциального ключа.

Внешним ключом может быть любой атрибут (или сочетание атрибутов) отношения, а не только компонент потенциального ключа.

Значение внешнего ключа называется ссылкой к кортежу, содержащему соответствующее значение потенциального ключа (ссылочный кортеж или целевой кортеж). Отношение, которое содержит внешний ключ, называется ссылающимся отношением, а отношением, которое содержит соответствующий потенциальный ключ – ссылочным или целевым отношением.

Для упрощения понимания, ссылки можно представлять с помощью ссылочных диаграмм – каждая стрелка обозначает внешний ключ в отношении, из которого эта стрелка выходит, этот ключ ссылается на первичный (потенциальный) ключ отношения, на который стрелка указывает (допустимо дополнительно указывать над стрелками атрибуты, которые составляют внешний ключ):

$$S \leftarrow SP \rightarrow P.$$

Отношение может быть одновременно ссылочным и ссылающимся (цепочка стрелок из  $R_n$  в  $R_1$  называется ссылочный путь):

$$R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow \dots \rightarrow R_2 \rightarrow R_1.$$

Также выделяют самоссылающиеся отношения (в них возникают ссылочные циклы):

$$R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow \dots \rightarrow R_2 \rightarrow R_1 \rightarrow R_n.$$

#### 4.3.3.2.3. Ссылочная целостность

Правило ссылочной целостности – база данных не должна содержать несогласованных значений внешних ключей (здесь «несогласованное значение внешнего ключа» - это значение внешнего ключа, для которого не существует отвечающего ему значения соответствующего потенциального ключа в

соответствующем целевом отношении) – т.е. если В ссылается на А, тогда А должно существовать.

Понятия «внешний ключ» и «ссылочная целостность» определены в терминах друг друга, т.е. «поддержка внешних ключей» и «поддержка ссылочной целостности» означают одно и то же.

Для поддержки ссылочной целостности необходимо внести компенсацию в БД в случаях (не в рамках реляционной модели):

- при удалении объекта ссылки внешнего ключа (возможные действия: ограничить – запрет операции удаления объекта, до тех пор, пока на этот объект есть ссылки; каскадировать – при удалении объекта удалить все объекты, ссылающиеся на него);
- при попытке обновить потенциальный ключ, на который ссылается внешний ключ (возможные действия: ограничить – запрет операции обновления объекта, до тех пор, пока на этот объект есть ссылки; каскадировать – при обновлении объекта обновить внешний ключ во всех объектах, ссылающиеся на него).

На практике, кроме перечисленных действий («ограничить» и «каскадировать»), должна быть возможность вызова процедуры баз данных (хранимой процедуры или процедуры триггеров), для частного решения возникшей проблемы.

С логической точки зрения, все операции с объектами БД (обновление и удаление) всегда атомарны (все или ничего), даже если они выполняются каскадно.

При рассмотрении реакции системы на операции с объектом (обновление или удаление) следует рассматривать все возможные ссылочные пути. Для самоссылающихся отношений некоторые проверки ограничений не могут быть выполнены во время одного обновления (они обычно откладываются на более позднее время – время фиксации).

#### 4.3.3.2.4. NULL-значения

NULL-значения (определитель NULL) введены для обозначения значений атрибутов, которые на настоящий момент неизвестны или неприемлемы для некоторого кортежа – т.е. логическая величина «неизвестно». Это не значение по умолчанию, а отсутствие какого-либо значения (например, данные об адресе нового сотрудника неизвестны (на данный момент)).

Для каждого атрибута должно быть установлено, может ли он принимать NULL-значения или нет.

Применение определителя NULL может вызвать проблемы на этапе реализации системы, т.к. реляционная модель основана на исчислении предикатов первого порядка (они обладают двузначной (булевой) логикой (т.е. могут иметь только два значения: «истина» или «ложь»)), а применение NULL-значений приводит к работе с логикой более высокого порядка (трехзначной или даже четырехзначной (Codd)).

Использование NULL-значений – вопрос теоретически спорный (Codd рассматривает NULL-значения как неотъемлемую часть реляционной модели, Date – «являются фундаментальным заблуждением и не должны иметь места в чистой формальной системе», вместо них можно использовать значение по умолчанию (упрощает правила целостности)). Не во всех реляционных системах поддерживается работа с определителем NULL.

NULL-значения влияют на концепции потенциальных и внешних ключей реляционной модели данных:

- 1) целостность объектов – в реляционной модели данных ни один атрибут потенциального ключа (в некоторых источниках – только первичные ключи) базового отношения не может содержать NULL-значений (т.к. реляционная БД не должна хранить информацию о чем-то, чего мы не можем определить и однозначно сослаться).
- 2) ссылочная целостность – если в реляционной модели в отношении существует внешний ключ, то значение внешнего ключа должно либо

соответствовать значению потенциального ключа некоторого кортежа в целевом отношении, либо задаваться определителем NULL (здесь NULL-значение обозначает не «значение неизвестно», а – «значение не существует»).

Определение внешнего ключа с учетом NULL-значений:

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

- существует базовое отношение R1 (R1 и R2 не обязательно различны) с потенциальным ключом K;
- каждое значение FK в текущем значении R2 или является NULL-значением, или совпадает со значением K некоторого кортежа в текущем значении R1.

При этом вносится дополнительное действие для поддержки ссылочной целостности в случаях (не в рамках реляционной модели):

- при удалении объекта ссылки внешнего ключа (дополнительное действие: аннулировать – для всех ссылок на объект устанавливаются NULL-значения, а затем сам объект удаляется);
- при попытке обновить потенциальный ключ, на который ссылается внешний ключ (дополнительное действие: аннулировать – для всех ссылок на объект устанавливаются NULL-значения, а затем сам объект обновляется).

#### 4.3.3.2.5. Третье средство обеспечения общей целостности в реляционной модели

Домены (ограничения домена) – обеспечивают целостность атрибутов (т.к. значения каждого атрибута берутся из соответствующего домена).

### 4.3.3.3. Реляционные операторы

Третья часть реляционной модели – реляционные операторы. Реляционные операторы – формальные (не дружественные пользователю) языки, использовались в реляционных базах данных в качестве основы для разработки других языков управления данными (более высокого уровня), сейчас представляют интерес, потому что описывают базовые реляционные операции.

Реляционные операторы делят на: реляционную алгебру и реляционное исчисление.

#### 4.3.3.3.1. Реляционная алгебра

Реляционная алгебра – теоретический язык операций над отношениями, предложен Коддом (1970). Результат операций реляционной алгебры также представляет собой отношение (свойство замкнутости), поэтому результаты одной операции могут стать исходными данными для другой операции (вложенные выражения). Реляционная алгебра является языком последовательного использования отношений, в котором все кортежи обрабатываются одной командой.

Операции реляционной алгебры делят на основные и дополнительные.

Основные операции реляционной алгебры:

1) Выборка (selection) – унарная операция (выполняется над одним отношением), определяет результирующее отношение, которое содержит только те кортежи отношения R, которые удовлетворяют заданному условию (предикату):

$\sigma_p(R)$ , где p – предикат (более сложные предикаты создаются с помощью логических операторов ( $\wedge$  - AND,  $\vee$  - OR,  $\sim$  - NOT)).

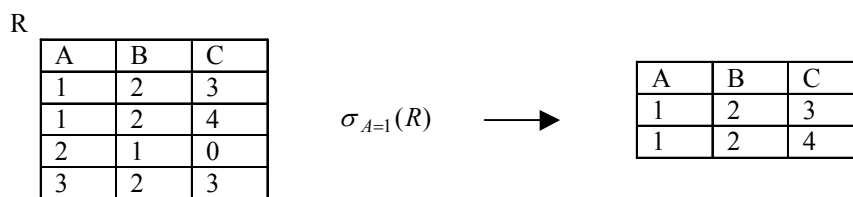


Рисунок 11. Выборка.

2) Проекция (projection) – унарная операция, определяет новое отношение, содержащее вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дубликатов (кортежей-дубликатов):

$\Pi_{A_1, A_2, \dots, A_n}(R)$ , где  $(A_1, A_2, \dots, A_n)$  – имена атрибутов.

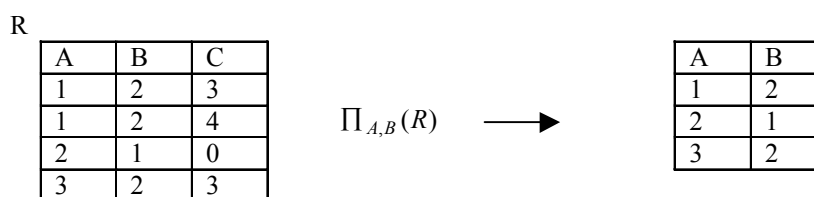


Рисунок 12. Проекция.

3) Декартово произведение (cartesian product) – определяет новое отношение, которое является результатом сцепления (конкатенации) каждого кортежа из отношения R с каждым кортежем из отношения S. Если одно отношение имеет I кортежей и N атрибутов, а другое – J кортежей и M атрибутов, то отношение их с декартовым произведением будет содержать  $(I \cdot J)$  кортежей и  $(N+M)$  атрибутов. Если исходные отношения содержат атрибуты с одинаковыми именами, то для обеспечения уникальности имен атрибутов в отношении они будут переименованы (обычно включая имя отношения как префикс):

$R \times S$ , где R и S – имена отношений.

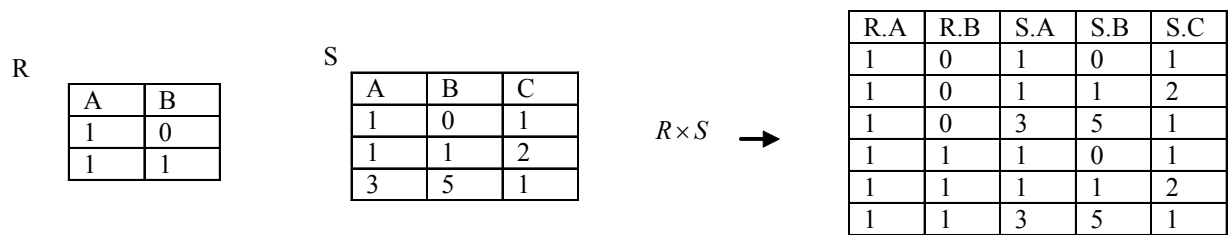


Рисунок 13. Декартово произведение.

4) Объединение (union) – определяет новое отношение, которое является результатом сцепления (конкатенации) кортежей отношений R и S, и исключения в полученном отношении кортежей-дубликатов. Отношения R и S должны быть совместимы по объединению, т.е. они должны иметь одинаковое количество атрибутов, причем парные атрибуты (совмещающиеся при склейке) должны быть определены на одном домене. Если исходные отношения содержат I и J кортежей, то результирующее отношение будет содержать максимум (I+J) кортежей:

$$R \cup S, \text{ где } R \text{ и } S \text{ – имена отношений.}$$

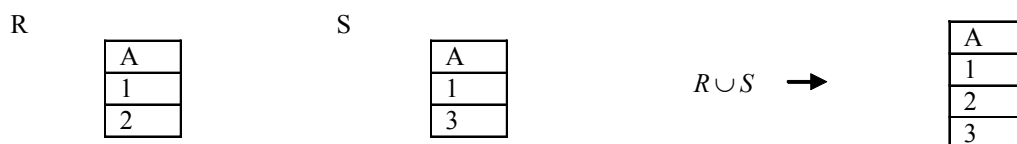


Рисунок 14. Объединение.

5) Разность (set difference) – определяет новое отношение, которое состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. При этом отношения R и S должны быть совместимы по объединению:

$$R - S, \text{ где } R \text{ и } S \text{ – имена отношений.}$$



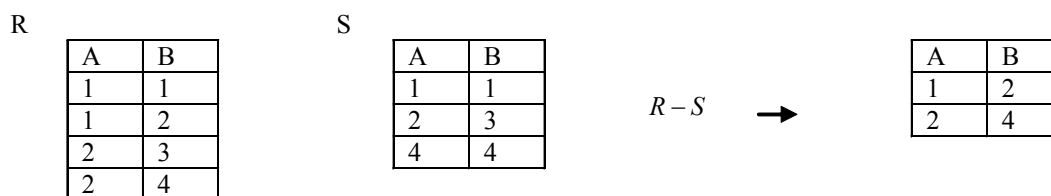


Рисунок 15. Разность.

Дополнительные операции реляционной алгебры:

б) Операции соединения (join) – одна из самых важных операций реляционной алгебры (является производной от операции декартова произведения, наиболее трудоемкая операция (трудно реализуется и малая производительность)). Типы операций соединения:

- Тета-соединение ( $\Theta$ -join) – определяет отношение, которое содержит кортежи из декартова произведения отношений R и S, удовлетворяющие предикату F. Предикат F имеет вид  $R.a_i \Theta S.b_i$ , где вместо  $\Theta$  может быть указан один из операторов сравнения ( $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $=$  или  $\sim$ ):

$$R \bowtie_{\Theta F} S = \sigma_F (R \times S).$$

Степень  $\Theta$ -соединения – сумма степеней операндов.

- Если предикат F содержит только оператор равенства ( $=$ ), то такое тета-соединение называется соединением по эквивалентности (equal-join).

- Естественное соединение (natural join) – соединение по эквивалентности двух отношений R и S, выполненное по всем общим атрибутам x, из результатов которого исключается по одному экземпляру каждого общего атрибута. Степень естественного соединения равна сумме степеней операндов отношений R и S минус количество общих атрибутов x (если отношения R и S не имеют общих имен атрибутов, то естественное соединение эквивалентно декартову произведению):

$$R \bowtie S, \text{ где } R \text{ и } S \text{ – имена отношений.}$$

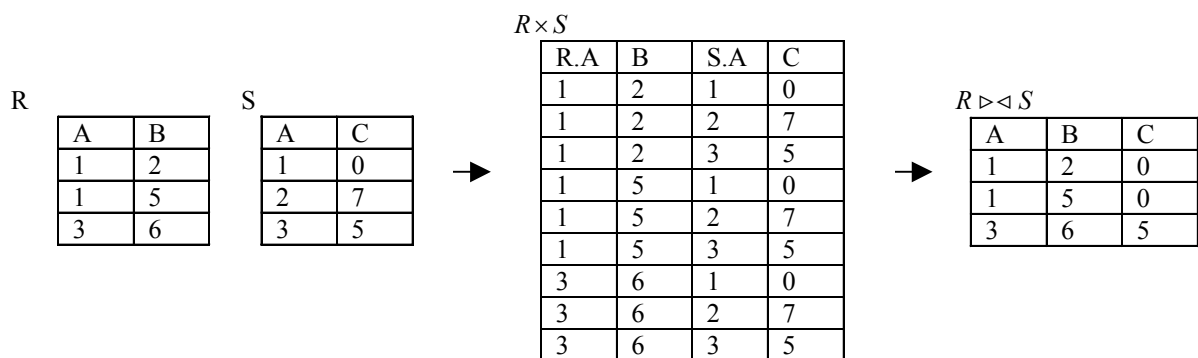


Рисунок 16. Естественное соединение.

- Внешнее соединение (outer join) - три вида:

-- левое внешнее соединение – соединение, при котором кортежи отношения R (находящегося в выражении слева), не имеющие совпадающих значений в общих столбцах отношения S (находящегося в отношении справа), также включаются в результирующее отношение (отсутствующие значения заменяются определителем NULL; в таком соединении сохраняются кортежи, которые были бы утрачены при использовании других типов соединения):

$R \supset\lrcorner S$ , где R и S – имена отношений.

-- правое внешнее соединение – соединение, при котором кортежи отношения S (находящегося в выражении справа), не имеющие совпадающих значений в общих столбцах отношения R (находящегося в отношении слева), также включаются в результирующее отношение (отсутствующие значения заменяются определителем NULL):

$R \supset\lrcorner S$ , где R и S – имена отношений.

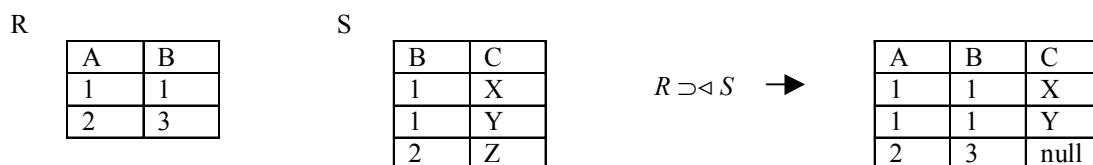


Рисунок 17. Левое внешнее соединение.

-- полное внешнее соединение – соединение, при котором кортежи

отношения R (находящегося в выражении слева), не имеющие совпадающих значений в общих столбцах отношения S (находящегося в отношении справа), а также кортежи отношения S (находящегося в выражении справа), не имеющие совпадающих значений в общих столбцах отношения R (находящегося в отношении слева), также включаются в результирующее отношение (отсутствующие значения заменяются определителем NULL):

$$R \supset S, \text{ где } R \text{ и } S \text{ – имена отношений.}$$

- Полусоединение (semi join) – определяет отношение, которое содержит те кортежи отношения R, которые входят в соединение отношений R и S:

$$R \triangleright_F S = \prod_A (R \triangleright_{\triangleleft_F} S), \text{ где } A \text{ – множество всех атрибутов в отношении } R.$$

7) Пересечение (intersection) – определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S (отношения R и S должны быть совместимы по объединению):

$$R \cap S = R - (R - S), \text{ где } R \text{ и } S \text{ – имена отношений.}$$

8) Деление (division) – определяет отношение, которое содержит набор кортежей отношения R, определенных на множестве атрибутов C, которые соответствуют комбинации всех кортежей отношения S:

$$R \div S = \prod_C (R) - \prod_C ((S \times \prod_C (R)) - R), \text{ где } R \text{ и } S \text{ – имена отношений.}$$

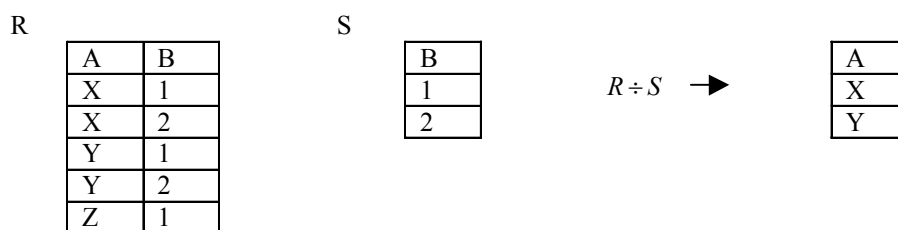


Рисунок 18. Деление.

Операции выборки, проекции, декартова произведения, объединения и разности – примитивные операции (их нельзя выразить через другие операции реляционной алгебры).

Операции соединения, пересечения и деления – не примитивные (т.е. их можно выразить через основные операции), но т.к. эти операции на практике часто используются, то они выделены в отдельную группу (чтобы не выражать их каждый раз – что-то типа макросов).

Цель реляционной алгебры – запись выражений для:

- определения области выборки – определение данных для их выбора;
- определение области обновления – определение данных для их вставки, изменения или удаления;
- определения представлений и снимков – определение данных для представления в виде некоторого отношения;
- определение правил безопасности – определение данных, для которых осуществляется контроль доступа;
- определение требований устойчивости – определение данных, которые входят в область для некоторых операций управления одновременным доступом;
- определение правил корпоративной целостности.

#### 4.3.3.3.2. Реляционное исчисление

Реляционное исчисление происходит от части символьной логики, которая называется исчислением предикатов.

Предикат (в логике первого порядка или теории исчисления предикатов) – истинностная функция с аргументами (при подстановке вместо аргументов значений эта функция становится выражением, называемым суждением, которое может быть истинным или ложным).

Если предикат содержит переменную (например, «*x* является студентом этой группы»), то у этой переменной должна быть соответствующая область определения (при подстановке одних значений из ее области определения суждение может быть истинным, а при подстановке других - ложным).

Если  $P$  – предикат, то множество всех значений переменной  $x$ , при которых  $P$  становится истинным:

$$\{x \mid P(x)\}.$$

Предикаты могут соединяться с помощью логических операторов  $\wedge$  (AND),  $\vee$  (OR) и  $\sim$  (NOT) (при этом образуются составные предикаты).

Реляционное исчисление существует в двух формах:

### 1) Реляционное исчисление кортежей (Кодд).

Задача реляционного исчисления кортежей – нахождение таких кортежей, для которых предикат является истинным.

Реляционное исчисление кортежей основано на переменных кортежа – переменные, областью определения которых является указанное отношение и допустимыми значениями могут быть только кортежи данного отношения.

Предикат  $P$  называется формулой (в математической логике – правильно построенной формулой (WFF – well formed formula)).

Для указания количества экземпляров (кортежей), к которым может быть применен предикат, в формулах могут использоваться два вида кванторов:

- квантор существования ( $\exists$  - «существует») – означает, что в отношении есть хотя бы один кортеж, для которого формула – истинна;
- квантор общности ( $\forall$  - «для всех») – означает, что формула будет истинна только для всех кортежей отношения.

Переменные кортежа называются свободными, если они не квалифицируются кванторами  $\exists$  и  $\forall$ , иначе они называются связанными переменными.

В реляционном исчислении не каждая последовательность формул является допустимой (т.е. не двусмысленной и не бессмысленной). Правила построения WFF:

- если  $P$  является  $n$ -арной формулой (предикатом с  $n$  аргументами), а  $t_1, t_2, \dots, t_n$  – это константы или переменные, то выражение  $P(t_1, t_2, \dots, t_n)$  – правильно построенная формула;

- если  $t1$  и  $t2$  – константы или переменные из одного домена, а  $\Theta$  представляет собой один из операторов сравнения ( $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $=$ ,  $\sim=$ ), то выражение  $t1\Theta t2$  – правильно построенная формула;
- если выражения  $F1$  и  $F2$  – формулы, то их конъюнкция (логическое «И») означается как  $F1 \wedge F2$ , дизъюнкция (логическое «ИЛИ») -  $F1 \vee F2$ , а отрицание -  $\sim F1$ ;
- если выражение  $F1$  является формулой со свободной переменной  $X$ , то выражения  $\exists F(X)$  и  $\forall F(X)$  - также являются формулами;
- все результирующие значения должны входить в область определения формулы (обычно область определения задается как RANGE OF переменная\_кортежа IS имя\_отношения1 (выражение\_исчисления\_кортежей1), имя\_отношения2 (выражение\_исчисления\_кортежей2), ... (замечание: все отношения при этом должны иметь идентичные заголовки)).

Например, есть два отношения SN(S#, NAME, CITY, G#) – данные о студенте и SG(G#, NAME, CITY) – данные о группе:

1) RANGE OF N IS SN, {N | N.S# > 1000} – получить значения всех кортежей (N – переменная кортежа, содержит текущий кортеж определенный на отношении SN в текущее время), значение атрибута S# у которых больше 1000;

2) RANGE OF N IS SN, RANGE OF G IS SG, {N.NAME, N.CITY |  $\exists G$  (G.G# = N.G#  $\wedge$  G.NAME = “012345”)} – получить имя студента и город (значения NAME и CITY отношения SN), если есть хотя бы один кортеж отношения SG, в котором значение атрибута G# равно значению G# в текущем кортеже N и номер группы GROUP = «012345»;

3) RANGE OF N IS SN, RANGE OF G IS SG, {N.NAME |  $\forall G$  (G.CITY  $\sim=$  N.CITY)} – получить имена студентов, город проживания которых не совпадает с расположением ни одной группы (что эквивалентно {N.NAME |  $\sim \exists G$  (G.CITY = N.CITY)}).

Опасные формулы – формулы без ограничения диапазона возможных

значений кортежей (при этом может быть получена бесконечная последовательность кортежей, что недопустимо).

## 2) Реляционное исчисление доменов (Лакруа и Пиро).

В реляционном исчислении доменов используются переменные, значения которых берутся из доменов (а не из кортежей отношений). Если  $P(a_1 : d_1, a_2 : d_2, \dots, a_n : d_n)$  – предикат с переменными домена  $d_1, d_2, \dots, d_n$  (определенными на атрибутах  $a_1, a_2, \dots, a_n$  соответственно), то множество переменных домена, для которых предикат истинен:

$$\{d_1, d_2, \dots, d_n \mid P(a_1 : d_1, a_2 : d_2, \dots, a_n : d_n)\}.$$

Задача реляционного исчисления доменов – проверить условие принадлежности, чтобы определить, принадлежат ли значения указанному отношению (например, выражение  $R(x : r_x, y : "y_1")$  считается истинным тогда и только тогда, когда в отношении  $R$  имеется кортеж со значениями  $r_x$  (переменная домена – содержит текущее значение атрибута) и  $"y_1"$  (литерал - константа) в его атрибутах  $x$  и  $y$  соответственно.

Например, есть отношение  $SN(S\#, NAME, RATING)$  – данные о студенте:

$\{N \mid \exists R (SN (NAME : N, RATING : R) \wedge R > 7.8)\}$  – получить значения всех студентов с рейтингом больше 7,8 (проверка существует ли кортеж с атрибутами  $NAME$  (переменная  $N$ ) и  $RATING$  (переменная  $R$ ) в отношении  $SN$ , причем значение атрибута  $RATING$  в этом кортеже  $> 7,8$ ).

Если формулы безопасны, то реляционное исчисление доменов семантически эквивалентно реляционному исчислению кортежей.

### 4.3.3.3.3. Связь реляционного исчисления и реляционной алгебры

Существует алгоритм («алгоритм редукции Кодда»), с помощью которого выражение реляционного исчисления (кортежей) можно преобразовать в семантически эквивалентное выражение реляционной алгебры (т.е.

реляционная алгебра и реляционное исчисление эквивалентны – каждому выражению в реляционной алгебре соответствует эквивалентное выражение в исчислении, и наоборот).

Так как реляционное исчисление носит описательный характер (описывает, что надо получить) – ближе к естественным языкам, а реляционная алгебра носит предписывающий характер (указывает, каким образом (как) должен быть получен результат) – ближе к машинным языкам, и между ними есть однозначное соответствие, то возможна компьютерная обработка запросов написанных с помощью реляционного исчисления (запрос берется у пользователя как выражение исчисления (легче описывается), к этому запросу применяется алгоритм редукции и вычисляется полученное реляционное выражение (+ оптимизация)).

Реляционно полный язык – язык по своим возможностям не уступающий реляционному исчислению или реляционной алгебре (т.е. реализующий операции эквивалентные основным операциям реляционной алгебры). Желательно чтобы в таком языке была и вычислительная полнота (наличие большинства вычислимых функций (+, -, \* и т.п.)).

#### 4.3.3.4. Перевод ER-диаграммы в реляционную модель данных

Преобразование ER-диаграммы в реляционную модель (схему) позволяет выполнить переход от концептуальной фазы проектирования к логической. Алгоритм преобразования следующий:

- 1) Каждая простая сущность (объект на ER-диаграмме) превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.
- 2) Каждый атрибут становится возможным столбцом с тем же именем; при этом может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать



неопределенные (NULL) значения; столбцы, соответствующие обязательным атрибутам, - не могут.

- 3) Компоненты уникального идентификатора сущности (уникальные атрибуты объекта) превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.
- 4) Связи многие-к-одному (и в частности - один-к-одному) становятся внешними ключами. Внешний ключ добавляется в виде столбца (столбцов) в таблицу, соответствующую объекту со стороны «многие» связи. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения. Связи этого типа, имеющие дополнительные атрибуты, желательно реализовывать через промежуточную таблицу, как описано в п. 5 алгоритма.
- 5) Связи многие-ко-многим реализуются через промежуточную таблицу. Эта таблица будет содержать внешние ключи на соответствующие объекты, а также другие атрибуты, которые описывают указанную связь. Первичный ключ промежуточной таблицы должен включать в себя как минимум внешние ключи на объекты связи. Также в первичный ключ должны входить атрибуты временной связи (например, дата события), для наиболее полной реализации модели.
- 6) Если в концептуальной схеме присутствовали подтипы, то возможны два способа: (а) все подтипы в одной таблице и (б) для каждого подтипа - отдельная таблица. При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться

представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа. При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

- 7) Имеется два способа работы при наличии исключаящих связей: (а) общий домен и (б) явные внешние ключи. Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи. Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Данный алгоритм применяется ко всем сущностям и связям ER-диаграммы.

Наиболее частые ошибки выполнения алгоритма перевода относятся к формированию ссылок:

- несоответствие типов внешних ключей (ссылок) и типов ссылочных атрибутов;
- ссылка не на первичный ключ, что порождает неоднозначные связи и не соответствует определению внешнего ключа;
- ссылки между первичными ключами отношений, что приводит к формированию связей один-к-одному, которые используются крайне редко (основная ошибка здесь - при формировании ссылки может не учитываться семантика атрибутов);

- в промежуточных таблицах, при реализации связей многие-ко-многим, первичный ключ включает в себя только часть внешних ссылок, что ограничивает число связей между объектами до уровня связей один-ко-многим, и даже один-к-одному.

Дальнейшее уточнение реляционной схемы можно проводить с использованием механизма нормализации реляционных данных, который приведен ниже.

#### 4.3.3.5. Нормализация реляционных данных

Нормализация – метод создания набора отношений с заданными свойствами на основе некоторых требований к данным. Процесс нормализации предложен Коддом (1972 г.). Процесс нормализации – формальный метод для оптимизации столбцов отношений и устранения аномалий.

##### 4.3.3.5.1. Избыточность данных и аномалии обновления

Основная цель проектирования реляционной БД – группирование атрибутов в отношениях таким образом, чтобы минимизировать избыточность данных (сокращение объема вторичной памяти для хранения БД) и повышение надежности при работе с данными. Обычно процесс проектирования отношений реляционной БД ведется на основе разработанной ER-диаграммы или на основе просто здравого смысла разработчика. В общем случае при таком подходе расположение атрибутов в отношениях неоптимальное.

В качестве примера рассмотрим отношение с явной избыточностью данных

DB\_STUD {NAME, SUBJECT, #PHONE, SPEC, TEACHER, RATING},  
которое описывает данные об обучении студента с именем NAME по предмету

SUBJECT, при этом RATING – оценка по предмету, TEACHER – экзаменатор, SPEC – кафедра специализации, #PHONE – номер телефона студента. Первичным ключом в этом отношении выбрана комбинация NAME, SUBJECT. В данном отношении существуют следующие зависимости (фактически ограничения на связи между атрибутами для заданной модели внешнего мира):

- у одного студента должен быть только один номер телефона;
- у одного студента должен быть только одна кафедра специализации;
- у одного студента по предмету может быть только одна оценка;
- предмет на одной кафедре принимает только один экзаменатор.

Для указанного отношения избыточными явно будут названия кафедр и номера телефонов для каждого повторяющегося студента, а также указания экзаменаторов для предметов изучаемых на каждой отдельной кафедре.

При работе с отношениями, содержащими избыточные данные, могут возникать проблемы – аномалии обновления. Аномалии обновления делят на три вида:

1) Аномалии вставки – возникают при добавлении новых несогласованных данных (нарушающих целостность данных в отношении); для выше приведенного примера к аномалиям вставки будут относиться:

- ввод другого номера телефона для конкретного студента, отличного от ранее введенного номера, порождает неоднозначность данных и нарушает зависимость между атрибутами;
- ввода нового экзаменатора по конкретному предмету, изучаемому на кафедре;
- в данное отношение нельзя внести данные только о экзаменаторе по предмету, т.к. в данном случае нельзя определить данные первичного ключа отношения (имя студента).

2) Аномалии изменения – возникают при изменении части ранее введенных данных; частичное обновление сведений (например, о смене номера телефона студента) приведет к нарушению целостности данных отношения (перестанет выполняться оговоренная выше зависимость).

3) Аномалии удаления – возникают при удалении строк из отношений; для приведенного примера – удаление последнего студента изучающего конкретный предмет приведет к уничтожению данных экзаменатора.

Интуитивно понятно, что решением проблем избыточности и аномалий для выше приведенного примера может стать разделение отношения на такие отношения, в которых избыточности не будет. Для выполнения такого процесса необходимо выявить все зависимости между атрибутами отношения (потеря одной такой зависимости меняет модель внешнего мира).

#### 4.3.3.5.2. Функциональные зависимости

Выявление смысловой зависимости между данными – один из способов формализации смысловой информации о данных.

Функциональная зависимость описывает связь типа многие-к-одному между атрибутами отношения, где много – детерминант функциональной зависимости. Функциональная зависимость является семантическим свойством атрибутов отношения.

Если в отношении  $R$ , содержащем атрибуты  $A$  и  $B$ , атрибут  $B$  функционально зависит от атрибута  $A$  ( $A$  является детерминантом атрибута  $B$ )

$$A \rightarrow B,$$

то в каждом кортеже этого отношения каждое конкретное значение атрибута  $A$  всегда связано только с одним значением атрибута  $B$ . Атрибуты  $A$  и  $B$  могут быть составными атрибутами. Для выше приведенного примера (п. 4.3.3.5.1), номер телефона функционально зависит от имени студента, т.е.  $NAME \rightarrow \#PHONE$ .

Особенности функциональных зависимостей, лежащие в основе процесса нормализации:

- функциональная зависимость является специализированным правилом целостности – она накладывает ограничения на допустимые значения

атрибутов отношений; эту особенность можно использовать при обновлении БД, т.к. зная какие функциональные зависимости есть в отношении, можно проанализировать нарушат ли новые данные целостность данных отношения;

- функциональная зависимость является обобщением понятия потенциального ключа; функциональные зависимости позволяют определить все потенциальные ключи отношения (и соответственно – первичный ключ): все атрибуты отношения, которые не являются частью первичного (или потенциального) ключа, должны функционально зависеть от этого ключа; если не все остальные атрибуты отношения зависят от некоторого детерминанта, то этот детерминант не является потенциальным ключом этого отношения.

Одни функциональные зависимости подразумевают другие зависимости. Для данного множества зависимостей  $S$  замыканием  $S^+$  называется множество всех функциональных зависимостей, подразумеваемых зависимостями множества  $S$ . Множество  $S$  обязательно является подмножеством собственного замыкания  $S^+$ . Правила логического вывода Армстронга (аксиомы Армстронга) обеспечивают исчерпывающую и полную основу для вычисления замыкания  $S^+$  для заданного множества  $S$ :

- рефлексивность  $X \rightarrow X$  ;
- пополнение  $X \rightarrow Y \Rightarrow XZ \rightarrow Y$  ;
- аддитивность  $(X \rightarrow Y) \text{ and } (X \rightarrow Z) \Rightarrow X \rightarrow YZ$  ;
- проективность  $X \rightarrow YZ \Rightarrow X \rightarrow Y$  ;
- транзитивность  $(X \rightarrow Y) \text{ and } (Y \rightarrow Z) \Rightarrow X \rightarrow Z$  ;
- псевдотранзитивность  $(X \rightarrow Y) \text{ and } (YZ \rightarrow W) \Rightarrow XZ \rightarrow W$  .

Два множества функциональных зависимостей  $S_1$  и  $S_2$  эквивалентны тогда и только тогда, когда они являются покрытиями друг друга, то есть  $S_1^+ = S_2^+$ .

Каждое множество функциональных зависимостей эквивалентно, по

крайней мере, одному неприводимому множеству. Множество функциональных зависимостей является неприводимым, если:

- каждая функциональная зависимость этого множества имеет одноэлементную правую часть;
- ни одна функциональная зависимость множества не может быть устранена без изменения замыкания этого множества;
- ни один атрибут не может быть устранен из левой части любой функциональной зависимости без изменения замыкания множества.

Если  $I$  является неприводимым множеством, эквивалентным множеству  $S$ , то проверка выполнения функциональных зависимостей из множества  $I$  автоматически проверяет выполнение всех функциональных зависимостей множества  $S$ .

#### 4.3.3.5.3. Нормальные формы и схемы выполнения нормализации

Нормализация – это формальный метод анализа отношений на основе их первичного ключа и существующих функциональных зависимостей.

В процессе нормализации реляционных отношений применяются концепции нормальных форм. Говорят, что отношение находится в определенной нормальной форме, если оно удовлетворяет правилам этой нормальной формы. В настоящее время используется шесть нормальных форм, которые зависят друг от друга путем усложнения (вложенности) набора правил:

$$1NF \rightarrow 2NF \rightarrow 3NF \rightarrow НФБК \rightarrow 4NF \rightarrow 5NF.$$

Каждая нормальная форма, таким образом, удовлетворяет всем предыдущим нормальным формам. Более высокая нормальная форма приводит к более строгому формату отношения (меньшее число аномалий обновления).

БД можно построить и на отношениях находящихся в первой нормальной форме, но такая БД будет сильно подвержена аномалиям и избыточности данных. На практике желательно использовать как минимум 3NF, чтобы

устранить большинство аномалий обновления. Следует отметить, что процесс нормализации обратим (денормализация), то есть всегда можно использовать его результат для обратного преобразования, т.к. в процессе нормализации не утрачиваются первоначальные функциональные зависимости.

Определения нормальных форм:

1) 1НФ. Отношение находится в 1НФ тогда и только тогда, когда в любом допустимом значении этого отношения каждый кортеж содержит только одно значение для каждого из атрибутов, т.е. это значение не имеет внутренней структуры (множество, таблица и т.п.). отношения в 1НФ имеют большое количество аномалий обновления, для поддержания целостности БД требуется разработка сложных триггеров.

2) 2НФ. Отношение находится в 2НФ тогда и только тогда, когда оно находится в 1НФ, и каждый атрибут отношения не входящий в состав первичного ключа характеризуется полной функциональной зависимостью от этого первичного ключа. Полной функциональной зависимостью называется такая зависимость  $A \rightarrow B$ , когда  $B$  функционально зависит от  $A$  и не зависит ни от какого подмножества  $A$  (т.е. удаление какого-либо атрибута из  $A$  приведет к утрате этой функциональной зависимости). 2НФ устраняет в отношении частичные функциональные зависимости неключевых атрибутов от первичного ключа, которые выносятся в отдельное отношение вместе с копиями своих детерминантов.

3) 3НФ. Отношение находится в 3НФ тогда и только тогда, когда оно находится в 2НФ и не имеет не входящих в первичный ключ атрибутов, которые находились бы в транзитивной функциональной зависимости от этого первичного ключа. Транзитивной функциональной зависимостью называется зависимость  $A \rightarrow C$ , если существуют зависимости  $A \rightarrow B$  и  $B \rightarrow C$  (говорят что атрибут  $C$  транзитивно зависит от  $A$  через атрибут  $B$ ), при условии, что атрибут  $A$  функционально не зависит ни от атрибута  $B$ , ни от атрибута  $C$ . 3НФ устраняет в отношении транзитивные функциональные зависимости неключевых атрибутов от первичного ключа, которые выносятся в отдельное



отношение вместе с копиями своих детерминантов. В 3НФ устранено большинство аномалий от первичного ключа, но отношение в этой форме имеет аномалии в случае наличия более чем одного потенциального ключа.

4) НФБК (нормальная форма Бойса-Кодда). Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом. Нарушения требований НФБК случаются, если в отношении есть два и более составных потенциальных ключа и эти ключи перекрываются (совместно используют хотя бы один общий атрибут). Для отношения с единственным потенциальным ключом, НФБК и 3НФ эквивалентны. В НФБК устраняются аномалии связанные с функциональными зависимостями не от потенциальных ключей отношения.

5) 4НФ. Отношение находится в 4НФ тогда и только тогда, если все его многозначные зависимости, которым оно удовлетворяет, являются функциональными зависимостями от потенциальных ключей этого отношения. Многозначной зависимостью называется такая зависимость между атрибутами А и В некоторого отношения, при которой для каждого значения атрибута А существуют соответствующие наборы (множество) значений атрибута В. В 4НФ устраняются многозначные зависимости, которые не являются функциональными зависимостями.

6) 5НФ. Отношение находится в 5НФ тогда и только тогда, когда каждая нетривиальная зависимость соединения в отношении подразумевается его потенциальными ключами. Зависимостью соединения называется такая зависимость, при которой декомпозиция (разделение) отношения может сопровождаться генерацией ложных строк при выполнении обратного соединения декомпозированных отношений путем выполнения операции естественного соединения. В 5НФ устраняются зависимости соединения, которые не обусловлены потенциальными ключами.

Способы выполнения нормализации можно разделить на:

1) Синтез – получение БД в 3НФ путем компоновки всех заведомо известных функциональных зависимостей с использованием нахождения

неприводимого минимального покрытия  $I$  множества функциональных зависимостей  $S$ . Метод синтеза не требует построения начальной логической схемы БД, однако сложен для выполнения, т.к. число функциональных зависимостей всей БД может быть большим (сложность формирования  $I$ ). Достоинства метода синтеза – оптимальное число отношений БД, все отношения сразу находятся в ЗНФ.

2) Декомпозиция – формирование отношений БД путем разделения их на более мелкие, если эти отношения не выполняют правила необходимой нормальной формы. Процесс декомпозиции имеет два свойства:

- соединение без потерь – восстановление любого кортежа исходного отношения, используя соединение кортежей отношений, полученных в результате декомпозиции;
- сохранение зависимостей – функциональные зависимости при декомпозиции сохраняются.

Для выполнения процесса декомпозиции в начале необходимо построение исходной концептуальной модели БД (например, ER-диаграммы), которую преобразуют в начальные ненормализованные отношения. К недостаткам нормализации путем декомпозиции относят:

- временная сложность – неполиномиальна и определяется полным перебором всех порождаемых отношений (это число заранее не известно);
- число полученных отношений может быть больше оптимального для ЗНФ;
- возможны потери либо порождение новых функциональных зависимостей (характерно для более высоких нормальных форм);

Достоинства метода декомпозиции:

- разделение задачи на подзадачи, что позволяет выполнять задачу параллельно и с меньшей нагрузкой (и соответственно с меньшим числом ошибок);
- получение отношений в любой нормальной форме в любых сочетаниях.

Пример применения декомпозиции. Пусть дано отношение

$F\{S\#,City,Status,P\#,Qty\}$

и известны его функциональные зависимости:

- $S\# \rightarrow City$ ,
- $S\# \rightarrow Status$ ,
- $City \rightarrow Status$ ,
- $S\#,P\# \rightarrow Qty$ .

Первичный ключ отношения –  $S\#, P\#$ . Необходимо получить отношения в 3НФ.

Выполнение процесса по шагам:

1) 1НФ. Отношение  $F$  находится в 1НФ.

2) 2НФ. Атрибуты  $City$  и  $Status$  в отношении  $F$  частично зависят от первичного ключа, поэтому выносятся из отношения  $F$  в новое отношение  $SF$ . В  $SF$  для сохранения функциональных зависимостей копируется детерминант  $S\#$ , который становится первичным ключом отношения  $SF$ . В результате преобразования в 2НФ получаем:

$F\{S\#,P\#,Qty\}$ , первичный ключ –  $S\#, P\#$ ;

$SF\{S\#,City,Status\}$ , первичный ключ –  $S\#$ .

3) 3НФ. В отношении  $F$  нет транзитивных зависимостей, и значит, оно находится в 3НФ. В отношении  $SF$  наблюдается транзитивная зависимость  $S\# \rightarrow Status$  через атрибут  $City$ . Транзитивный атрибут  $Status$  выносится из отношения  $SF$  в новое отношение  $SSF$ . В  $SSF$  для сохранения функциональных зависимостей копируется детерминант  $City$ , который становится первичным ключом отношения  $SSF$ . В результате преобразования в 3НФ получаем:

$F\{S\#,P\#,Qty\}$ , первичный ключ –  $S\#, P\#$ ;

$SF\{S\#,City\}$ , первичный ключ –  $S\#$ ;

$SSF\{City,Status\}$ , первичный ключ –  $City$ .

Полученные отношения  $F$ ,  $SF$  и  $SSF$  – результат выполнения декомпозиции исходного отношения в 3НФ.

Суть процесса нормализации:

1) В нормализованных отношениях не разрешаются никакие функциональные зависимости, кроме функциональных зависимостей вида  $K \rightarrow A$ , где  $K$  – потенциальный ключ отношения  $R$ , а  $A$  – неключевой атрибут.

2) Если же отношение  $R$  имеет функциональные зависимости  $B \rightarrow A$ , где  $B$  не является потенциальным ключом, то в отношении  $R$  будет наблюдаться избыточность данных.

Выполнение нормализации важно не только при первичном проектировании реляционной БД, но также и при корректировке модели данных в процессе эксплуатации БД для учета новых функциональных зависимостей и устранения внесенных аномалий.

#### 4.3.3.6. Недостатки и пути развития реляционной модели

##### 4.3.3.6.1. Недостатки традиционных реляционных систем

Недостатки реляционных систем можно увидеть, проанализировав программные приложения, связанные с базами данных.

Традиционными для уверенного использования реляционных СУБД являются такие бизнес-приложения, как:

- обработка заказов;
- учет складских запасов;
- банковское дело;
- заказ билетов на транспорт (например, поезда и самолеты).

Однако существует ряд современных приложений, в которых реляционные системы продемонстрировали свою неадекватность:

1) Автоматизированное проектирование (CAD) – компьютерные системы проектирования сложных проектов – зданий, самолетов, микросхем и т.п. Такие

проекты имеют следующие особенности:

- большое количество разных типов при небольшом количестве экземпляров объектов, что формирует большое количество отношений и сложно для обработки в РСУБД;
- сложные проекты насчитывают миллионы элементов и связаны с другими проектами;
- проект не является статичным, любые изменения должны своевременно отражаться во всех представлениях проекта – перепроектирование БД на лету часто неприемлемо для РСУБД;
- обновление проекта в небольшой части затрагивает практически весь проект;
- для одного проекта возможно множество альтернативных решений, требуется вести их учет, сравнение и внесение изменений.
- в проекте могут участвовать одновременно сотни сотрудников, работающих над разными частями и версиями – проблема непротиворечивости и скоординированности действий.

2) Автоматизированное производство (САМ) – например, производство автомобилей, химический синтез – проблемы обработки данных в реальном времени, большой объем статистики, много операторов.

3) Автоматизированная разработка программного обеспечения (CASE) – подобно САД-проектам – коллективная разработка, много версий и т.п.

4) Офисные информационные системы, мультимедиа системы, цифровое издательское дело – большой набор разнообразных данных произвольных форматов (например, текстовых, видео, аудио, ...) и необходимость осмысленной навигации по данным, нестандартные операции с данными.

Несмотря на многие сильные стороны реляционной модели, такие как:

- теоретическая обоснованность и опора на математику,
- простота,
- пригодность для систем интерактивной обработки транзакций,
- обеспечение независимости от данных,

- стандарт по обработке данных SQL, РСУБД имеют и определенные недостатки:
- слабое представление сущностей реального мира – процесс нормализации приводит к созданию фрагментов объектов реального мира в БД, в процессе обработки данных требуется большое число соединений;
- семантическая перегрузка – все виды семантических данных представляются в РСУБД только одной структурой – отношением, такое представление в операциях частично теряет смысл данных;
- слабая поддержка ограничений целостности и корпоративных ограничений – современные РСУБД не поддерживают все возможности реляционной модели данных (например, домены), из-за чего многие правила целостности приходится встраивать в приложения, что небезопасно; в реляционной модели вообще не предусмотрены корпоративные правила целостности – только SQL или приложения;
- однородная структура данных – с одной стороны таблица представлена очень просто, что удобно для обработки (набор атомарных значений ячеек), с другой стороны – нет возможности обрабатывать более сложные структуры данных – их надо хранить неестественным образом; для сложных BLOB-объектов есть только общие операции - нет доступа к элементам объекта;
- ограниченный набор операций – нет определения новых операций с данными, например, в GIS-системах требуется обрабатывать векторные и растровые понятия;
- трудности организации рекурсивных запросов – большая вложенность запросов на SQL2 не может быть реализована без дополнительных операторов или языка программирования высокого уровня;
- проблема рассогласования – преобразования данных между разными языками (например, SQL->C->SQL) приводит к большим затратам производительности (до 30%) и ресурсов;
- жесткость структуры БД и трудности ее преобразования во время

эксплуатации;

- слабый навигационный доступ – противоречит принципам реляционных операций.

#### 4.3.3.6.2. Объектно-реляционные СУБД

Разработка СУБД с использованием расширенной реляционной модели данных или объектно-реляционной СУБД (ОРСУБД) относится к разработке СУБД третьего поколения.

Разработка ОРСУБД (старое название - расширенная реляционная СУБД) является ответом на развитие нереляционных СУБД третьего поколения, которые захватывают новые области применения (например, WWW). Другая сторона разработки ОРСУБД – множество компаний уже инвестировали средства в РСУБД, полная замена которых на нереляционные СУБД практически невозможна без значительных затрат – здесь ОРСУБД планируется как перспективная замена РСУБД, поэтому в этом направлении активно работают такие разработчики РСУБД, как Oracle и IBM.

Наиболее очевидный способ преодоления недостатков РСУБД - это внесение в нее дополнительных функций:

- расширяемая пользователем система типов,
- инкапсуляция,
- наследование,
- полиморфизм,
- динамическое связывание методов,
- использование составных объектов (в том числе и не в 1НФ),
- поддержка идентичности объектов.

Не существует единого стандарта расширенной реляционной модели, в каждой ОРСУБД формируется свой набор функциональных возможностей. Общие принципы построения ОРСУБД базируются на совмещении подходов РСУБД (использование базовых реляционных таблиц и расширенного

реляционного языка запросов (SQL3)) и объектно-ориентированных систем (определение понятия объекта и методов работы с ним). Развитие исследовательских проектов ОРСУБД ведется примерно с середины 1980 гг., а коммерческие ОРСУБД декларированы серединой 1990 гг. Примерами ОРСУБД можно назвать Postgres, IUS (Informix Universal Server), Oracle 8 и Oracle 9i, DB2 Universal Database.

Основные преимущества ОРСУБД:

- устранение недостатков широко распространенных РСУБД – большой рынок приложений;
- повторное и совместное использование компонентов – расширение сервера СУБД специальными функциями, которые выполняются централизованно и не входят в состав кода приложений (например, векторные вычисления);
- сохранение ранее наработанных знаний и опыта, связанных с разработкой реляционных приложений – эволюционный (а не революционный) путь развития требует меньших финансовых затрат;

Недостатки ОРСУБД:

- повышенные расходы на эксплуатацию сложных систем;
- утрата простоты и ясности реляционной модели;
- ожидание низкой производительности от расширений (особенно в традиционных реляционных приложениях);
- неполное соответствие объектно-ориентированному подходу – подмена терминов, т.к. есть большой семантический разрыв между реляционными и объектно-ориентированными системами (понятие объектов гораздо шире понятия данных);
- значительное усложнение языка обработки данных, для поддержки SQL2.

Отделом CADF (Committee for Advanced DBMS Function) сформирован манифест баз данных третьего поколения, которому должна следовать любая СУБД третьего поколения (основные положения):

- богатая система типов;



- желательна поддержка механизма наследования;
- поддержка функций, процедур и методов, а также механизм инкапсуляции;
- использование системных уникальных идентификаторов только при отсутствии первичных ключей, определенных пользователем;
- наличие триггеров и ограничений;
- все программные способы доступа к базе данных осуществляются с помощью непроцедурного языка высокого уровня
- должны быть как минимум два способа задания коллекций: на основе перечисления членов коллекции и с использованием языка запросов для определения членства в коллекции;
- наличие обновляемых представлений;
- СУБД третьего поколения должны быть доступны из многих языков высокого уровня;
- язык SQL остается стандартом для работы с данными;
- запросы и ответы на них должны составлять самый нижний уровень взаимодействия сервера и клиента.

С другой стороны, существует ряд сторонников традиционной реляционной модели данных (Дейт), и по их мнению объектно-ориентированные и реляционные системы ортогональны друг другу и «реляционная модель не нуждается ни в каких расширениях, исправлениях, категоризациях и, более всего, в извращениях». Развитие реляционных систем этой группой видится в полной и правильной реализации реляционной модели в РСУБД и переработка (замена) языка SQL (вместо него предложен язык D).

#### 4.3.3.6.3. Нереляционные СУБД третьего поколения

Основной конкурент РСУБД среди нереляционных СУБД представлен объектно-ориентированными СУБД (ООСУБД). ООСУБД применяются с

конца 1980-х для обеспечения управления базами данных приложениями, построенными в соответствии с концепцией объектно-ориентированного программирования.

В манифесте ООСУБД (1989 г.) предлагаются правила определения ООСУБД:

- поддержка составных объектов;
- поддержка идентичности объектов – наличие у объекта уникального идентификатора, независящего от значений атрибутов;
- поддержка инкапсуляции – программисты имеют права доступа только к методам объектов;
- поддержка типов или классов;
- поддержка наследования типов или классов от их предков;
- поддержка динамического связывания – перегрузка и переопределение методов во время выполнения программы;
- язык DML должен обладать вычислительной полнотой – фактически это должен быть язык программирования общего назначения;
- набор типов данных должен быть расширяемым – должны быть средства создания новых типов данных пользователя;
- поддержка перманентности данных – данные должны сохраниться после завершения работы приложения без явных указаний пользователя;
- поддержка очень больших баз данных – должны быть средства, как и у РСУБД, для управления буферизацией, индексированием и т.п.
- поддержка параллельной работы многих пользователей;
- обеспечение восстановления после сбоев аппаратного и программного обеспечения;
- предоставление простых способов создания запросов к данным – высокоуровневое, эффективное и независимое от приложений средство создания запросов, но не обязательно декларативный язык.

К преимуществам ООСУБД можно отнести:

- улучшенные возможности моделирования – объектно-ориентированная модель более точно представляет реальный мир (понятие объектов, который инкапсулирует состояние и поведение, связи с другими объектами хранятся в самом объекте (в том числе и связи «многие-ко-многим»));
- расширяемость – создание новых абстрактных типов данных, работает наследование и переопределение методов, легко выполняется повторное использование классов;
- устранение проблемы несоответствия – нет проблемы преобразования типов как в РСУБД (SQL->C->SQL), DML обладает вычислительной полнотой;
- более выразительный язык запросов – использование навигационного способа обработки данных (хорошо подходит для выполнения рекурсивных запросов, ветвлений и т.п.), есть версии декларативных языков построенных на объектно-ориентированной форме SQL – OQL (для конечных пользователей);
- поддержка эволюции схемы – использования наследования и генерализации для более понятной и семантической точной схемы данных;
- поддержка долговременных транзакций – обычно это является проблемой больших РСУБД;
- применимость для сложных специализированных приложений баз данных – проблемные области для применения традиционных РСУБД;
- повышенная производительность – не для традиционных приложений РСУБД, преимущество может достигаться в том, что для ООСУБД отсутствует уровень преобразований между языком СУБД и языком приложения, а также отображение объектов в память с вторичного выполняется более эффективно (нет лишних преобразований и контроля типов).

Недостатками ООСУБД являются:

- отсутствие универсальной модели данных – нет стандарта на объектно-ориентированную модель данных и теоретического обоснования этой модели;
- недостаточность опыта эксплуатации – использование ООСУБД пока ограничено, более подходит для программистов, чем для конечных пользователей;
- отсутствие стандартов – нет фактических стандартов на язык запросов (OQL), предложен фактический, но не международный стандарт на объектно-ориентированную БД – ODMG 2.0(1995 г.);
- влияние оптимизации запросов на инкапсуляцию – доступ к объектам напрямую без защиты;
- влияние блокировки на уровне объекта на производительность – может вызывать проблемы в отношении иерархии наследования;
- сложность – ООСУБД сложнее РСУБД, сложная система – сложные продукты, поддержка одноуровневой модели (диск-память) хранения данных тоже сложнее, чем двухуровневая модель (диск-преобразование-память) в РСУБД;
- отсутствие поддержки представлений – существенное отличие от реляционной модели в плане администрирования;
- недостаточность средств обеспечения безопасности – не реализованы адекватные механизмы обеспечения безопасности (только высокий уровень блокировок, плохое управление правами доступа).

Примерами ООСУБД можно назвать Objectivity/DB 2.1, GemStone 5.0, ONTOS, OpenODB, ObjectStore, Versant и UniSQL.

Направление ООСУБД - динамически развивающееся направление на рынке СУБД.

#### 4.4. Физические модели данных

Физические модели данных в данном курсе рассмотрены со стороны структур и методов хранения данных во вторичной памяти компьютера, т.к. эти вопросы оказывают существенное влияние на физическое проектирование БД.

#### 4.4.1. Основные понятия физического хранения данных

Под физическим хранением (структурой хранения) данных будем понимать размещение данных на вторичных устройствах хранения (например, на основных сегодня вторичных устройствах хранения - накопителях на жестких магнитных дисках типа «винчестер»)

Не существует идеальной структуры хранения, которая была бы оптимальной для всех задач, поэтому СУБД обычно поддерживает несколько разных структур хранения данных. Основным критерий оптимальности структуры хранения – число операций ввода-вывода данных.

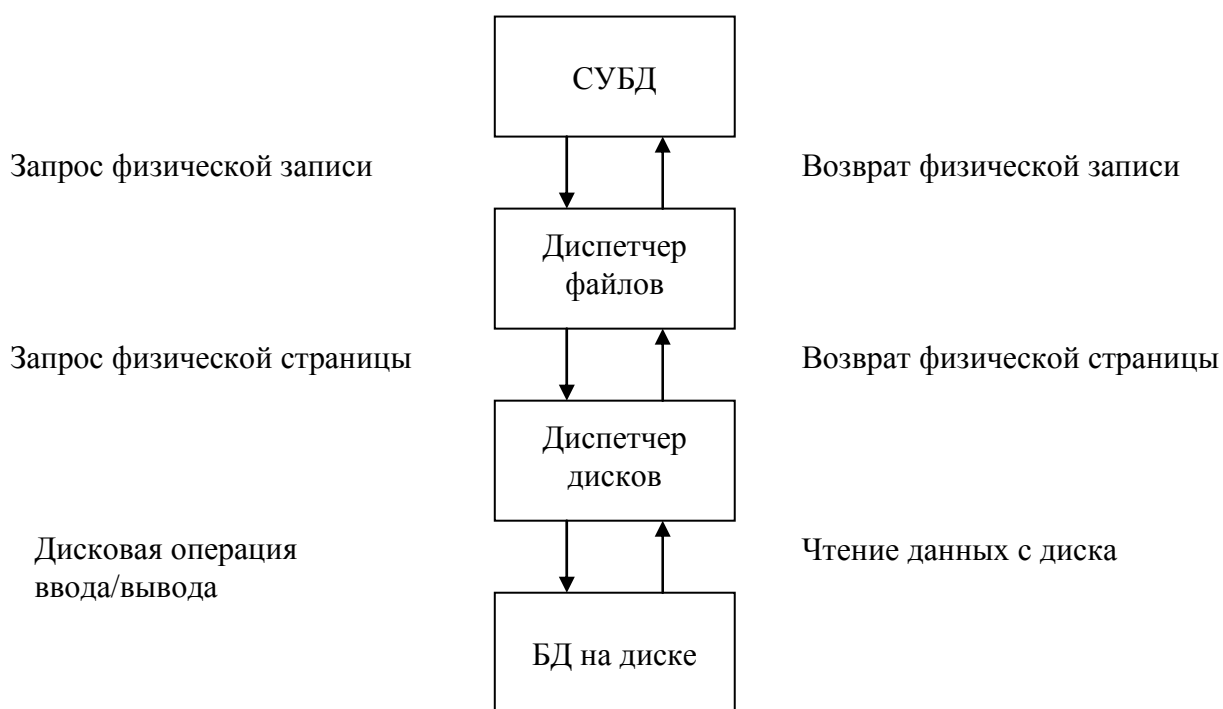


Рисунок 19. Компоненты доступа к данным БД.

Для организации доступа к данным в СУБД используются следующие

компоненты:

1) Диспетчер дисков – компонент операционной системы, с помощью которого выполняются определение физического адреса данных и все дисковые операции ввода-вывода данных. Диспетчер дисков рассматривает диск как набор страниц фиксированного размера. Каждая страница имеет свой уникальный идентификатор – идентификационный номер страницы. Каждый набор также имеет свой уникальный идентификационный номер набора, а также наборы не имеют общих страниц. Существует специальный набор – набор пустых страниц, в который входят все свободные от данных страницы.

Основные операции диспетчера дисков:

- извлечь страницу  $s$  из набора страниц  $f$ ;
- заменить страницу  $s$  из набора страниц  $f$ ;
- добавить новую страницу в набор страниц  $f$  (данная страница извлекается из набора пустых страниц, как правило добавление страниц идет экстендами – блоками физически связанных страниц (например, страница = 8 Кб, экстенд = 8 страниц));
- удалить страницу  $s$  из набора страниц  $f$  (данная страница возвращается в набор пустых страниц);

2) Диспетчер файлов – компонент, с помощью которого СУБД получает возможность выполнять операции над файлами. Файл – хранимый на диске набор однотипных физических записей, которые в свою очередь состоят из полей. БД организуется на диске в виде совокупности файлов. Каждый файл имеет имя (или уникальный идентификатор). Каждая запись файла имеет уникальный идентификационный номер записи. Обычно размер записи гораздо меньше размера страницы, но не исключается и обратное – одна запись занимает несколько страниц диска. Основные операции диспетчера файлов:

- извлечь запись  $r$  из файла  $f$ ;
- изменить запись  $r$  в файле  $f$ ;
- добавить запись  $r$  в файл  $f$ ;
- удалить запись  $r$  из файла  $f$ ;

- создать новый файл  $f$ ;
- удалить файл  $f$ ;

Процесс доступа к данным в этой схеме следующий:

1) СУБД определяет искомую логическую запись, отображает логическую запись на физическую запись и запрашивает для ее извлечения диспетчер файлов.

2) Диспетчер файлов по описанию физической записи (имя файла, номер записи) определяет страницу, на которой запись расположена. Для извлечения этой страницы запрашивается диспетчер дисков.

3) Диспетчер дисков определяет физическое положение страницы на диске и выполняет операции ввода-вывода данных. Для повышения производительности работы с диском выполняется кэширование запрашиваемых страниц данных в специальных буферах памяти, страница может храниться в этом буфере пока не потребуется место для обработки другой страницы данных. Диспетчер дисков уведомляет о проведенной операции диспетчер файлов.

4) Диспетчер файлов извлекает искомую физическую запись из страницы данных, переданной ему диспетчером дисков, и уведомляет об операции СУБД.

5) СУБД получает искомую запись от диспетчера файлов.

Порядок хранения записей и доступ к ним зависит от организации файла – физического распределения данных файла по записям и страницам на диске.

Основные типы организации файлов:

- последовательные неупорядоченные и упорядоченные файлы;
- хешированные файлы.

Кроме указанных типов организации файлов также используются специальные структуры хранения в виде совокупности файлов – индексы.

Для обращения к данным (определения местоположения записей) используется соответствующий метод доступа.

#### 4.4.2. Последовательные неупорядоченные и упорядоченные файлы

Неупорядоченные последовательные файлы (heap, pipe) – файлы, в которых записи размещаются в том порядке, в котором они в этот файл добавляются. Новая запись всегда помещается на последнюю страницу файла. Если на данной странице нет места для размещения записи, то к файлу добавляется новая страница.

Преимущество неупорядоченных последовательных файлов - операции вставки записей выполняются очень эффективно, поэтому такие структуры идеально подходят для пакетной загрузки данных.

Недостатками данных структур являются:

- медленный поиск записей, т.к. поиск выполняется последовательно путем перебора записей файла от первой до последней, пока не встретиться искомая запись;
- медленное удаление записей, т.к. для поиска удаляемой записи также используется последовательный поиск;
- при удалении записей возникают «дырки» - пометки об удаленных записях (с одной стороны это не требует немедленного физического сжатия файла, с другой стороны – раздувается размер файла и соответственно увеличивается время поиска). Физическое сжатие файла для избавления от «дырок» («сборка мусора») требует определенных временных затрат.

Упорядоченные последовательные файлы – файлы, в которых записи располагаются упорядоченно (т.е. отсортированы по некоторому ключу). В качестве ключа обычно выступает первичный ключ в структурах БД.

Главное преимущество подобной организации файлов – возможность выполнения быстрого поиска записей по значению ключевого поля. Быстрый поиск выполняется методом бинарного поиска (или делением пополам):

- 1) извлечение средней страницы в заданной зоне поиска (для первого вхождения рассматривается весь файл целиком);



2) выполнения сравнения значения ключевого поля первой записи извлеченной страницы с искомым значением ключа:

- если значения совпадают, то искомая запись найдена, поиск завершен;
- если осталась только одна страница в зоне поиска, то поиск проводится среди записей этой страницы (методом перебора), поиск завершается либо успешно, либо неудачно (нет совпадений значений ключа);
- если извлеченное значение больше искомого, то поиск надо продолжить в зоне поиска, начиная с извлеченной страницы, которая будет рассматриваться как новая зона поиска, переход к шагу 1;
- если извлеченное значение меньше искомого, то поиск надо продолжить в зоне поиска перед извлеченной страницей, которая будет рассматриваться как новая зона поиска, переход к шагу 1;

Также достоинством упорядоченных файлов является быстрое удаление данных (влияние быстрого поиска, данные физически не удаляются, а только помечаются, удаленные записи позволяют быстрее проводить вставку данных, т.к. потребуется сдвиг записей для освобождения места только до места удаленной записи, а не до конца файла).

При последовательном доступе к записям упорядоченного файла, получаем отсортированный по ключу список, что выгодно использовать для формирования отчетов.

Недостатком упорядоченных последовательных файлов: усложнение операций вставки записей – необходимо соблюдать упорядоченность записей, поэтому на поиск места расположения записи и перемещение записей для освобождения свободного места приходится тратить время; для повышения производительности в этом случае могут использоваться файлы переполнения – последовательные неупорядоченные файлы, которые выступают в роли буфера, куда помещаются данные пока выполняется вставка записей, записи из буфера помещаются в упорядоченный файл либо когда есть свободное время в системе, либо перед выполнением чтения записей из упорядоченного файла.

Если в объекте данных БД определен первичный индекс, то для его хранения оптимально использовать упорядоченный последовательный файл.

#### 4.4.3. Хешированные файлы

Хеширование (hashing - перемешивание) – это технология быстрого прямого доступа к физической записи на основе заданного значения поля перемешивания (хеш-ключ).

Различают статическое и динамическое хеширование.

Для статического хеширования, размер хеш-файла задается один раз при его создании и больше не изменяется (фиксируется пространство хеш-адресов).

Для вычисления адреса страницы, на которой должна храниться запись, используется хеш-функция, аргументами которой являются значения одного или нескольких полей записи (хеш-ключ). Хеш-функция выбирается таким образом, чтобы записи внутри файла были размещены максимально равномерно, но в статических хеш-файлах все равно остается чистое место (чистые страницы разбросанные по всему файлу) под неиспользованные значения хеш-ключа.

Методы хеширования основываются на:

- свертке – арифметические действия над частями поля перемешивания;
- использовании остатка от деления – хеш-ключ делится на некоторой простое число;

Преимущество хеш-файлов – быстрый доступ к записям по известному значению хеш-ключа, который осуществляется в два этапа:

- вычисление адреса страницы с помощью хеш-функции и известного хеш-ключа;
- извлечение данной страницы и последовательный поиск в ней заданной записи.

К недостаткам статических хеш-файлов относят следующие:

- физическая последовательность записей в файле почти всегда отличается от логической последовательности, последовательный перебор записей фактически не имеет смысла;
- между записями в файле могут быть чистые промежутки – зависит от выбора хеш-функции;
- возникновение коллизий – ситуаций, когда записи имеют одинаковые хеш-адреса и не вмещаются на адресуемую страницу.

Для разрешения коллизий используют следующие методы:

1) Открытая адресация – после нахождения хеш-адреса, запись вставляется на ближайшее свободное место, которое находится последовательным перебором начиная с данной страницы, поиск выполняется аналогично. Этот метод используют при очень редких коллизиях.

2) Несвязанная область переполнения – все записи непомещающиеся на страницу размещаются в специальной области (например, в последовательном неупорядоченном файле), поиск проводится по странице, а в случае неудачи продолжается в области переполнения. Этот метод также используется при очень редких коллизиях.

3) Связанная область переполнения – для каждой страницы с коллизиями, формируется специальная область (связный список страниц), с которой данная страница связана через указатель. Этот метод можно использовать при частых коллизиях.

4) Многократное хеширование – применение нескольких хеш-функций, если первая хеш-функция приводит к коллизии.

5) Смена хеш-функции – при этом придется сформировать новый хеш-файл и скопировать в него данные, используя новую хеш-функцию.

Динамическое хеширование позволяет динамическое изменение размера хеш-файла при добавлении в него записей. Кроме того, динамическое хеширование исключает коллизии. Основным принципом динамического хеширования заключается в рассмотрении хеш-адреса, как битовой последовательности и распределение записей на страницах на основе

прогрессивной оцифровки этой последовательности.

Один из типов динамического хеширования называется расширяемое хеширование. Основные принципы расширяемого хеширования:

1) В результате вычисления хеш-функции  $h$  для поля  $k$  записи  $r$  будет получен псевдоключ  $s$ . Псевдоключ используется не как адрес, а как косвенный указатель на место хранения записей.

2) Хеш-файл имеет каталог, имеющий глубину каталога  $d$  и содержащий  $2^d$  указателей на страницы данных.

3) Если старшие  $d$  бит псевдоключа рассматривать как двоичное число  $b$ , то  $i$ -ый указатель в каталоге ( $1 \leq i \leq 2^d$ ) будет относиться к странице, на которой хранятся записи с  $b = i - 1$ .

4) Каждая страница данных имеет заголовок с указанием локальной глубины  $p$  ( $p \leq d$ ) этой страницы (локальная глубина показывает, сколько указателей каталога могут на нее ссылаться -  $2^{d-p}$  указателей).

Чтобы найти искомую запись в динамическом хеш-файле, необходимо выполнить следующие действия:

- 1) найти значение псевдоключа  $s$ ;
- 2) по первым  $d$  битам псевдоключа  $s$  определить значение указателя каталога (первое обращение к диску – считывание каталога);
- 3) по этому указателю считать с диска страницу, содержащую исходную запись (второе обращение к диску).

Записи добавляются в хеш-файл следующим образом:

- 1) выполняется поиск страницы для вычисленного псевдоключа  $s$  вставляемой записи (аналогично процессу чтения записи);
- 2) если место на найденной странице для записи есть, то производится запись данных на указанную страницу;
- 3) если найденная страница заполнена полностью, то необходимо расширение хеш-файла:

- если локальная глубина страницы меньше локальной глубины каталога (рисунок 20,  $2 < 3$ ), то указанная страница делится на две части,

локальная глубина полученных страниц будет на 1 больше исходной ( $2 + 1 = 3$ ); на первой странице будут размещены все записи с  $b = 000\dots$ , а на второй странице – с  $b = 001\dots$ ; указатели каталога будут соответственным образом скорректированы;

- если локальная глубина страницы равна глубине каталога, то требуется перед разделением страницы выполнить дублирование каталога – эта операция увеличивает глубину каталога на 1, а каждый указатель каталога заменяется парой смежных указателей (например, указатель 000 разделится на 0000 и 0001), значения которых будут хранить ссылку на страницу данных, которая хранилась в старом указателе; дублирование каталога производится быстро и без обращения к страницам данных.

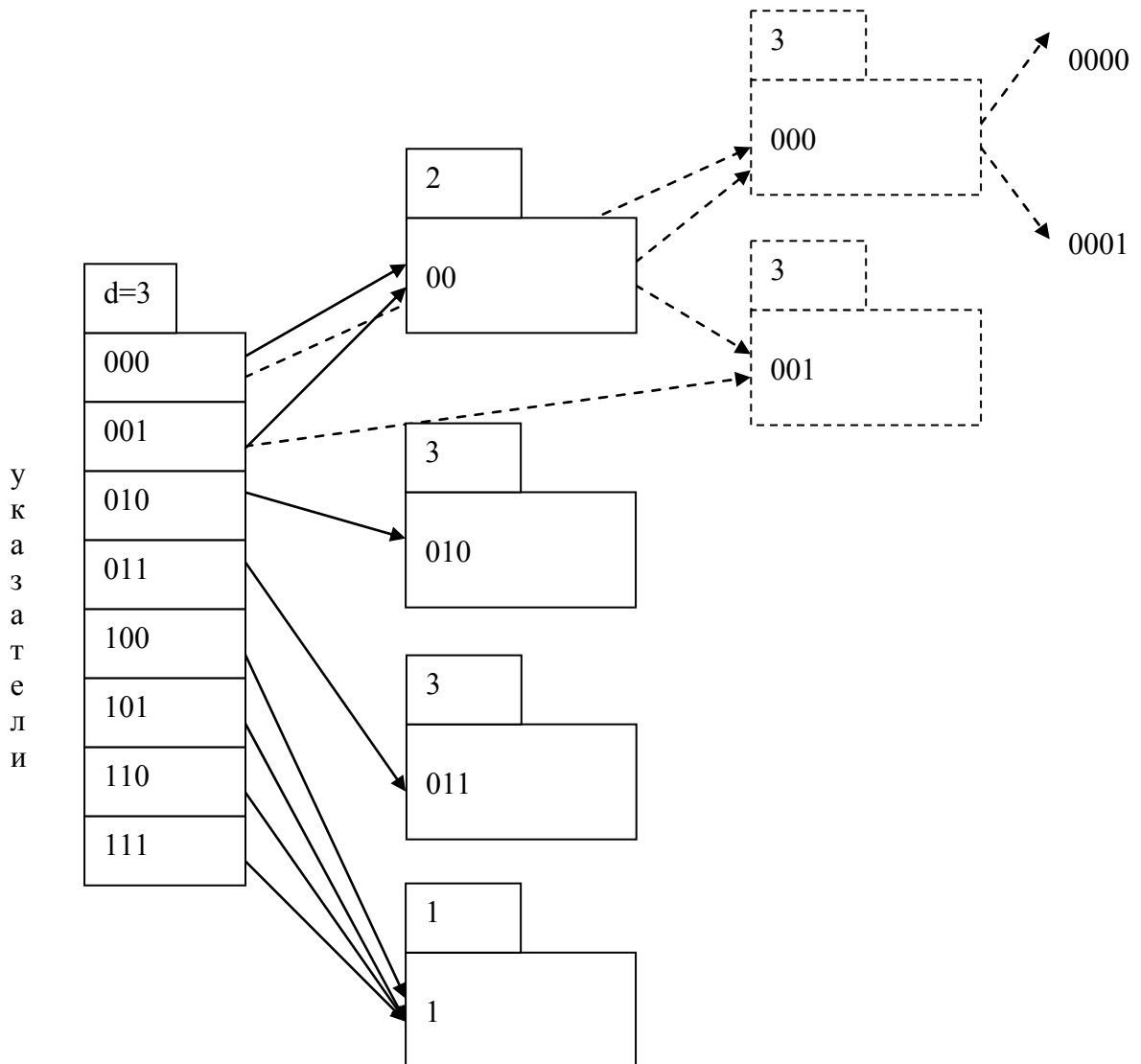


Рисунок 20. Расширяемое хеширование.

Удаление записей является обратной операцией к записи данных в хеш-файл и приводит к его уменьшению.

Главный недостаток любого варианта хеширования – поиск записей возможен только по хеш-ключу.

#### 4.4.4. Индексы

Индекс – структура данных определенного вида, которая предназначена для ускорения поиска записей файла данных.

В простейшем варианте, индекс представляет собой файл, записи которого содержат ключ (поле, содержащее одно или несколько атрибутов записи файла данных и предназначенное для осуществления поиска записей по этому критерию) и указатель (поле, содержащее адрес записи в файле данных). Поле адреса заполняется СУБД. Записи индекса упорядочиваются по ключевому полю. Файл данных, для которого существует индекс, называется индексированным, а поле индексированного файла, значения которого используется в индексе, называется индексным полем. Индекс можно создать как по одному полю, так и по нескольким полям, причем не обязательно относящимся к первичному ключу.

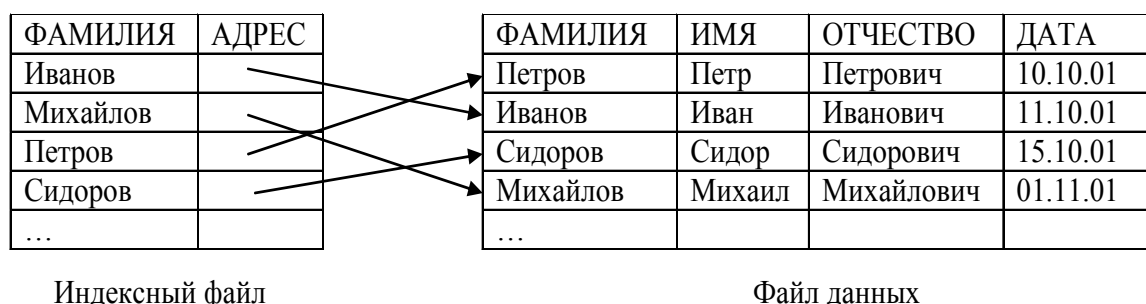


Рисунок 21. Структура простого индекса.

Основное преимущество использования индексов – значительное ускорение выборки данных. Вместо поиска записи в файле данных (обычно

методом последовательного перебора), который имеет большой объем, поиск проще проводить в индексе с использованием метода бинарного поиска. Тогда выборка одной записи данных с помощью индекса состоит из следующих шагов:

- быстрый поиск записи в индексе, которая содержит искомый ключ;
- выборка записи из файла данных по найденному адресу записи.

С использованием индекса возможно также выполнение последовательного доступа к данным, отсортированным по ключу индекса. Например, выбрать всех сотрудников, фамилии которых начинаются на букву «И», в этом случае движение по индексу (перебор соседних записей индекса, которые имеют значения в заданном интервале) позволяет обращаться к записям файла данных в указанном порядке.

Основной недостаток использования индексов – замедление обновления файла данных, т.к. при добавлении новой записи в файл данных требуется обновление индекса, а индекс представляет собой упорядоченный последовательный файл и обновляется медленно.

Индексы можно классифицировать следующим образом:

1) по виду используемых полей записи данных:

- первичный индекс – индекс, построенный на основе поля первичного ключа данных; такой индекс всегда один, значение ключа индекса уникально; введение первичного индекса позволяет хранить данные в неупорядоченном последовательном файле;
- вторичный индекс – индекс, построенный не по первичному ключу, позволяет ускорить операции выборки для запросов, выполняющих фильтрацию не по полям первичного ключа; вторичных индексов может быть несколько (однако следует помнить, что чем больше индексов у файла данных, тем медленнее вставка в него данных – вторичные индексы должны строиться только для самых критичных и часто выполняемых запросов); значения ключей в индексе могут повторяться.

2) по числу используемых полей записи данных:

- несоставной – ключ индекса состоит только из одного поля записи;
- составной – ключ индекса состоит из нескольких полей записи, при этом сортировка ключа индекса выполняется в следующем виде: основной порядок сортировки задает первое поле ключа, дополнительный порядок сортировки (сортировка в группе) задает второе поле ключа, и т.п.; составной индекс по полям записи  $(f_1, f_2, f_3, \dots, f_n)$  может использоваться для поиска по полю  $f_1$  либо по комбинациям полей  $(f_1, f_2)$ ,  $(f_1, f_2, f_3)$ ,  $(f_1, f_2, f_3, f_4)$ , ...,  $(f_1, f_2, f_3, \dots, f_{n-1})$ ,  $(f_1, f_2, f_3, \dots, f_n)$  – т.е. один составной индекс может обслуживать ряд запросов, но поля используемые при этом должны располагаться с начала ключа индекса (т.к. они зададут порядок сортировки записей индекса) и без разрывов; для  $N$  полей записи максимальное число индексов определяется как

$$C_N^n = \frac{N!}{n! * (N - n)!},$$

где  $n$  – целое число равное  $N/2$  (округление при перевод в целое число производится здесь в большую сторону).

3) по числу ссылок на данные:

- плотный – число индексных записей равно числу записей данных, одна индексная запись ссылается только на одну запись данных;
- неплотный – число индексных записей меньше числа записей данных, индекс указывает либо на первую запись в определенной группе, либо на страницу с определенной группой записей данных, записи данных при этом также должны быть упорядочены по некоторому полю. Например, если список сотрудников упорядочен по фамилии, то можно построить неплотный индекс, которых будет в качестве ключа содержать первую букву фамилии, и этот индекс будет ссылаться на записи данных следующим образом: «А» -> на первую фамилии в списке начинающуюся на «А», «Б» -> на первую фамилии в списке начинающуюся на «Б», и т.п. Поиск конкретного сотрудника по фамилии тогда можно осуществить следующим образом:



- а) найти букву, на которую начинается фамилия сотрудника;
- б) выполнить поиск фрагмента файла данных, отвечающего за размещение фамилий начинающихся на данную букву, с использованием неплотного индекса;
- в) выполнить поиск в этом фрагменте файла данных (либо последовательным перебором, либо используя упорядоченность по фамилии для бинарного поиска).

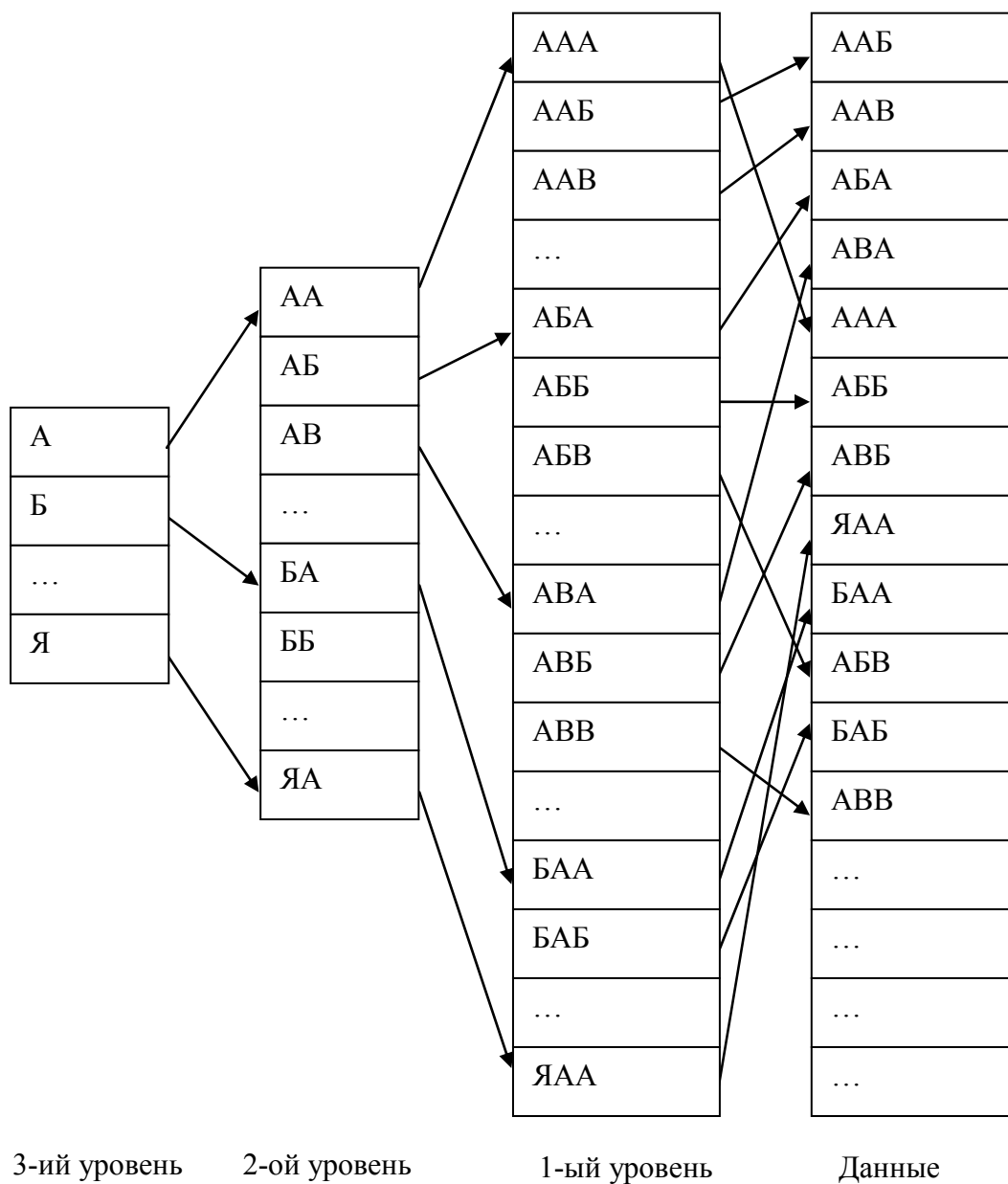


Рисунок 22. Многоуровневый индекс.

4) по числу уровней индекса;

- одноуровневый – индекс, который непосредственно ссылается на данные, а не на другие индексные структуры;
- многоуровневый – индекс, состоящий из нескольких индексных файлов, при этом только индекс первого уровня ссылается на реальные данные (обычно это плотный индекс), а индексы более высоких уровней ссылаются на предыдущие уровни (эти индексы обязательно неплотные). Структура многоуровневого индекса за счет увеличения объема данных позволяет сократить время поиска записей, т.к. данные уже разбиты на фрагменты, которые бы получались, например, в результате бинарного поиска. Многоуровневый индекс вводится при наличии в файле данных большого числа записей, обычно при условии, что бинарный поиск в одноуровневом плотном индексе проводится за значительное число шагов. Поиск данных в многоуровневом индексе начинается с самого верхнего уровня и продолжается пока не будет достигнута запись данных, поиск ссылки на следующий уровень на текущем уровне проводится методом двоичного поиска в определенном диапазоне. На практике, число уровней многоуровневого индекса обычно не превышает трех.

Особым типом многоуровневых индексов являются древовидные индексы. Рассмотрим одну из разновидностей древовидных индексов – В<sup>+</sup>-деревья.

Дерево, как структура индексов, состоит из иерархии узлов, в которой каждый узел за исключением корня, имеет родительский узел, а также этот узел может ссылаться на дочерние узлы. Узел, не имеющий дочерних узлов, называется листом дерева. Глубиной дерева называется максимальное количество уровней между корнем дерева и листьями дерева. Если глубина дерева одинакова для всех листьев, то такое дерево называется сбалансированным или В-деревом. Степень или порядок дерева – максимально допустимое количество дочерних узлов для каждого родительского узла (за

исключением листьев).

$B^+$ -дерево представляет собой сочетание  $B$ -дерева (сбалансированного неплотного древовидного индекса) и плотного индекса (листовой уровень, который дополнительно упорядочен). Правила определения  $B^+$ -дерева:

1) если корень не является листовым узлом, то он должен иметь, по крайней мере, два дочерних узла;

2) в дереве порядка  $n$  каждый узел (за исключением корня и листьев) должен иметь от  $n/2$  до  $n$  указателей и дочерних узлов, если число  $n/2$  не является целым, то оно округляется до ближайшего большего целого числа;

3) в дереве порядка  $n$  количество ключевых значений в узле должно находиться в пределах от  $(n-1)/2$  до  $(n-1)$ , если число  $(n-1)/2$  не является целым, то оно округляется до ближайшего большего целого числа;

4) количество ключевых значений в нелистовом узле на единицу меньше количества указателей;

5) дерево всегда должно быть сбалансированным;

6) листья дерева должны быть связаны в порядке возрастания ключевых значений.

На практике, каждый узел в дереве является страницей (4 Кб), порядок дерева равен 512, число ярусов (уровней) дерева равно трем, тогда:

1-ый уровень (корень дерева) имеет максимум 512 ссылок на дочерние узлы;

2-ой уровень имеет 262144 (512 узлов \* 512 ссылок) ссылок на дочерние узлы;

3-ий уровень имеет 133955584 (262144 узлов \* 511 ссылок) ссылок на страницы данных;

Доступ к одной записи данных в такой структуре осуществляется за четыре обращения к диску (1-ый уровень, 2-ой уровень, 3-ий уровень, непосредственные данные).

Для поддержки сбалансированности дерева применяется специальный алгоритм, который работает при вставке и удалении записей данных.

Рассмотрим этот алгоритм с точки зрения изменения индекса (добавления нового ключа) при вставке данных:

1) на самом низком уровне набора индексов нужно найти узел  $N$ , с которым будет связано вставляемое значение ключа  $V$ ; поиск узла  $N$  осуществляется также, как поиск записи по известному ключу – двигаясь от корня до листа по ссылкам, для которых выполняется условие «значение текущего ключа в текущем узле меньше или равно значению  $V$ »;

2) если узел  $N$  содержит свободное пространство для ключа  $V$ , то значение  $V$  вставляется в него (с учетом упорядоченности значений ключей) и процесс вставки завершается, в противном случае выполняется разделение узла  $N$  на два узла  $N1$  и  $N2$  (причем, все упорядоченное множество значений ( $2n+1$  значений) ключей узла  $N$  также разделяется на две части –  $n$  первых значений этой последовательности будут помещены в  $N1$ ,  $n$  последних – в  $N2$ , а среднее между ними значение  $W$  будет помещено в родительский элемент  $P$  на более высоком уровне (для листьев значение  $W$  будет сохранено в узле  $N1$ , т.к. листья это плотный индекс)); в дальнейшем, при выполнении поиска значения  $u$  и достижения  $P$ , поиск будет перенаправлен в сторону  $N1$ , если  $U \leq W$ , либо в сторону  $N2$ , если  $U > W$ ;

3) Далее процесс повторяется для вставки значения  $W$  в узел  $P$ ; процесс разделения элементов структуры может идти вплоть до корневого узла (с образованием нового иерархического уровня).

Удаление записи – процесс обратный вставке, может вести к удалению уровня дерева. Изменение данных обычно выполняется как удаление старого значения и вставка нового значения.

Формирование  $B^+$ -дерева рассмотрим на следующем примере:

Дано: требуется произвести вставку ключей 41, 37, 14, 3, 5, 20 (в указанном порядке) в дерево порядка 3 (узел состоит из трех указателей и двух ключей).

Выполнение по шагам:

1) вставка ключа 41 – формирование первого узла дерева (корня дерева, и

он же одновременно - лист);

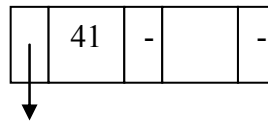


Рисунок 23. Вставка ключа 41.

2) вставка ключа 37 – значения в листе упорядочиваются по возрастанию значений ключей (плотный индекс), перед каждым ключом расположен указатель на страницу с данными;

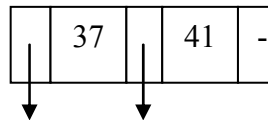


Рисунок 24. Вставка ключа 37.

3) вставка ключа 14 – корневой узел заполнен полностью – выполняется разделение узла и формируется новый корень;

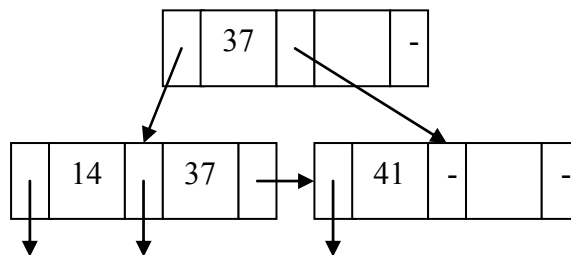


Рисунок 25. Вставка ключа 14.

4) вставка ключа 3 – листовой узел заполнен полностью – выполняется разделение узла;

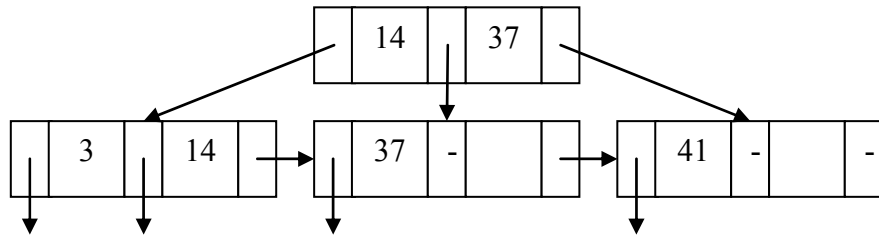


Рисунок 26. Вставка ключа 3.

5) вставка ключа 5 – листовой узел заполнен полностью – выполняется разделение узла и формирование нового родительского уровня, т.к. корневой уровень тоже заполнен;

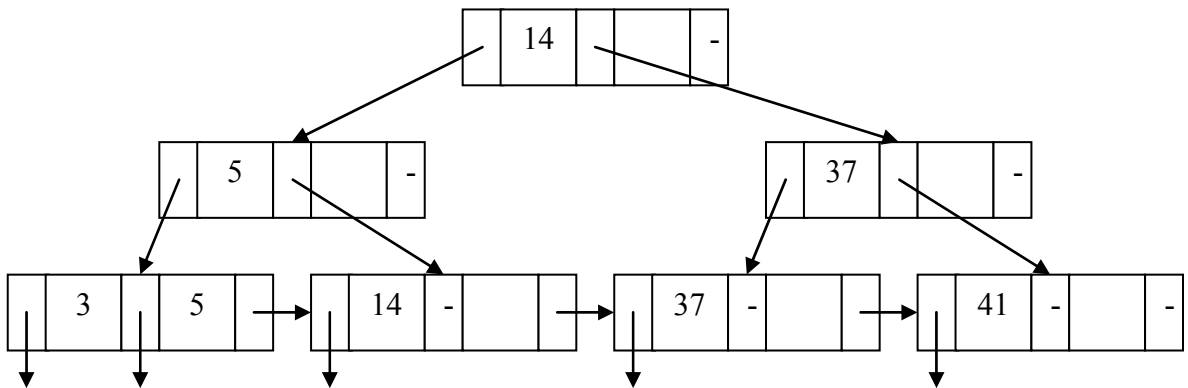


Рисунок 27. Вставка ключа 5.

6) вставка ключа 20 – поиск места дает свободное место - не требуется расширения;

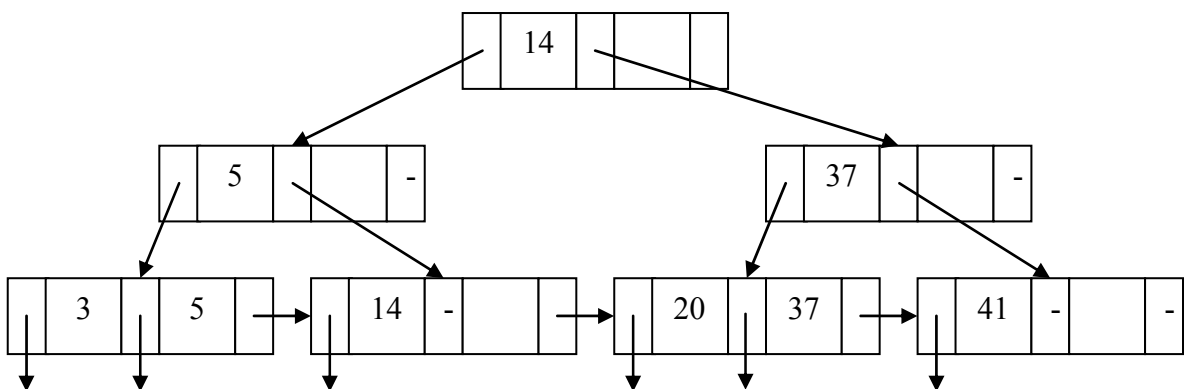


Рисунок 28. Вставка ключа 20.

Процесс вставки ключей закончен, получено сбалансированное дерево, состоящее из трех уровней.

## 5. ЯЗЫКИ БАЗ ДАННЫХ

### 5.1. SQL

#### 5.1.1. Общие сведения

SQL (Structured Query Language) – язык реляционных баз данных, который в настоящее время имеет очень широкое распространение.

В 1974 г. Д.Чамберлин (работал вместе с Е.Ф.Коддом в лаборатории IBM в Сан-Хосе, США) опубликовал определение языка SEQUEL (Structured English Query Language). В 1976 г. вышла переработанная версия языка – SEQUEL/2, в последствии из юридических соображений это название было изменено на SQL, поэтому в ходу остались два варианта произношения «си-кью-эл» и «эс-кью-эл».

Тестовая версия SQL входила в состав экспериментальной реляционной СУБД «System R». В 1981 г. IBM объявила о своем первом, основанном на SQL программном продукте – SQL/DS. Чуть позже к ней присоединились ORACLE и другие производители РСУБД.

В 1982 г. ANSI (американский национальный институт стандартов) начал работу над языком RDL (Relation Database Language). В работе над этим языком были использованы разработки языка SQL корпорации IBM. В 1983 г. К разработке языка подключился ISO (международный комитет по стандартизации). В процессе разработки название языка было заменено на SQL, первый стандарт которого вышел в 1987 г. Этот вариант языка был подвергнут критике со стороны разработчиков реляционной модели данных (отсутствие ссылочной целостности, недостаточное число операторов, чрезмерная избыточность языка), но стандарт был принят для развития основы языков реляционных БД. В 1992 г. вышла обновленная версия языка - SQL-92 или



SQL2. Расширение версии языка SQL2 для поддержки объектно-ориентированного управления данными называется SQL3, но этот вариант еще фактически находится в доработке, и в рамках курса рассматриваться не будет.

На сегодня для реляционных БД SQL является основным стандартом языка обработки данных. Разные СУБД реализуют стандарт SQL с добавлением дополнительных функций и синтаксических конструкций - расширений, поэтому можно говорить о различных диалектах языка SQL.

SQL – язык с трансформирующей ориентацией, т.е. язык для работы с таблицами данных с целью преобразования их к требуемому виду.

Язык SQL разделяется на два основных компонента:

- язык DDL (Data Definition Language) – язык определения структуры БД (DDL в свою очередь можно разделить на DDL (определение структуры БД) и DCL (управление доступом к данным));
- язык DML (Data Manipulation Language) – язык обработки данных.

В язык SQL не входят команды управления ходом вычислительного процесса (типа IF...ELSE, GOTO, DO...WHILE), поэтому SQL можно использовать либо в интерактивном режиме (режим работы пользователей), либо с использованием внедренных в языки высокого уровня SQL-операторов.

Основные особенности языка SQL:

- SQL не процедурный язык, поэтому в нем нужно указывать, какая информация должна быть получена, а не как ее можно получить - SQL не требует указания методов доступа к данным;
- SQL поддерживает свободный формат записи операторов;
- структура команд задается набором ключевых слов;
- язык SQL предназначен для широкого круга пользователей БД;
- стандарт языка SQL2 не поддерживает определений реляционной модели данных (например, вместо «отношение», «кортеж», «атрибут» используются понятия «таблица», «строка», «столбец») и не реализует полностью реляционную модель (например, в SQL таблица может иметь дублирующие строки, а домен – не множество значений, а тип данных

пользователя).

## 5.1.2. Язык DDL SQL

Язык DDL предназначен для формирования различных объектов БД – схем, доменов, таблиц, представлений, индексов и т.п.

### 5.1.2.1. Идентификаторы, типы данных, скалярные операторы

Идентификаторы SQL предназначены для обозначения объектов в БД и являются их именами. Идентификатор по стандарту ISO состоит из символов ('A'-'Z', 'a'-'z', '0'-'9' (не может быть первым символом), '\_'), длина идентификатора не более 128 символов.

По стандарту ISO в SQL определено шесть типов данных:

- символьные данные – позволяют задать последовательности символов  
CHARACTER [VARYING] [length]  
(сокращения CHAR, VARCHAR; VARYING – переменная длина);
- битовые данные – позволяют задать строки бит (символ может принимать значение '0' или '1')  
BIT [VARYING] [length]
- точные числа – используются для определения точного формата чисел  
NUMERIC [precision [, scale]]  
DECIMAL [precision [, scale]]  
INTEGER  
SMALLINT  
(сокращения INT, DEC; precision – число знаков, scale – число знаков дробной части);
- округленные числа – используются для определения чисел с плавающей

запятой

FLOAT [precision]

REAL

DOUBLE PRECISION

- дата и время – предназначены для определения моментов времени

DATE

TIME [time\_precision] [WITH TIME ZONE]

TIMESTAMP [time\_precision] [WITH TIME ZONE]

- интервальные данные – используются для представления периодов времени

INTERVAL {{start\_field TO end\_field} single\_datetime\_field}

start\_field = YEAR | MONTH | DAY | HOUR | MINUTE

[(interval leading field precision)]

end\_field = YEAR | MONTH | DAY | HOUR | MINUTE

[(interval leading field precision)]

single\_datetime\_field = start\_field | SECOND

[(interval leading field precision

[,fractional seconds precision])]

Дополнительные типы данных:

- денежные данные – описывают представление денежных величин

MONEY

- двоичные данные – позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.)

BINARY

BYTE

BLOB

- последовательные данные – используются для представления возрастающих числовых последовательностей

SERIAL

В стандарте ISO предусмотрены и встроенные скалярные операторы и функции, которые используются для построения скалярных выражений. Эти

операторы можно разделить на следующие группы:

1) '+', '-', '\*', '/' – арифметические операторы

2) операторы для работы с символьными и битовыми данными:

BIT\_LENGTH() – длина строки в битах;

OCTET\_LENGTH() – длина строки в байтах;

CHAR\_LENGTH() – длина строки в символах;

|| - оператор конкатенации;

LOWER(), UPPER(), TRIM() – функции преобразования символов;

POSITION(), SUBSTRING() – поиск позиции в строке.

3) операторы преобразования данных:

CAST(), EXTRACT().

4) операторы для получения состояния системных переменных:

CURRENT\_USER(), USER(), SESSION\_USER(), SYSTEM\_USER());

CURRENT\_DATE(), CURRENT\_TIME, CURRENT\_TIME\_STAMP.

#### 5.1.2.2. Создание структуры и объектов БД

В стандарте SQL содержится спецификация языка SQL, используемого для описания структуры БД, но не указывается способ создания БД, поэтому в каждом из диалектов языка SQL используется собственный подход к созданию БД. Например, в СУБД ORACLE БД создается в ходе процесса установки программного обеспечения. Как правило, таблицы пользователя помещаются в единую общесистемную базу данных, имя которой определяется файлом конфигурации ORACLE. Эта БД связана с данной конкретной копией серверного программного обеспечения. В СУБД Microsoft SQL Server для создания БД используется инструкция CREATE DATABASE. Имена создаваемых баз данных отслеживаются в специальной “главной” базе данных, которая связана с конкретной установленной СУБД.

Ядро инструкций DDL образуют три команды:

- CREATE - определить и создать объект базы данных;
- ALTER - изменить определение объекта базы данных;
- DROP - удалить существующий объект базы данных.

Все основные реляционные СУБД позволяют использовать указанные команды DDL во время своей работы. Таким образом, структура реляционной БД является динамической. Например, СУБД может создавать, удалять или изменять объекты БД, одновременно с этим обеспечивая доступ пользователям к БД, что является одним из преимуществ реляционных баз данных по сравнению с более ранними системами, в которых изменять структуру базы данных можно было только после прекращения работы СУБД.

По стандарту ISO, все объекты БД существуют в некоторой среде (environment). Среда, в свою очередь, состоит из каталогов (catalog). Каждый каталог состоит из набора схем (schema). Схема – это поименованная коллекция объектов БД, которые некоторым образом связаны друг с другом (все объекты БД должны быть описаны в схеме).

Операторы работы со схемами (реализованы не во всех СУБД):

```
CREATE SCHEMA name [AUTHORIZATION creator_identifier]
```

```
DROP SCHEMA name [RESTRICT | CASCADE]
```

В некоторых СУБД вместо схемы реализован объект БД (database):

```
CREATE DATABASE name
```

```
DROP DATABASE name
```

После создания общей структуры БД, эта структура заполняется объектами БД (таблицами, индексами, представлениями и т.п.).

Самый распространенный объект в БД – базовая таблица (table), представляющая реляционное отношение. Операторы для работы с базовыми таблицами следующие:

1) Создание базовой таблицы:

```
CREATE TABLE table_name (
```

```
{column_name data_type [NOT NULL] [UNIQUE]
```

```
[DEFAULT default_option] [CHECK (search_condition)] [, ...]}
```

```

[PRIMARY KEY (list_of_columns) [,]]
{[UNIQUE (list_of_columns), [, ...]]}
{[FOREIGN KEY (list_of_foreign_key_columns)
    REFERENCES parent_table_name [(list_of_candidate_key_columns)],
    [MATCH {PARTIAL | FULL}]
    [ON UPDATE referential_action]
    [ON DELETE referential_action]
] [, ...]}
{[CHECK (search_condition)] [, ...]}
)

```

После выполнения пользователем инструкции CREATE TABLE в БД формируется новая таблица, которой присваивается имя table\_name, и владельцем которой становится создавший её пользователь. Имя таблицы должно быть уникальным идентификатором, допустимым в SQL, и не конфликтующим с именами других таблиц БД. Созданная таблица будет пустой.

Обязательная часть в создании таблицы – задание описания столбцов таблицы. Определения столбцов представляют собой заключенный в скобки список, элементы которого отделены друг от друга запятыми. Порядок следования определений столбцов в списке соответствует расположению столбцов в таблице. Каждый столбец описывается следующими данными:

- column\_name – имя столбца, которое должно быть уникальным среди столбцов данной таблицы, но в разных таблицах имена столбцов могут совпадать;
- data\_type - тип данных столбца, показывающий, данные какого вида хранятся в столбце; вместо системного типа данных может быть записано имя домена, определенного пользователем;
- ограничение NOT NULL – ограничение на внесение неопределенных значений если оно задано, то СУБД предотвращает занесение в столбец значений NULL, в противном случае (и по умолчанию) значения NULL

допускаются;

- UNIQUE – проверка СУБД уникальности значений в столбце, если задано;
- DEFAULT – определение значения по умолчанию;
- CHECK – задание ограничений для данного столбца.

Пример создания простой таблицы, состоящей из трех столбцов:

```
CREATE TABLE student (  
  student_ID integer NOT NULL,  
  name char(100),  
  rating float DEFAULT 0.0);
```

Кроме определений столбцов таблицы, в инструкции CREATE TABLE может быть указана информация о первичном ключе таблицы после PRIMARY KEY путем перечисления списка столбцов, входящих в первичный ключ. Все столбцы, входящие в первичный ключ, должны быть заданы как NOT NULL. Кроме первичного ключа таблицы, после UNIQUE можно описать все потенциальные ключи отношения. Уникальность значений потенциальных ключей поддерживается СУБД. Например:

```
CREATE TABLE student (  
  student_ID integer NOT NULL,  
  name char(100),  
  rating float DEFAULT 0.0,  
  PRIMARY KEY (student_ID));
```

Задание внешних ссылок на другие таблицы задается в секции FOREIGN KEY:

- list\_of\_foreign\_key\_columns - список столбцов создаваемой таблицы, которые образуют внешний ключ;
- parent\_table\_name – родительская таблица, связь с которой создает внешний ключ;
- list\_of\_candidate\_key\_columns – список столбцов на которые будет указывать ссылка, если этот список опущен, то будет использован

первичный ключ родительской таблицы в порядке определения;

- MATCH – уточнение способа обработки значений NULL в составном внешнем ключе (FULL – либо все атрибуты внешнего заданы как NULL, либо все имеют существующие значения (по умолчанию); PARTIAL – допускается частично определенная ссылка);
- ON UPDATE, ON DELETE – определение действий при обновлении и удалении соответственно конкретного ссылочного кортежа родительской таблицы, на который есть ссылки в данной таблице (NO ACTION (RESTRICT) – блокировка операции пока существует хоть одна ссылка (это действие задано по умолчанию); CASCADE – каскадирование (распространение) операции на ссылки данной таблицы; SET NULL – в ссылки данной таблицы занести NULL (должно быть допустимо), SET DEFAULT – задать ссылке значение по умолчанию (должно быть задано в определении столбца ссылки)).

Когда СУБД создает таблицу, то сравнивается определение каждого внешнего ключа с определениями родительских таблиц. При этом СУБД проверяет, соответствуют ли друг другу внешний ключ и первичный ключ в связанных таблицах, как по числу столбцов, так и по типу данных. Для того чтобы такая проверка была возможна, связанная таблица уже должна быть определена. Если две или более таблиц образуют ссылочный цикл, то для первой создаваемой таблицы невозможно определить внешний ключ, так как связанная с ней таблица еще не существует. При этом СУБД откажется выполнять инструкцию CREATE TABLE, выдав сообщение о том, что в определении таблицы присутствует ссылка на несуществующую таблицу. В этом случае необходимо создать таблицу без определения внешнего, ключа и добавить данное определение с помощью инструкции ALTER TABLE, которая будет рассмотрена ниже.

Финальное ограничение CHECK позволяет определить бизнес-правила на данные всей строки, создаваемой таблицы. Оно содержит условие search\_condition на значения столбцов добавляемой или модифицируемой



строки. Если данные в строке таблицы не соответствуют условию, то выдается сообщение об ошибке.

Для удобства дальнейшего изменения структуры таблицы, каждая часть оператора CREATE TABLE может быть описана через именованное ограничение CONSTRAINT.

Пример таблицы с внешней ссылкой и реализацией бизнес правил:

```
CREATE TABLE student (  
  student_ID integer NOT NULL,  
  name char(100),  
  rating float DEFAULT 0.0,  
  group_ID integer,  
  PRIMARY KEY (student_ID),  
  FOREIGN KEY (group_ID) REFERENCES group  
  ON UPDATE CASCADE,  
  CONSTRAINT rating_OK CHECK ((rating > 0) and (rating <= 10.0)),  
  CONSTRAINT student_ID_OK CHECK (student_ID > 0));
```

2) Изменение структуры базовой таблицы:

```
ALTER TABLE table_name  
[ADD [COLUMN] column_name data_type [NOT NULL] [UNIQUE]  
  [DEFAULT default_option] [CHECK (search_condition)]]  
[DROP [COLUMN] column_name [RESTRICT | CASCADE]]  
[ADD [CONSTRAINT constraint_name] table_constraint_definition]  
[DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]]  
[ALTER [COLUMN] SET DEFAULT default_option]  
[ALTER [COLUMN] DROP DEFAULT]
```

Как видно из описания оператора, инструкция ALTER TABLE может:

- добавить в таблицу новый столбец (столбец добавляется в конец определений столбцов таблицы; СУБД обычно предполагает, что новый столбец во всех существующих строках содержит значения NULL, если

столбец объявлен как NOT NULL и имеет значение по умолчанию, то он будет заполнен значением по умолчанию);

- удалить существующий столбец из таблицы (следует учесть, что операция удаления столбца может вызвать проблемы целостности данных (например, при удалении столбца, являющегося первичным ключом в каком-либо отношении, связанные с ним внешние ключи становятся недействительными, такая же проблема возникает, когда удаляется столбец, участвующий в проверке ограничений); описанные проблемы разрешаются заданием правил удаления (RESTRICT - если с удаляемым столбцом связан какой-либо объект в базе данных (внешний ключ, ограничение и т.п.), то инструкция ALTER TABLE завершится выдачей сообщения об ошибке и столбец не будет удален; CASCADE - любой объект базы данных, связанный с удаляемым столбцом, также будет удален (использовать с осторожностью!)));
- добавить новое или удалить существующее ограничение (если для CREATE TABLE таблицы описано ограничение CONSTRAINT (например, при определении внешнего ключа), то удаление ограничения приведет к удалению этого ключа (в приведенном выше примере можно удалить бизнес-правила rating\_OK и student\_ID\_OK); иначе, если для какой-либо части определения CONSTRAINT не задан, то удалить эту часть с помощью ALTER нельзя);
- добавить новое или удалить существующее значение по умолчанию для какого-либо столбца.

Например, удаление значения по умолчанию будет описано так:

```
ALTER TABLE student  
ALTER rating DROP DEFAULT;
```

Добавление внешнего ключа будет описано так (если в таблице такой ключ не задан):

```
ALTER TABLE student  
ADD CONSTRAINT group_refer
```

FOREIGN KEY (group\_ID) REFERENCES group  
ON UPDATE CASCADE;

Во многих СУБД данный оператор может иметь другой синтаксис (функциональные возможности) или название (например, вместо регламентированного стандартом предложения ALTER СУБД ORACLE использует MODIFY). Стандарт требует, чтобы инструкция ALTER TABLE применялась для единичного изменения таблицы. Например, для добавления столбца и определения нового внешнего ключа потребуются две различные инструкции. В некоторых СУБД это ограничение ослаблено и допускается присутствие нескольких предложений в одной инструкции ALTER TABLE.

Оператор ALTER TABLE реализован не во всех СУБД, если важны данные таблицы, то их можно выгрузить в буферный файл, удалить таблицу, создать новую структуру таблицы и загрузить в нее данные из буфера.

3) Удаление базовой таблицы (включая все ее данные):

```
DROP TABLE table_name [RESTRICT | CASCADE]
```

Параметры CASCADE и RESTRICT определяют, как влияет удаление таблицы на другие объекты БД, зависящие от этой таблицы (например, на другие таблицы или представления, ссылающиеся на данную таблицу). Если задан параметр CASCADE и в БД имеются объекты, которые содержат ссылку на удаляемую таблицу, то выполнение инструкции закончится ошибкой. Применение CASCADE распространит оператор DROP на зависящие объекты (использовать осторожно, можно удалить всю БД!).

4) Работа с псевдонимами таблицы:

Для доступа к таблицам необходимо использовать полные имена таблиц, в результате чего запросы становятся длинными, а их ввод – утомительным. Для решения этой проблемы во многих СУБД вводится понятие псевдонима (синонима) – имя, назначаемое пользователем, которое заменяет имя некоторой таблицы. После создания псевдонима командой CREATE ALIAS его можно

использовать в запросах SQL как обычное имя таблицы. Псевдоним можно удалить посредством инструкции DROP ALIAS.

Для задания пользовательского типа столбцов в БД можно использовать домены (domain). Домены в SQL не соответствуют определению доменов в реляционной модели данных, а являются просто синтаксическими сокращениями и могут на практике не использоваться (в реляционной модели – домены обязательная часть БД). Также выполнение доменами ограничений сравнений в SQL не выполняется – нет строгого контроля типов, нельзя задать операции для домена (по сути дела домены в SQL даже не истинные типы данных). Для работы с доменами в SQL используются следующие операторы:

1) Создание домена:

```
CREATE DOMAIN domain_name [AS] data_type  
[DEFAULT default_option]  
[CHECK (search_condition)]
```

Определение параметров оператора аналогично определению столбцов базовой таблицы в операторе CREATE TABLE. Например:

```
CREATE DOMAIN rating AS FLOAT  
DEFAULT 0.0  
CHECK ((VALUE > 0.0) and (VALUE <= 10.0));
```

2) Удаление домена:

```
DROP DOMAIN domain_name [RESTRICT | CASCADE]
```

Определение параметров оператора аналогично удалению базовой таблицы в операторе DROP TABLE.

Для работы с индексами в SQL определен объект индекс (index) и операторы для работы с ним:

1) Создание индекса:

```
CREATE [UNIQUE] INDEX index_name ON table_name
```

(column [ASC | DESC] [, ...])

Создание индекса допустимо только для базовых таблиц, но не для представлений. Указанные в операторе столбцы составляют ключ индекса и должны быть перечислены в возрастающем (ASC, по умолчанию) или убывающем (DESC) порядке. UNIQUE – поддержка уникальности значений СУБД. Например:

```
CREATE UNIQUE INDEX student_ID_IND ON student (student_ID);
```

2) Удаление индекса:

```
DROP INDEX index_name
```

Для предоставления прав доступа к поименованным объектам БД в SQL можно задать оператор GRANT:

```
GRANT {privilege_list | ALL PRIVELEGES}  
ON object_name  
TO {authorization_id_list | PUBLIC}  
{WITH GRANT OPTION}
```

Список привилегий (privilege\_list) может содержать опции:

- SELECT - получение информации;
- UPDATE [(column\_name [, ...])] - изменение информации;
- INSERT [(column\_name [, ...])] - добавление записей;
- DELETE - удаление записей;
- REFERENCES [(column\_name [, ...])] – ссылки;
- INDEX - индексирование таблицы;
- ALTER - изменение схемы определения объекта;
- ALL PRIVELEGES - все права.

Использование фразы WITH GRANT OPTION позволяет всем указанным в списке authorization\_id\_list пользователям передавать другим пользователям все свои привилегии. Если в списке пользователей указано PUBLIC, то права выданы всем пользователям БД.

Отмена предоставленных пользователям привилегий выполняется оператором REVOKE:

```
REVOKE [GRAND OPTION FOR] {privilege_list | ALL PRIVILEGES}
ON object_name
FROM {authorization_list | PUBLIC} [RESTRICT | CASCADE]
```

Для работы с пользовательскими представлениями в SQL задан объект представление (view). Операторы для работы с представлениями следующие:

1) Создание представления:

```
CREATE VIEW view_name [(column_name [, ...])]
AS subselect [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Здесь, subselect – результирующая таблица запроса, для пересортировки столбцов которой может быть задан собственный набор столбцов представления (число столбцов должно соответствовать друг другу). Фраза WITH SELECT OPTION гарантирует, что строки представления, которые в результате работы с представлением перестанут соответствовать определяющему запросу, будут удалены из данного представления. Для того чтобы представление было обновляемым, СУБД должна иметь возможность однозначно отобразить любую его строку или столбец на соответствующую строку или столбец его исходной таблицы (для этого вносятся ограничения на запрос: запрещено использование DISTINCT, GROUP BY и HAVING; в запросе участвует только одна таблица, и не должно быть подзапросов с ней связанных; в списке столбцов не должно быть констант, выражений или обобщающих функций).

2) Удаление представления:

```
DROP VIEW view_name [RESTRICT | CASCADE]
```

Создание представлений в плане формирования запросов затрагивает оператор SELECT языка DML SQL, который будет рассмотрен в следующем разделе.

### 5.1.3. Язык DML SQL

Язык DML предназначен для обработки данных, в основном, формирования запросов для выборки данных и модификации данных (добавления, изменения и удаления записей) в заранее определенных объектах БД.

#### 5.1.3.1. Оператор выборки данных SELECT

Оператор SELECT предназначен для выборки и отображения необходимым образом данных БД. Инструкция SELECT извлекает информацию из базы данных и возвращает её в виде таблицы результатов запроса, имеющей такой же формат, что и обычные таблицы, содержащиеся в БД. Даже в случае если запрос возвращает ноль строк, его результат считается таблицей – пустой таблицей. Поддержка отсутствующих данных в SQL распространяется и на результаты запроса.

То, что запрос всегда возвращает таблицу, позволяет выполнять над этими результатами следующие действия:

- результаты запроса можно записать обратно в базу данных в виде таблицы;
- результаты запроса сами могут стать данными для дальнейших запросов.

##### 5.1.3.1.1. Общий формат оператора выборки

Общий формат оператора выборки SELECT:

```
SELECT [DISTINCT | ALL] {* | [column [AS new_column_name]] [, ...]}  
FROM table [alias] [...]
```

[WHERE condition]

[GROUP BY list [HAVING condition]]

[ORDER BY list]

Здесь использованы следующие обозначения:

- column – имя столбца (или константа, или выражение);
- DISTINCT - результат не будет содержать строк-дубликатов;
- ALL - результат может содержать дублирующие строки (по умолчанию);
- \* - все столбцы;
- table - имя таблицы;
- alias - сокращение для имени таблицы;
- condition - условие фильтрации строк данных;
- list - список столбцов;

Порядок секций (предложений) в операторе SELECT должен строго соблюдаться (например, GROUP BY должно всегда предшествовать ORDER BY), иначе это приведет к появлению ошибок. Обязательными секциями оператора являются первые две: SELECT и FROM.

В предложении SELECT, с которого начинаются все инструкции SELECT, необходимо указать элементы данных, которые будут возвращены в результате запроса. Эти элементы задаются в виде списка возвращаемых столбцов (column), разделенных запятыми. Для каждого элемента из этого списка в таблице результатов запроса будет создан один столбец. Столбцы в таблице результатов будут расположены в том же порядке, что и элементы списка возвращаемых столбцов. Возвращаемый столбец может представлять собой:

- имя столбца - один из столбцов, содержащихся в исходных таблицах; в результат подставляются значения из указанного столбца;
- константу - в каждой строке результатов запроса должно содержаться одно и то же константное значение;
- выражение - СУБД будет вычислять значение, помещаемое в таблицу результатов запроса, по формуле, заданной выражением, которые могут



состоять из имен столбцов, констант, скобок ('(', ')'), скалярных операторов (например: +, -, \*, /); при попытке выполнения арифметических операций над столбцами, содержащие данные неподходящих типов (например, текст), будет выдано сообщение об ошибке.

Предложение FROM состоит из ключевого слова FROM, за которым следует список таблиц, разделенных запятыми. Для каждой таблицы в списке FROM можно задать псевдоним (сокращение), для облегчения описания ссылок на таблицы.

Например, самый простой запрос - вернуть все столбцы таблицы student:

```
SELECT * FROM student;
```

Другой пример – вычисление выражений:

```
SELECT column1 AS koeff, '%', column1*100 AS procent FROM t1;
```

	column1	column2	column3	column4
▶	0.3	1	AAA	AAA
	0.4	2	BBB	
	0.9	3	AAB	
	0.25	4	BAB	
*	0	0		

а) исходные данные

	koeff	Expr1001	procent
▶	0.3	%	30.000001192
	0.4	%	40.000000596
	0.9	%	89.999997616
	0.25	%	25
*	0		

б) результат выполнения запроса

Рисунок 29. Запрос вычисления выражений.

На логическом уровне запрос выполняется путем построчного просмотра таблицы, указанной в предложении FROM. Для каждой строки таблицы берутся значения столбцов, входящих в список возвращаемых столбцов, и создается одна строка результатов запроса. Таким образом, таблица результатов простого запроса на выборку содержит одну строку для каждой строки исходной таблицы БД.

#### 5.1.3.1.2. Секция WHERE - фильтр строк данных по условию

Запросы, извлекающие из таблицы все строки, полезны только при просмотре БД и создании отчетов. Чаще требуется выбрать из таблицы несколько строк и включить в результаты запроса только их. Чтобы указать, какие строки нужно отобрать, следует использовать секцию фильтра WHERE.

Предложение WHERE состоит из ключевого слова WHERE, за которым следует условие отбора, определяющее, какие именно строки требуется извлечь. Если условие отбора имеет значение TRUE, строка будет включена в результаты запроса. Если же оно имеет значение FALSE или NULL, то строка исключается из результатов запроса.

Основные типы условий фильтрации:

1) Сравнение – использование операторов сравнения (>, <, >=, <=, =, <>), скобок ('(', ')'), логических связок (AND («и»), OR («или»), NOT («нет»)) и констант:

... WHERE ((rating > 0.0) AND (rating < 10.0)) ...

2) Проверка на принадлежность значений диапазону (оператор [NOT] BETWEEN ... AND ...):

... WHERE rating BETWEEN 0.0 AND 10.0 ...

или (инверсная форма)

... WHERE rating NOT BETWEEN 0.0 AND 10.0 ...

Следует отметить, что проверка на принадлежность диапазону может

быть реализована через сравнения.

3) Принадлежность к множеству – использование [NOT] IN (value\_list):

... WHERE student IN ('Ivanov', 'Petrov', 'Sidorov') ...

4) Соответствие символному шаблону – использование [NOT] LIKE 'template'. Специальные символы шаблона по стандарту: '%' – любая последовательность символов; '\_' – любой одиночный символ; для поиска одиночных символов '%' и '\_' можно задать ESCAPE символ. Например, для поиска по шаблону '15%' можно задать:

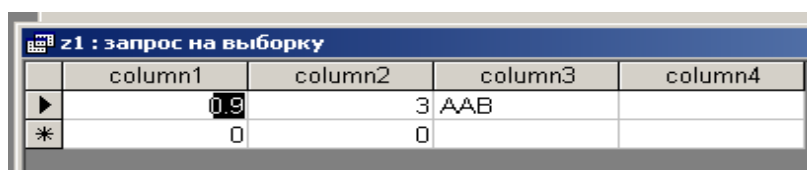
... WHERE data LIKE '15#%' ESCAPE '#' ...

5) Обработка неизвестных значений (NULL) – использование IS NULL или NOT NULL:

... WHERE rating IS NULL ...

Пример фильтрации (шаблон построен по правилам синтаксиса MS ACCESS (? = \_, \* = %)):

```
SELECT *
FROM t1
WHERE ((column1>0.2) AND (column2 IN (1,2,3)) AND
      (column3 LIKE '?A*') AND (column4 IS NULL));
```



	column1	column2	column3	column4
▶	0.9	3	AAB	
*	0	0		

Рисунок 30. Результат выполнения фильтрации (см. исходные данные на рисунке 29).

### 5.1.3.1.3. Обобщающие (агрегатные) функции

Для подведения итогов по информации, содержащейся в БД, в SQL предусмотрены обобщающие (агрегатные) функции, которые в качестве

аргументов принимают какой-либо столбец данных (столбец при этом представляется как множество данных), а возвращают одно значение, которое определенным образом подытоживает этот столбец.

В SQL имеется пять статистических функций:

- SUM() – для вычисления суммы всех значений столбца (только числовые типы);
- AVG() – для вычисления среднего значения столбца (только числовые типы);
- MIN() – определяет минимальное значение столбца;
- MAX() – определяет максимальное значение столбца;
- COUNT() – подсчитывает число всех определенных значений столбца.

Аргументом статистической функции может быть простое имя столбца или выражение на основе столбцов. На использование обобщающих функций накладываются следующие ограничения:

- обобщающие функции по стандарту SQL2 могут располагаться только в секциях SELECT и HAVING запроса SELECT;
- все обобщающие функции игнорируют NULL значения;
- не допускается вложение обобщающих функций друг в друга.

Варианты использования функции COUNT:

- COUNT(column) – количество значений в столбце column;
- COUNT(\*) - количество строк в таблице;
- COUNT (DISTINCT column) – число не повторяющихся значений в столбце column.

Запись DISTINCT в скобках перед именем столбца устраняет значения-дубликаты в данном столбце.

Пример использования обобщающих функций в простых запросах:

```
SELECT COUNT(*), COUNT(column4), SUM(column1), AVG(column2),  
MIN(column3), MAX(column3), SUM(column1+column2) FROM t1;
```

	Expr1000	Expr1001	Expr1002	Expr1003	Expr1004	Expr1005	Expr1006
▶	4	1	1.8499999940	2.5	AAA	BBB	11.849999994

Рисунок 31. Выполнение запроса с обобщающими функциями (см. исходные данные на рисунке 29).

Следует заметить, что в списке возвращаемых столбцов таких запросов нельзя одновременно указывать статистическую функцию и обычные имена столбцов, поскольку при этом СУБД не может сформировать таблицу – противоречия в структуре (одно значение и столбец значений).

#### 5.1.3.1.4. Группирующие запросы (секция GROUP BY)

Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходных таблиц в группы и для каждой группы строк генерирует одну строку в таблице результатов запроса. Столбцы, указанные в предложении GROUP BY, называются столбцами группировки (возможно указание нескольких столбцов – группировка по комбинации значений), поскольку именно они определяют, по какому признаку строки делятся на группы.

Ограничения на синтаксис группирующих запросов и особенности выполнения:

1) Столбцы с группировкой должны представлять собой реальные столбцы таблиц, перечисленных в предложении FROM. Нельзя группировать строки на основании значения вычисляемого выражения.

2) Все имена столбцов, приведенные в описании SELECT должны обязательно присутствовать и в секции GROUP BY. Это означает, что возвращаемым столбцом может быть:

- константа;
- статистическая функция, возвращающая одно значение для всех строк, входящих в группу;
- столбец группировки, который по определению имеет одно и то же значение во всех строках группы;
- выражение, включающее в себя перечисленные выше элементы.

3) Если совместно с GROUP BY используется WHERE, то WHERE обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию фильтра. По ISO, NULL-значения входят в одну группу.

Для фильтрации данных группы по заданным условиям используется подгруппа HAVING, которая в условиях отбора может использовать агрегатные функции (что запрещено делать в секции фильтра WHERE!). Кроме того, в HAVING нельзя записывать просто имена полей, не используемых в GROUP BY, т.к. эти данные будут представлять множество значений и сравнение с ними констант и множеств констант не возможно.

Пример группирующего запроса:

```
SELECT column2, COUNT(*), COUNT(column3), AVG(column1)
FROM t1
WHERE (column1 > 0.4)
GROUP BY column2 HAVING (COUNT(column3) > 1);
```

На логическом уровне данный запрос выполняется следующим образом:

1) Для всех строк таблицы t1 выполняется фильтрация по предикату (column1 > 0.4).

2) Данные, прошедшие фильтр WHERE, по столбцу column2 делятся на группы, по одной группе для каждого значения из column2.

3) Для каждой группы вычисляется COUNT(\*), COUNT(column3), AVG(column1) по всем строкам, входящим в группу, и генерируется одна итоговая строка результатов. К этой строке будет дописано соответствующее значение из column2.

4) Полученные результаты (данные групп) будут профильтрованы по предикату (COUNT(column3) > 1) в секции HAVING.

	column1	column2	column3	column4
	0.5	1	AAA	AAA
	0.6	1	AAA	
	0.7	1	AAB	
	0.8	2	BAB	
	0.9	2	BAB	
	0.4	2	DAA	
	0.6	2	AQQ	
	0.8	3	DEW	
	0.5	4	AQQ	
	0.7	4	FSD	
	0.4	4	FFA	
*	0	0		

а) исходные данные

	column2	Expr1001	Expr1002	Expr1003
	1	3	3	0.600000004
	2	3	3	0.7666666706
	4	2	2	0.5999999940

б) результат выполнения запроса

Рисунок 32. Выполнение группирующего запроса.

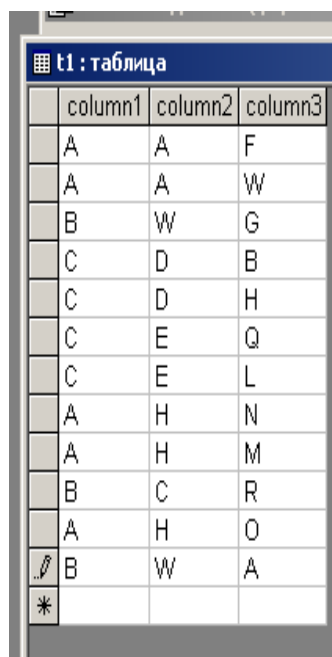
#### 5.1.3.1.5. Секция ORDER BY – сортировка результатов запроса

Секция ORDER BY определяет упорядоченность результатов по столбцам. Первый столбец в списке называется главным ключом и определяет общую упорядоченность строк результирующей таблицы. Последующие столбцы в списке сортировки определяют дополнительное упорядочивание в общей упорядоченности. Порядок сортировки данных столбца задается после имени столбца: ASC – сортировка по возрастанию значений (используется по

умолчанию); DESC – сортировка по убыванию значений. По ISO, NULL-значения либо наибольшие, либо наименьшие (на усмотрение разработчиков СУБД).

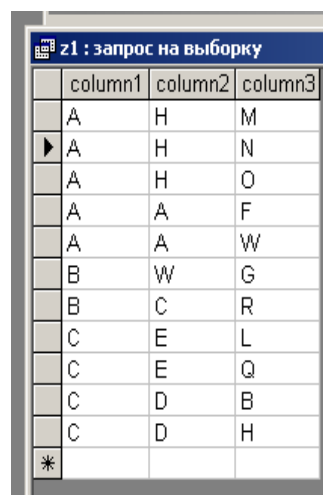
Пример использования сортировки:

```
SELECT *  
FROM t1  
ORDER BY column1, column2 DESC, column3;
```



column1	column2	column3
A	A	F
A	A	W
B	W	G
C	D	B
C	D	H
C	E	Q
C	E	L
A	H	N
A	H	M
B	C	R
A	H	O
B	W	A
*		

а) исходные данные



column1	column2	column3
A	H	M
A	H	N
A	H	O
A	A	F
A	A	W
B	W	G
B	C	R
C	E	L
C	E	Q
C	D	B
C	D	H
*		

б) результат выполнения запроса

Рисунок 33. Сортировка данных.

### 5.1.3.1.6. Подзапросы

Подзапросы – запросы с помощью оператора SELECT, помещенные в секции WHERE и (или) HAVING внешнего оператора SELECT. Подзапрос создает временную таблицу, содержимое которой извлекается и обрабатывается внешним оператором (обычно предикатом внешнего запроса). Текст подзапроса должен быть заключен в круглые скобки и располагается всегда в правой части операции внешнего запроса. В подзапросах не должна



использоваться секция ORDER BY.

Подзапрос играет важную роль в SQL по трем следующим причинам:

- инструкция SQL с подзапросом зачастую является самым естественным способом выражения запроса, так как она лучше всего соответствует словесному описанию запроса.
- подзапросы облегчают написание инструкции SELECT, поскольку они позволяют разбивать запрос на части (на основной запрос и подзапросы).
- существуют запросы, которые нельзя сформулировать на SQL иначе как с помощью подзапросов.

Подзапрос может вернуть следующее число значений:

- ничего,
- одно значение (ячейка),
- столбец значений (множество),
- таблицу значений (несколько столбцов).

Обработка данных вариантов обычно следующая:

1) При возврате одного значения обычно используются операторы сравнения (но для большей устойчивости запросов желательно '=' заменять на 'IN' (соответственно, '<>' на NOT IN), т.к. запрос может не вернуть ни одного значения; кроме того, в подзапросах желательно использовать обобщающие функции, которые всегда возвращают одно значение). Например:

```
SELECT * FROM t1
WHERE number > (SELECT AVG(rating) FROM t2);
```

2) При возврате множества значений (одного столбца) используется сравнение на принадлежность к множеству ('IN'), а также операторы ANY и ALL, которые используются как кванторы совместно с операторами сравнения. Для ANY(SOME) условие верно тогда, когда хоть одно значение, которое вернул подзапрос, удовлетворяет условию. Для ALL – условие верно только тогда, когда все значения, которые вернул подзапрос, удовлетворяют условию. Например:

```
SELECT * FROM t1
```

```
WHERE number IN (SELECT number FROM t2);
```

```
SELECT * FROM t1
```

```
WHERE number <= ANY (SELECT number FROM t2);
```

```
SELECT * FROM t1
```

```
WHERE number > ALL (SELECT number FROM t2);
```

Со всеми указанными конструкциями можно использовать логическое отрицание - NOT.

3) При возврате подзапросом таблицы (множество столбцов) можно проверить только факт наличия данных с помощью оператора EXISTS (если подзапрос ничего не возвращает, то результат EXISTS - ложь). Например:

```
SELECT *  
FROM t1  
WHERE EXISTS (SELECT * FROM t2);
```

Нет смысла использовать EXISTS, если подзапрос построен с помощью обобщающей функции, которая всегда возвращает значение. Можно изменить логику проверки EXISTS и использовать форму NOT EXISTS.

Обычно внешний и внутренний подзапросы ничем не связаны, однако есть случаи, когда подзапрос должен использовать данные из внешнего запроса, такие подзапросы называются соотнесенными. Например, найти данные о фирмах, которые имеют филиалы в городе 'Minsk' (при этом БД состоит из отношений t1 {firm, boss, about} и t2 {firm, city, branch\_address}):

```
SELECT *  
FROM t1 A  
WHERE 'Minsk' IN (SELECT city FROM t2 B WHERE A.firm = B.firm);
```

Данный запрос работает следующим образом:

- 1) берется строка таблицы для внешнего запроса (строка-кандидат);
- 2) подзапрос выполняется, используя данные из строки-кандидата;
- 3) производится оценка условия для внешнего запроса

4) переход к следующей строке таблицы.

Большинство запросов являются "двухуровневыми" и состоят из главного запроса и подзапроса. Однако, как внутри главного запроса может находиться подзапрос, то внутри подзапроса может находиться еще один подзапрос, называемый в таком случае вложенным. Стандарт ANSI/ISO не определяет максимальное число уровней вложенности, но на практике с ростом их числа очень быстро увеличивается время выполнения запроса. Когда запрос имеет более двух уровней вложенности, он становится трудным для чтения и понимания. Во многих СУБД количество уровней вложенности запросов ограничено относительно небольшим числом.

#### 5.1.3.1.7. Многотабличные запросы

Для выборки значений из нескольких таблиц БД используются многотабличные запросы. В многотабличных запросах используются операции соединения таблиц и, соответственно, в основе всех многотабличных запросов лежит операция декартового произведения.

Процедура выполнения многотабличного запроса состоит в следующем:

- 1) выполняется секция FROM (формируется декартово произведение таблиц и выполняется соответствующее соединение (если это задано синтаксисом секции FROM));
- 2) выполняется секция WHERE;
- 3) выполняется секция GROUP BY ... HAVING;
- 4) выполняется секция SELECT (формируются результирующие строки);
- 5) выполняется секция ORDER BY.

Виды соединений таблиц (соответствуют видам соединений из реляционной алгебры):

- 1) Декартово произведение двух таблиц:

```
SELECT t1.*, t2.*
```

```
FROM t1, t2;
```

2) Тета-соединение таблиц (используются знаки сравнения, на практике используется редко, так как трудно найти смысл соединения):

```
SELECT *  
FROM t1, t2  
WHERE (t1.number > t2.number);
```

3) Экви-соединение таблиц (выполняется по равенству значений общего атрибута, например значений первичного и внешнего ключа):

```
SELECT t1.*, t2.*  
FROM t1, t2  
WHERE (t1.number = t2.number);
```

или эквивалентный вариант соединения (inner или natural join):

```
SELECT t1.*, t2.*  
FROM t1 INNER JOIN t2 ON (t1.number = t2.number);
```

4) Внешние соединения таблиц (левое, правое и полное, соответственно):

```
SELECT t1.*, t2.*  
FROM t1 LEFT JOIN t2 ON t1.number = t2.number;
```

```
SELECT t1.*, t2.*  
FROM t1 RIGHT JOIN t2 ON t1.number = t2.number;
```

```
SELECT t1.*, t2.*  
FROM t1 FULL JOIN t2 ON t1.number = t2.number;
```

Кроме приведенных выше соединений, есть различные нестандартные применения многотабличных запросов, например найти все пары студентов имеющих один и тот же рейтинг (запрос формируется для одной таблицы t1):

```
SELECT A.name, B.name, A.rating
FROM t1 A, t1 B
WHERE A.rating = B.rating;
```

Здесь псевдоним существует только на время выполнения SELECT.

С увеличением количества таблиц в запросе резко возрастает объем работы, необходимой для выполнения запроса. Хотя ограничений на количество объединяемых таблиц нет, на практике высокие затраты на обработку многотабличных запросов во многих приложениях накладывают серьезные ограничения на использование многотабличных запросов. В запросах, для которых время выполнения критично, число используемых таблиц обычно не превышает трех.

#### 5.1.3.2. Операторы UNION, INTERSECT и EXCEPT

Операции UNION (объединение), INTERSECT (пересечение) и EXCEPT (разность) представляют собой операции над множествами элементов, которые рассмотрены в реляционной алгебре. Данные операции позволяют комбинировать результаты двух и более запросов в единую результирующую таблицу.

Формат записи операторов следующий:

```
(SELECT ...)
operator [ALL] [CORRESPONDING [BY {column1 [, ...]}]]
(SELECT ...)
```

Здесь:

- (SELECT ...) – подзапросы, возвращающие данные для выполнения оператора; таблицы сформированные этими подзапросами должны быть совместимы по соединению – т.е. они должны иметь одну и ту же структуру (смысл данных должен поддерживаться пользователем);
- operator – один из поддерживаемых (UNION, INTERSECT, EXCEPT

(MINUS));

- ALL – при указании в результирующей таблице будут оставлены дубликаты строк (по умолчанию дубликаты строк удаляются);
- CORRESPONDING – операция выполняется для общих столбцов таблиц (если указан список столбцов, то операция выполняется для указанных столбцов);

Например:

```
(SELECT * FROM student1)  
UNION CORRESPONDING name  
(SELECT * FROM student2);
```

### 5.1.3.3. Операторы изменения содержимого БД

К операторам модификации данных относятся операторы INSERT (вставка), UPDATE (изменение) и DELETE (удаление).

#### 5.1.3.3.1. Добавление (вставка) новых данных в таблицу

Наименьшей единицей информации, которую можно добавить в реляционную базу данных, является одна строка. Новые данные можно добавлять только в одну таблицу.

В РСУБД существует два способа добавления новых строк в базу данных:

1) Однострочная инструкция INSERT позволяет добавить в таблицу одну новую строку. Этот вариант широко используется в повседневных приложениях, например в программах ввода данных, и имеет формат:

```
INSERT INTO table_name [(column_list)]  
VALUES (data_value_list)
```

Здесь, table\_name – имя таблицы или обновляемого представления.

Параметр `column_list` – список столбцов, для которых добавляются данные, если данный параметр опущен, то используется список столбцов из оператора `CREATE TABLE` (в порядке определения столбцов). Параметр `data_value_list` – список значений данных. Должно выполняться соответствие списка столбцов и списка данных, как по количеству, так и по порядку присвоения и по типам данных. Если таблица содержит столбцы, не перечисленные в списке столбцов, то этим неуказанным столбцам будут присвоены значения по умолчанию, если они заданы, или `NULL` (если это не возможно, то возникает ошибка).

Пример данного вида оператора:

```
INSERT INTO student (student_ID, name, group_ID)
VALUES (10, 'Ivanov', 1);
```

2) Многострочная инструкция `INSERT` обеспечивает извлечение строк из одной части базы данных и добавление их в другую таблицу. Данная инструкция имеет следующий формат:

```
INSERT INTO table_name [(column_list)]
SELECT ...
```

Эта операция выполняет вставку множества строк в таблицу `table_name`. `Column_list` соответствует описанию из однострочной инструкции `INSERT`. Запрос `SELECT` формирует добавляемые строки данных. На этот запрос накладывается несколько логических ограничений:

- в запрос нельзя включать предложение `ORDER BY` – не имеет смысла сортировать таблицу результатов запроса, поскольку она добавляется в таблицу, которая не упорядочена;
- таблица результатов запроса должна содержать количество столбцов, равное длине списка столбцов в инструкции `INSERT` (или полностью всю целевую таблицу, если список столбцов опущен), а типы данных соответствующих столбцов таблицы результатов запроса и целевой таблицы должны быть совместимыми;
- имя целевой таблицы инструкции не может присутствовать в

предложении FROM запроса на выборку или любого запроса, вложенного в него, тем самым запрещается добавление таблицы самой к себе.

Пример данного вида оператора:

```
INSERT INTO student (student_ID, name, group_ID)
SELECT student_ID, name, group_ID
FROM new_student;
```

#### 5.1.3.3.2. Изменение данных в таблице

Инструкция UPDATE обновляет значения одного или нескольких столбцов в выбранных строках одной таблицы. Формат оператора UPDATE следующий:

```
UPDATE table_name
SET column_name1 = data_value1 [, column_name2 = data_value2 ...]
[WHERE search_condition]
```

Здесь, table\_name – целевая таблица, которая должна быть модифицирована. Предложение SET в инструкции представляет собой список операций присваивания, отделяемых друг от друга запятыми. В каждой операции идентифицируется целевой столбец, который должен обновляться, и определяется новое значение для этого столбца. Каждый целевой столбец должен встречаться в списке только один раз и не должно быть двух операций присваивания для одного и того же целевого столбца. Выражение в операции присваивания может быть любым допустимым SQL-выражением, результирующее значение которого имеет тип данных, соответствующий целевому столбцу. Необходимо, чтобы значение выражения вычислялось на основе значений строки, которая в данный момент обновляется в целевой таблице. Это выражение не может включать в себя какие-либо статистические функции или подчиненные запросы. Если выражение в операции присваивания содержит ссылку на один из столбцов целевой таблицы, то для вычисления



выражения используется значение этого столбца в текущей строке, которое было до обновления.

Пример полного обновления таблицы:

```
UPDATE student  
SET rating = rating + 1.0;
```

Предложение WHERE в инструкции UPDATE является необязательным. Если оно опущено, то обновляются все строки целевой таблицы. Если задано условие отбора WHERE, то обновлены будут только строки, удовлетворяющие этому условию. Условия отбора, которые можно задать в предложении WHERE инструкции UPDATE, полностью совпадают с условиями отбора, доступными в одноименном предложении инструкции SELECT рассмотренными ранее. Подчиненные запросы в предложении WHERE инструкции UPDATE могут иметь любой уровень вложенности и могут ссылаться на данные текущей строки-кандидата модифицируемой таблицы. В секции FROM подзапроса нельзя указывать модифицируемую таблицу.

Пример неполного обновления таблицы:

```
UPDATE student  
SET rating = rating + 1.0  
WHERE name IN (SELECT student_name FROM konkurs);
```

#### 5.1.3.3.3. Удаление данных из таблицы

Инструкция DELETE удаляет выбранные строки из одной таблицы. Формат оператора DELETE следующий:

```
DELETE FROM table_name  
[WHERE search_condition]
```

Здесь, table\_name - таблица, содержащая строки, которые требуется удалить. Пример полной очистки таблицы (сама таблица не удаляется из БД):

```
DELETE FROM student;
```

В предложении WHERE указывается критерий отбора строк, которые должны быть удалены. Условия отбора, которые можно задать в предложении WHERE инструкции DELETE, полностью совпадают с условиями отбора, доступными в одноименном предложении инструкции SELECT рассмотренными ранее. Подчиненные запросы в предложении WHERE инструкции DELETE могут иметь любой уровень вложенности и могут ссылаться на данные текущей строки-кандидата целевой таблицы. В секции FROM подзапроса нельзя указывать целевую таблицу.

Пример неполного удаления данных таблицы:

```
DELETE FROM student
```

```
WHERE name IN (SELECT student_name FROM prikaz);
```

## 5.2. QBE

QBE (Query By Example) – язык, который использует визуальный подход для организации доступа к информации в БД, построенный на применении шаблонов запросов. Данный подход был предложен в 1977 г. (Zloof). Работа в QBE осуществляется посредством задания образцов значений в шаблоне запроса, предусматривающем тот тип доступа к БД, который требуется пользователю в данный момент.

Язык QBE разрабатывался компанией IBM в 1970-е гг. и предназначался для пользователей, не имеющих профессионального компьютерного образования, но использующих БД. По своим возможностям QBE подобен SQL, однако более интуитивно понятен (например, в плане реализации таких специфичных запросов, как перекрестные запросы).

На сегодняшний день QBE не имеет официального стандарта, но он получил у пользователей достаточно широкое признание, поэтому во многих

популярных СУБД он реализован наряду с SQL. Возможности и порядок формирования запросов в QBE зависят от конкретной СУБД.

## 6. ТРАНЗАКЦИИ

### 6.1. Основные определения

Транзакция - это последовательность операторов манипулирования данными, выполняющаяся как единое целое (т.е. соответствует принципу «все или ничего») и переводящая базу данных из одного целостного состояния в другое целостное состояние. Операции, входящие в транзакцию, выполняют взаимосвязанные действия. Каждое из таких действий решает часть общей задачи, но для того, чтобы задачу можно было считать решенной, требуется выполнить весь набор операций без исключения.

Транзакции в СУБД представляют собой логическую единицу работы с БД и важны как в многопользовательских, так и в однопользовательских системах. В однопользовательских системах транзакции необходимы для поддержания БД в корректном (целостном) состоянии. В многопользовательских системах, транзакции служат также для обеспечения изолированной работы отдельных пользователей - т.е. пользователям, одновременно работающим с одной базой данных, кажется, что они работают как бы в однопользовательской системе и не мешают друг другу. Кроме того, транзакции также являются единицами восстановления данных (и целостности БД вообще) после сбоев и отказов.

Транзакция обладает четырьмя отличительными свойствами:

- (А) атомарность - транзакция выполняется как атомарная операция: либо выполняется вся транзакция целиком, либо она целиком не выполняется;
- (С) согласованность - транзакция переводит базу данных из одного согласованного (целостного) состояния в другое согласованное (целостное) состояние (однако внутри транзакции согласованность базы данных может нарушаться).
- (И) изоляция - транзакции разных пользователей должны выполняться

независимо друг от друга (например, как если бы они выполнялись последовательно, одна за другой) и не должны мешать друг другу;

- (Д) долговечность - если транзакция выполнена, то результаты ее работы должны сохраниться в БД, даже если в следующий момент произойдет сбой системы.

Выполнение транзакции в СУБД обычно начинается автоматически (начинается с выполнения пользователем или программой первой инструкции SQL) и продолжается до возникновения одного из следующих событий:

- фиксация транзакции командой COMMIT;
- откат транзакции командой ROLLBACK;
- успешное завершение сеанса работы с СУБД;
- сбой системы.

В SQL транзакция может быть явно начата путем ввода команды BEGIN TRANSACTION. Для завершения транзакции в SQL существуют следующие команды:

- COMMIT – сообщает СУБД об успешном окончании транзакции; результаты транзакции должны быть сохранены (зафиксированы) в БД;
- ROLLBACK - сообщает о неудачном окончании транзакции (например, пользователь не хочет сохранять изменения в БД или необходимые операции выполнить невозможно); СУБД должна отменить (откатить) все изменения, внесенные в БД в результате выполнения транзакции, т.е. СУБД должна вернуть базу данных в состояние, в котором она находилась до начала транзакции.

После выполнения COMMIT или ROLLBACK непосредственно может начинаться новая транзакция. Проверка успешности выполнения операций, входящих в транзакцию и выполнение отката транзакции (ROLLBACK) или подтверждение ее успешного выполнения (COMMIT) осуществляется обычно программно, посредством специально разработанных и стандартизированных интерфейсов. Программа в таком случае должна обеспечить корректную генерацию строк запросов SQL с целью дальнейшей их пересылки на

выполнение СУБД. В функцию интерфейсов помимо пересылки по сети пользовательских запросов входит также обеспечение корректного соединения с базами данных на основе существующих сетевых протоколов, а также предварительная обработка запросов, включающая их грамматический разбор и “связывание” параметров в предложениях WHERE.

База данных находится в согласованном состоянии, если для этого состояния выполнены все ограничения целостности. Под ограничение целостности понимают некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния БД.

Ограничения целостности классифицируются по следующим критериям:

1) По способам реализации – различают:

- декларативную поддержку ограничений целостности - определение ограничений средствами DDL; обычно средства декларативной поддержки целостности (если они имеются в СУБД) определяют ограничения на значения доменов и атрибутов, целостность сущностей (потенциальные ключи отношений) и ссылочную целостность (целостность внешних ключей); декларативные ограничения целостности можно использовать при создании и модификации таблиц средствами языка DDL или в виде отдельных утверждений (ASSERTION);
- процедурную поддержку ограничений целостности - посредством задания таких средств поддержания целостности СУБД, как триггеры и хранимые процедуры.

2) По времени проверки – проверка ограничений допускается как после выполнения каждого оператора, могущего нарушить ограничение, так и в конце транзакции. Во время выполнения транзакции можно изменить режим проверки ограничения. Различают:

- немедленно проверяемые ограничения – ограничения проверяются непосредственно в момент выполнения операции, могущей нарушить ограничение (например, проверка уникальности потенциального ключа проверяется в момент вставки записи в таблицу), если ограничение

нарушается, то такая операция отвергается;

- ограничения с отложенной проверкой – ограничение проверяется в момент фиксации транзакции (характерно для сложных транзакций )

3) По области действия – ограничения, накладываемые обычно на допустимые значения данных. Различают:

- ограничения домена;
- ограничения атрибута;
- ограничения кортежа;
- ограничения отношения;
- ограничения базы данных.

## 6.2. Параллельное выполнение транзакций

Выполнение транзакций в многопользовательских СУБД требует поддержания выполнения изолированности транзакций – СУБД должна гарантировать, что пользователи не будут мешать друг другу. В идеальном случае каждый пользователь должен работать с БД так, как если бы он имел к ней монопольный доступ, и не должен беспокоиться о действиях других пользователей.

### 6.2.1. Проблемы параллельного выполнения транзакций

Одна транзакция может быть представлена как последовательность элементарных атомарных операций. Элементарными операциями транзакции будут считаться, например, считывание страницы данных с диска или запись страницы данных на диск. На практике СУБД может выполнять несколько различных элементарных операций в один и тот же момент. Например, данные могут храниться на нескольких физически различных дисках и операции

чтения-записи на эти диски могут выполняться одновременно, а также параллельно с дисковыми операциями возможно выполнение операций с данными на процессорах сервера.

Элементарные операции различных транзакций могут выполняться в произвольной очередности (заметим, что внутри каждой транзакции последовательность элементарных операций этой транзакции является строго определенной). Например, если есть несколько транзакций, состоящих из последовательности элементарных операций:

$$T = \{T_1, T_2, T_3, \dots, T_n\}, Q = \{Q_1, Q_2, Q_3, \dots, Q_m\}, S = \{S_1, S_2, S_3, \dots, S_l\}$$

то реальная последовательность, в которой СУБД выполняет эти транзакции, может быть, например, такой:

$$\{T_1, Q_1, T_2, S_1, T_3, S_2, S_3, Q_2, \dots\}$$

В этом случае, набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется смесью транзакций.

Графиком выполнения транзакций называется последовательность запуска операций множества параллельно выполняемых транзакций, сохраняющая очередность выполнения операций в каждой отдельной транзакции. Для чередующихся операций различных транзакций график называется непоследовательным графиком, в отличие от строгого выполнения транзакций друг за другом поочередно – последовательного графика выполнения транзакций. Для заданного набора транзакций может быть достаточно много различных непоследовательных графиков запуска. Обеспечение изолированности пользователей, таким образом, сводится к выбору корректного непоследовательного графика запуска транзакций, приводящего к таким же результатам, что и в последовательном графике выполнения для данного набора транзакций. Данная концепция называется сериализацией транзакций. Одновременно с этим график запуска должен быть оптимальным в некотором смысле, например, давать минимальное среднее время выполнения транзакций каждым пользователем (заметим, что последовательный график – не оптимален по времени выполнения транзакций,



т.к. как говорилось выше, сервер может выполнять ряд операций по обработке данных параллельно). Для получения корректных графиков необходимо учитывать выполнение свойства изоляции транзакции во время выполнения.

Параллельное выполнение транзакций без выполнения свойства изоляции приводит к возникновению ряда проблемных ситуаций. К таким проблемам относятся:

1) Проблема потерянного обновления – данная проблема возникает тогда, когда две транзакции параллельно обновляют в БД одни и те же данные. При этом данные транзакции могут получить доступ к ячейке данных практически одновременно (т.е. получить для обработки одно и то же значение), а при записи обработанного значения обратно в эту ячейку последняя операция записи вызовет перекрытие предыдущего значения – потерю обновления данных. Например, при обработке снятия денег с вклада ( $100 - 10$ ) и параллельном добавлении ( $100 + 10$ ), результат обработки может быть равен либо 90, либо 110, что в обоих случаях не верно (в параллельном графике результат равен 100).

2) Проблема зависимости от незафиксированных результатов – данная проблема возникает в случае, когда вторая транзакция использует данные, измененные первой (еще незавершившейся транзакцией). При этом при возникновении проблем с первой транзакцией (что приведет к откату первой транзакции), вторая будет продолжать выполнение и зафиксирует фиктивные (уже не существующие) данные в БД. Например, первая транзакция снимает со счета деньги ( $100 - 10$ ), записывает в ячейку счета новое значение (90) и продолжает выполнение; если вторая транзакция также должна снять деньги с этого счета ( $90 - 10$ ), тогда при ошибке дальнейшего выполнения первой транзакции на счету будет зафиксировано неверное значение (80) после завершения второй транзакции.

3) Проблема несогласованной обработки – данная проблема возникает для транзакций ничего не меняющих в БД, а осуществляющих только чтение данных. Обычно такая транзакция (назовем ее первой) собирает статистику по

данным, например, выполняет расчет общей суммы денег на всех счетах. Если существуют параллельные транзакции, которые изменяют данные в БД, пока выполняется первая (например, переводят деньги с одного счета на другой), то работа первой транзакции при этом может приводить к неверным результатам, т.к. данные обрабатываются несогласованно и ранее считанные данные могут быть изменены. Такие проблемы еще называют проблемами “нестабильных результатов выборки” и “фантомов (строк-призраков)” (в последнем случае – проблема возникает для выполнения циклической обработки одних и тех же данных в первой транзакции (например, транзакция А дважды выполняет выборку строк с одним и тем же условием, а между выборками вклинивается транзакция В, которая добавляет новую строку, удовлетворяющую условию отбора)).

Как видно из примеров, если не предпринимать специальных мер, то при работе транзакций в смеси нарушается свойство изолированности транзакций.

Для получения корректного (сериализуемого) графика, необходимо разделить выполнение конкурирующих транзакций, т.е. таких которые пересекаются по времени и обращаются к одним и тем же данным. В результате конкуренции за данными между транзакциями возникают конфликты доступа к данным. Различают следующие виды конфликтов:

1) Запись-запись (W-W). Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - потеря обновления.

2) Запись-чтение (W-R). Первая транзакция изменила объект и не закончилась. Вторая транзакция пытается прочитать этот объект. Результат - чтение "грязных" данных.

3) Чтение-запись (R-W). Первая транзакция прочитала объект и не закончилась. Вторая транзакция пытается изменить этот объект. Результат - несовместимый анализ (неповторяемое считывание).

Конфликты будут отсутствовать только в ситуации типа чтение-чтение (R-R), т.к. данные при чтении не изменяются.

Другие проблемы параллелизма (фантомы и собственно несовместимый анализ) являются более сложными, т.к. принципиальное отличие их в том, что они не могут возникать при работе с одним объектом. Для возникновения этих проблем требуется, чтобы транзакции работали с целыми наборами данных.

Для построения корректных и оптимальных графиков для известного набора транзакций разработаны специальные алгоритмы упорядочивания (т.к. набор всех транзакций заранее известен, теоретически можно перебрать все возможные варианты графиков запусков (их конечное число, хотя и очень большое), и выбрать из них те графики, которые, корректны и оптимальны по выбранному критерию (например, времени выполнения транзакции)), однако, с точки зрения практического использования, использование этих алгоритмов в многопользовательских СУБД невозможно из-за того, что заранее не известен ни набор транзакций пользователей, ни время выполнения конкретной транзакции (число ее операций). В реальной ситуации необходимо, чтобы система работала так, чтобы к любому моменту времени набор выполненных и выполняющихся в этот момент транзакций был бы правильным и не слишком далек от оптимального.

Транзакции не мешают друг другу, если они обращаются к разным данным или выполняются в разное время, т.е. имеется два способа разрешить конкуренцию между поступающими в произвольные моменты транзакциями:

- задерживать выполнение некоторых транзакций, если это необходимо для обеспечения правильности смеси транзакций в каждый момент времени (т.е. обеспечить, чтобы конкурирующие транзакции выполнялись в разное время);
- выделить конкурирующим транзакциям разные экземпляры данных (т.е. обеспечить, чтобы конкурирующие транзакции работали с разными версиями данными, а проверку на наличие конфликтов проводить только на момент фиксации транзакции).

Первый метод рассчитывает, что число конфликтов выполнения транзакций в системе велико и называется «пессимистическим». Второй - что

число конфликтов мало и называется «оптимистическим».

## 6.2.2. Пессимистические методы параллельного выполнения транзакций

К пессимистическим методам параллельного выполнения транзакций относятся следующие методы: метод блокировки и метод временных отметок.

### 6.2.2.1. Метод блокировок

Основная идея метода блокировок заключается в том, что если для выполнения некоторой транзакции необходимо, чтобы некоторый объект не изменялся без ведома этой транзакции, то этот объект должен быть заблокирован, т.е. доступ к этому объекту со стороны других транзакций ограничивается на время выполнения транзакции, вызвавшей блокировку. В современных СУБД обычно реализован механизм блокировки на уровне строк таблиц БД.

Различают два типа блокировок:

- монополярная блокировка (X-locks - eXclusive locks) или блокировка по записи; если некоторая транзакция блокирует объект при помощи X-блокировки, то всякий доступ к этому объекту со стороны других транзакций отвергается, а эти транзакции переводятся до снятия блокировки в состояние ожидания.
- разделяемая блокировка (S-locks - Shared locks) или блокировка по чтению; если некоторая транзакция блокирует объект при помощи S-блокировки, то запросы со стороны других транзакций на X-блокировку этого объекта будут отвергнуты, а запросы со стороны других транзакций на S-блокировку этого объекта будут приняты.

Таким образом, механизм блокировок задает такие правила доступа к

данным, при которых исключаются конфликты типа R-W, W-R и W-W между транзакциями.

Доступ к объектам БД должен осуществляться в соответствии со следующим протоколом доступа к данным:

- прежде чем прочитать объект, транзакция должна наложить на этот объект S-блокировку;
- прежде чем обновить объект, транзакция должна наложить на этот объект X-блокировку; если транзакция уже заблокировала объект S-блокировкой (для чтения), то перед обновлением объекта S-блокировка должна быть заменена X-блокировкой;
- если блокировка объекта транзакцией В отвергается из-за того, что объект уже заблокирован транзакцией А, то транзакция В переходит в состояние ожидания; транзакция В будет находиться в состоянии ожидания до тех пор, пока транзакция А не снимет блокировку объекта;
- X-блокировки, наложенные транзакцией А, сохраняются до конца транзакции А.

Можно самостоятельно убедиться, что данный протокол метода блокировок позволяет устранить все проблемы параллельного выполнения транзакций (за исключением фантомов), приведенные выше.

К основным недостаткам метода блокировок можно отнести:

1) Возможность каскадных откатов транзакций – приводит к потере производительности. Каскадный откат транзакций возникает в том случае, когда X-блокировки не сохраняются до конца транзакции и возникает проблема чтения «грязных» данных, и соответственно отмена транзакций, которые воспользовались данными откатываемой транзакции. Решение данной проблемы – при написании каждой транзакции точно следовать двухфазному протоколу блокировок: в первой фазе выполнения (фазе нарастания) транзакции разрешено только блокировать и обрабатывать данные, вторая фаза (фаза сжатия), наступающая с первой разблокировки данных, запрещает блокировку новых данных, а допускает только разблокировку и обработку

данных (обычно вторая фаза на практике сводится к одной операции (завершения или отката транзакции) с одновременным снятием всех блокировок, кроме того, чем длиннее транзакция, тем больше вероятность возникновения проблем, поэтому лучше использовать как можно более короткие транзакции).

2) взаимная блокировка транзакций (dead locks) – возникает обычно при необходимости доступа нескольких транзакций более чем к одному общему ресурсу, т.к. например одна транзакция может заблокировать доступ к одному ресурсу и ждать второго, а вторая транзакция выполнит зеркальные действия – заблокирует второй ресурс, и будет ожидать доступа к первому. На практике в тупике может участвовать много транзакций, ожидающих друг друга. Т.к. нормального выхода из тупиковой ситуации нет, то такую ситуацию необходимо распознавать и устранять. Методом разрешения тупиковой ситуации является откат одной из транзакций (транзакции-жертвы) так, чтобы другие транзакции продолжили свою работу. После разрешения тупика, транзакцию, выбранную в качестве жертвы можно повторить заново. Существует два принципиальных подхода к обнаружению тупиковой ситуации и выбору транзакции-жертвы:

- СУБД не следит за возникновением тупиков. Транзакции сами принимают решение, быть ли им жертвой. Этот подход характерен для так называемых настольных СУБД (FoxPro и т.п.). Этот метод является более простым и не требует дополнительных ресурсов системы. Для транзакций задается время ожидания (или число попыток), в течение которого транзакция пытается установить нужную блокировку. Если за указанное время (или после указанного числа попыток) блокировка не завершается успешно, то транзакция откатывается (или генерируется ошибочная ситуация). За простоту этого метода приходится платить тем, что транзакции-жертвы выбираются, вообще говоря, случайным образом. В результате из-за одной простой транзакции может откатиться очень дорогая транзакция, на выполнение которой уже потрачено много

времени и ресурсов системы.

- За возникновением тупиковой ситуации следит сама СУБД, она же принимает решение, какой транзакцией пожертвовать. Этот способ характерен для промышленных СУБД (ORACLE, MS SQL Server и т.п.). В этом случае система сама следит за возникновением ситуации тупика, путем построения (или постоянного поддержания) графа ожидания транзакций. Граф ожидания транзакций - это ориентированный двудольный граф, в котором существует два типа вершин: вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. Ситуация тупика возникает, если в графе ожидания транзакций имеется хотя бы один цикл. Одну из транзакций, попавших в цикл, необходимо откатить, причем, система сама может выбрать эту транзакцию в соответствии с некоторыми стоимостными соображениями (например, самую короткую, или с минимальным приоритетом и т.п.).

3) Если рассматривать только блокировки данных по строкам таблицы, то при использовании приведенного выше протокола доступа к данным не решается проблема фантомов. Например, при добавлении новой строки в таблицу, т.к. эта строка может быть добавлена транзакцией В после выполнения транзакцией А всех своих блокировок на первом цикле обработки, а на втором цикле обработки в транзакции А будут получены новые данные. Для решения этой проблемы можно использовать блокировки более крупных объектов БД (всей БД, файлов, таблиц (LOCK TABLE), страниц, строк, полей), либо совместное использование блокировок объектов разной величины.

Чем крупнее объект блокировки, тем меньше возможностей для параллельной работы. Достоинством блокировок крупных объектов является уменьшение накладных расходов системы и решение проблем, не решаемых с использованием блокировок менее крупных объектов. Например, использование монопольной блокировки на уровне таблицы, очевидно, решит проблему фантомов, приведенную выше. Современные СУБД, как правило, поддерживают минимальный уровень блокировки на уровне строк или страниц.

При использовании блокировок объектов разной величины возникает проблема обнаружения уже наложенных блокировок разных уровней. Если транзакция А пытается заблокировать таблицу, то необходимо иметь информацию, не наложены ли уже блокировки на уровне строк этой таблицы, несовместимые с блокировкой таблицы. Для решения этой проблемы используется протокол преднамеренных блокировок, являющийся расширением протокола доступа к данным. Суть этого протокола в том, что перед тем, как наложить блокировку на объект (например, на строку таблицы), необходимо наложить специальную преднамеренную блокировку (блокировку намерения) на объекты, в состав которых входит блокируемый объект - на таблицу, содержащую строку, на файл, содержащий таблицу, на базу данных, содержащую файл. Тогда наличие преднамеренной блокировки таблицы будет свидетельствовать о наличии блокировки строк таблицы и для другой транзакции, пытающейся заблокировать целую таблицу не нужно проверять наличие блокировок отдельных строк. Поэтому необходимо введение дополнительных типов блокировок:

- Преднамеренная блокировка с возможностью взаимного доступа (IS-блокировка - Intent Shared lock). Накладывается на некоторый составной объект Т и означает намерение заблокировать некоторый входящий в Т объект в режиме S-блокировки. Например, при намерении читать строки из таблицы Т, эта таблица должна быть заблокирована в режиме IS (до этого в таком же режиме должен быть заблокирован файл).
- Преднамеренная блокировка без взаимного доступа (IX-блокировка - Intent eXclusive lock). Накладывается на некоторый составной объект Т и означает намерение заблокировать некоторый входящий в Т объект в режиме X-блокировки. Например, при намерении удалять или модифицировать строки из таблицы Т эта таблица должна быть заблокирована в режиме IX (до этого в таком же режиме должен быть заблокирован файл).
- Преднамеренная блокировка как с возможностью взаимного доступа, так



и без него (SIX-блокировка - Shared Intent eXclusive lock). Накладывается на некоторый составной объект T и означает разделяемую блокировку всего этого объекта с намерением впоследствии заблокировать какие-либо входящие в него объекты в режиме X-блокировок. Например, если выполняется длинная операция просмотра таблицы с возможностью удаления некоторых просматриваемых строк, то можно заблокировать эту таблицу в режиме SIX (до этого захватить файл в режиме IS).

IS, IX и SIX-блокировки должны накладываться на сложные объекты базы данных (таблицы, файлы). Кроме того, на сложные объекты могут накладываться и блокировки типов S и X. Для сложных объектов (например, для таблицы БД) совместимость блокировок имеет следующий вид:

	Транзакция В пытается наложить на таблицу блокировку				
Транзакция А наложила на таблицу блокировку	IS	S	IX	SIX	X
IS	Да	Да	Да	Да	Нет
S	Да	Да	Нет	Нет	Нет
IX	Да	Нет	Да	Нет	Нет
SIX	Да	Нет	Нет	Нет	Нет
X	Нет	Нет	Нет	Нет	Нет

Тогда точная формулировка протокола преднамеренных блокировок для доступа к данным выглядит следующим образом:

- при задании X-блокировки для сложного объекта неявным образом задается X-блокировка для всех дочерних объектов этого объекта;
- при задании S- или SIX-блокировки для сложного объекта неявным образом задается S-блокировка для всех дочерних объектов этого

объекта;

- прежде чем транзакция наложит S- или IS-блокировку на заданный объект, она должна задать IS-блокировку (или более сильную) по крайней мере, для одного родительского объекта этого объекта;
- прежде чем транзакция наложит X-, IX- или SIX-блокировку на заданный объект, она должна задать IX-блокировку (или более сильную) для всех родительских объектов этого объекта;
- прежде чем для данной транзакции будет отменена блокировка для данного объекта, должны быть отменены все блокировки для дочерних объектов этого объекта.

Понятие относительной силы блокировок можно описать при помощи следующей цепочки (от более слабой к более сильной блокировке):

$$IS \rightarrow \frac{S}{IX} \rightarrow SIX \rightarrow X$$

Кроме блокирования самих объектов БД, может быть использована блокировка условий, которым могут удовлетворять объекты. Такие блокировки называются предикатными блокировками.

Поскольку любая операция над реляционной базой данных задается некоторым условием (т.е. в ней указывается не конкретный набор объектов БД, над которыми нужно выполнить операцию, а условие, которому должны удовлетворять объекты этого набора), то удобным способом было бы S- или X-блокирование именно этого условия. Однако при попытке использовать этот метод в реальной СУБД возникает трудность определения совместимости различных условий. Действительно, в языке SQL допускаются условия с подзапросами и другими сложными предикатами. Проблема совместимости сравнительно легко решается для случая простых условий, имеющих вид:

{Имя атрибута {= | <> | > | >= | < | <=} Значение}

[{OR | AND} {Имя атрибута {= | <> | > | >= | < | <=} Значение}...]

Проблема фантомов легко решается с использованием предикатных

блокировок, т.к. вторая транзакция не может вставить новые строки, удовлетворяющие уже заблокированному условию. Заметим, что блокировка всей таблицы в каком-либо режиме фактически есть предикатная блокировка, т.к. каждая таблица имеет предикат, определяющий какие строки содержатся в таблице и блокировка таблицы есть блокировка предиката этой таблицы.

#### 6.2.2.2. Метод временных отметок

Другой пессимистический метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных отметок.

Основная идея метода состоит в следующем: если транзакция А началась раньше транзакции В, то система обеспечивает такой режим выполнения, как если бы А была целиком выполнена до начала В. Для этого, каждой транзакции Т предписывается уникальная временная метка  $t$ , соответствующая времени начала Т. При выполнении операции над объектом  $r$  БД на этот объект ставится временная отметка транзакции Т и тип операции (чтение или изменение).

Перед выполнением операции транзакции В над объектом  $r$ , СУБД выполняет следующие действия:

1) Проверяет, не закончилась ли транзакция А, пометившая этот объект. Если А закончилась, то объект  $r$  помечается временной меткой транзакции В и транзакция В выполняет над объектом необходимую операцию.

2) Если транзакция А не завершилась, то проверяется конфликтность операций над данными. Если операции неконфликтны, при объекте  $r$  остается или проставляется временная метка с меньшим значением (более ранняя), и транзакция В выполняет свою операцию.

3) Если операции В и А конфликтуют, то если  $t(A) > t(B)$  (т.е. транзакция А является более "молодой", чем В), то транзакция А откатывается и, получив

новую временную метку, начинается заново. Транзакция В продолжает работу.

4) Если же  $t(A) < t(B)$  (А "старше" В), то транзакция В откатывается и, получив новую временную метку, начинается заново. Транзакция А продолжает работу.

В результате, меток временных отметок обеспечивает такую работу, при которой при возникновении конфликтов всегда откатывается более молодая транзакция (начавшаяся позже).

Недостатки метода временных отметок:

- может откатиться более дорогая транзакция, начавшаяся позже более дешевой;
- потенциально более частые откаты транзакций, чем в случае использования метода блокировок, т.к. конфликтность транзакций определяется более грубо.

### 6.2.3. Оптимистические методы параллельного выполнения транзакций

Использование пессимистичных методов выполнения транзакций гарантирует сериальность планов выполнения смеси транзакций за счет общего замедления работы - конфликтующие транзакции либо ожидают (метод блокировок), либо откатываются (метод временных отметок).

Однако, в некоторых типах вычислительных систем конфликты между транзакциями происходят очень редко (например, мало изменений данных) , поэтому в таких системах пессимистический подход к выполнению транзакций оказывается излишним (перезапуск транзакции в такой системе в случае конфликта более производителен, т.к. случается крайне редко).

Обобщенный оптимистический протокол управления параллельностью состоит из трех фаз:

1) Фаза чтения. Данная фаза начинается от начала транзакции и выполняется до тех пор, пока не потребуются выполнение фиксации

результатов транзакции в БД. На этой фазе транзакция считывает значения всех необходимых ей элементов данных и помещает их в локальную область памяти, доступную только данной транзакции. Все изменения данных разрешены транзакции только в этой области.

2) Фаза проверки. Данная фаза следует за фазой чтения. На этой фазе выполняются проверки, необходимые для получения гарантий нарушения целостности БД в случае переноса в нее изменений данных из локальной копии. Для транзакций, включающих только операции чтения, проверка состоит в подтверждении того, что использованные данные не менялись другими транзакциями. Если транзакция обновляла значения данных, то выполняется контроль, имели ли другие транзакции доступ к этим данным. Проверки на этой фазе выполняются с использованием временных отметок Start (начало), Validation (начало проверки) и Finish (завершение выполнения транзакции) и сравниваются с такими же отметками других транзакций по правилам (для подтверждения должно выполняться одно из правил):

- все транзакции S с более старыми временными отметками должны быть уже завершены до начала выполнения проверяемой транзакции T ( $Finish(S) < Start(T)$ );
- если транзакция T стартовала до завершения какой-либо из транзакций S, то множество элементов данных, записанных стартовавшей ранее транзакцией, не должно включать ни одного из элементов данных, прочитанных данной транзакцией;
- если транзакция T стартовала до завершения какой-либо из транзакций S, то фаза записи стартовавшей ранее транзакции S должна быть завершена до перехода текущей транзакции T в фазу проверки ( $Start(T) < Finish(S) < Validation(T)$ ).

При обнаружении ошибок данная транзакция отменяется и перезапускается. Если же ошибок на фазе проверки не обнаружено, то выполняется переход к третьей фазе.

3) Фаза записи. Данная фаза выполняется только после успешно

проведенной фазы проверки, но только для тех транзакций, которые выполняли обновление локальных данных. Эти изменения переносятся собственно в БД.

Следует заметить, что откат данных в такой системе не выполняется, т.к. при отмене транзакции ее локальная копия просто уничтожается, также в принципе невозможны каскадные откаты. Однако длинные транзакции в таких системах не приветствуются, по причине большой затраты процессорного времени в случае полного перезапуска транзакции.

Если перезапуск транзакций возникает достаточно часто, то это может указывать на то, что оптимистические технологии мало подходят для управления параллельностью в данной конкретной системе.

#### 6.2.4. Реализация изолированности транзакций средствами SQL

Стандарт SQL не предусматривает понятие блокировок для реализации сериализуемости смеси транзакций. Вместо этого вводится понятие уровней изоляции. Этот подход обеспечивает необходимые требования к изолированности транзакций, оставляя возможность производителям различных СУБД реализовывать эти требования своими способами (в частности, с использованием блокировок или выделением версий данных).

Стандарт SQL предусматривает 4 уровня изоляции:

- READ UNCOMMITTED - уровень незавершенного считывания;
- READ COMMITTED - уровень завершенного считывания;
- REPEATABLE READ - уровень повторяемого считывания;
- SERIALIZABLE - уровень способности к упорядочению.

Если все транзакции выполняются на уровне SERIALIZABLE (принятом по умолчанию), то чередующееся выполнение любого множества параллельных транзакций может быть упорядочено. Если некоторые транзакции выполняются на более низких уровнях, то имеется множество способов нарушить способность к упорядочению. В стандарте SQL выделены три особых случая

нарушения способности к упорядочению, фактически именно те, которые были описаны выше как проблемы параллелизма:

- неаккуратное считывание ("грязное" чтение, зависимость от незафиксированных результатов);
- неповторяемое считывание (частный случай несогласованной обработки);
- фантомы (фиктивные элементы - частный случай несогласованной обработки).

Потеря результатов обновления стандартом SQL не допускается, т.е. на самом низком уровне изолированности транзакции должны работать так, чтобы не допустить потери результатов обновления.

Различные уровни изоляции определяются по возможности или исключению этих особых случаев нарушения способности к упорядочению. Эти определения описываются следующим образом:

Уровень изоляции	Неаккуратное считывание	Неповторяемое считывание	Фантомы
READ UNCOMMITTED	Да	Да	Да
READ COMMITTED	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет

Уровень изоляции транзакции задается следующим оператором:

```
SET TRANSACTION {ISOLATION LEVEL
{READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SERIALIZABLE}
| {READ ONLY | READ WRITE}};...
```

Этот оператор определяет режим выполнения следующей транзакции, т.е.

этот оператор не влияет на изменение режима той транзакции, в которой он подается. Обычно, выполнение оператора SET TRANSACTION выделяется как отдельная транзакция. Если задано предложение ISOLATION LEVEL, то за ним должно следовать один из параметров, определяющих уровень изоляции. Кроме того, можно задать признаки READ ONLY или READ WRITE. Если указан признак READ ONLY, то предполагается, что транзакция будет только читать данные. При попытке записи для такой транзакции будет сгенерирована ошибка. Признак READ ONLY введен для того, чтобы дать производителям СУБД возможность уменьшать количество блокировок путем использования других методов сериализации.

Оператор SET TRANSACTION должен удовлетворять следующим условиям:

- если предложение ISOLATION LEVEL отсутствует, то по умолчанию принимается уровень SERIALIZABLE;
- если задан признак READ WRITE, то параметр ISOLATION LEVEL не может принимать значение READ UNCOMMITTED;
- если параметр ISOLATION LEVEL определен как READ UNCOMMITTED, то транзакция становится по умолчанию READ ONLY, в противном случае по умолчанию транзакция считается как READ WRITE.

### 6.3. Восстановление данных

Восстановление данных после сбоя системы связано с таким свойством транзакций, как долговечность. Главное требование долговечности данных транзакций состоит в том, что данные зафиксированных транзакций должны сохраняться в системе, даже если в следующий момент произойдет сбой системы.



### 6.3.1. Влияние сбоев и отказов на БД

Работа с данными в СУБД проводится с использованием двух видов памяти:

- долговременная память – память, которая может хранить данные при выключении питания системы; данный вид памяти характеризуется большими объемами хранимой информации и достаточно большим временем доступа к данным (плохая производительность по сравнению с производительностью процессора); используется для хранения рабочей версии БД (накопители на жестких дисках) или ее копии (магнитные ленты, оптические диски);
- кратковременная память – память, которая хранит данные только при сохранении питания (ОЗУ); такая память характеризуется малым временем доступа к данным, но также и малыми объемами хранения информации; используется для промежуточного хранения данных.

Для поддержки необходимой производительности системы нельзя непосредственно оперировать данными, хранящимися в долговременной памяти системы, т.к. скорость работы с данными снизится и очень критично. Поэтому для работы с данными используется их промежуточная буферизация в оперативной памяти. Это означает, что текущие данные попадают во внешнюю долговременную память не сразу после внесения изменений, а через некоторое (достаточно большое) время.

Буфер памяти в СУБД в общем случае состоит из страниц. Страницы базы данных, содержимое которых в буфере отличается от содержимого на диске, называются "грязными" (dirty) страницами. Система постоянно поддерживает список "грязных" страниц - dirty-список. Запись "грязных" страниц из буфера на диск называется выталкиванием страниц во внешнюю память. Для обеспечения максимальной скорости выполнения транзакций необходимо выталкивать страницы на диск как можно реже. Если бы

оперативная память была бы бесконечной, а сбои никогда бы не происходили, то наилучшим выходом была бы загрузка всей БД в оперативную память и работа с данными только в оперативной памяти, а запись измененных страниц на диск только в момент завершения работы всей системы. Однако, на настоящий момент объемы оперативной (энергозависимой) памяти не могут вместить всех обрабатываемых данных, и нет полной защиты от сбоев, поэтому данные должны из буфера памяти время от времени выталкиваться на диск (даже если бы мы обладали бесконечной, но энергозависимой памятью).

Влияние внешних и внутренних факторов на память системы приводит к утрате необходимых данных. Восстановление БД может производиться в следующих характерных случаях:

1) Индивидуальный откат транзакции. Откат индивидуальной транзакции может быть инициирован либо самой транзакцией путем подачи команды ROLLBACK, либо СУБД в случае возникновения какой-либо ошибки в работе транзакции (например, деление на нуль) или если эта транзакция выбрана в качестве жертвы при разрешении тупика. При таких сбоях нет потерь данных в обоих видах памяти системы.

2) Мягкий сбой системы. Мягкий сбой характеризуется утратой оперативной памяти системы. При этом поражаются все выполняющиеся в момент сбоя транзакции, теряется содержимое буферов. Данные, хранящиеся в долговременной памяти, при таких сбоях остаются неповрежденными. Мягкий сбой может произойти, например, в результате аварийного отключения электрического питания, в результате неустранимого сбоя других электронных компонентов системы (например, процессора) или аварийного отказа программного обеспечения (например, сбой операционной системы).

3) Жесткий сбой системы (аварийный отказ аппаратуры). Жесткий сбой характеризуется повреждением долговременной памяти. Жесткий сбой может произойти, например, в результате поломки головок дисковых накопителей. При таком сбое обычно теряется и содержимое буферов оперативной памяти.

### 6.3.2. Средства защиты данных

Для защиты от сбоев и отказов в СУБД предусмотрены специализированные методы и средства защиты данных:

1) Резервное копирование данных. Резервное копирование данных представляет собой дублирование БД на другом долговременном носителе. Выполнение резервного копирования обычно осуществляется в периоды минимальной работы с данными, поэтому обычно выполняется не чаще одного раза в сутки (например, ночью). Резервное копирование может выполняться в следующих вариантах:

- полное дублирование БД – требует достаточно больших объемов памяти резервного носителя, но не требует хранения всех предыдущих копий БД;
- дублирование только изменений в данных, произошедших с момента предыдущего резервного копирования, характеризуется небольшими объемами хранимых данных за одну сессию резервирования, но зависит от всех предыдущих сессий.

В настоящее время для резервного копирования обычно используются оптические носители (CD- (DVD-) ROM или RW).

2) Ведение журнала транзакций. Требование атомарности транзакций утверждает, что не законченные или откатившиеся транзакции не должны оставлять следов в БД. Это означает, что данные должны храниться с избыточностью, позволяющей иметь информацию, по которой восстанавливается состояние БД на момент начала неудачной транзакции. Такую избыточность обычно обеспечивает журнал транзакций. Журнал транзакций содержит детали всех операций модификации данных в БД, в частности:

- системный номер транзакции;
- время начала и окончания транзакции;
- старое и новое значение модифицированного транзакцией объекта и вид

проводимой операции;

- системная статистика (пользователи, системные ошибки, свободное место на диске и т.п.).

Как и страницы БД, данные журнала транзакций не записываются сразу на диск, а предварительно буферизируются в оперативной памяти. Основным принципом работы с буфером журнала является то, что запись об изменении объекта БД должна попадать во внешнюю память журнала раньше, чем измененный объект данных оказывается во внешней памяти БД. Соответствующий протокол журнализации (и управления буферизацией) называется WAL (Write Ahead Log - пиши сначала в журнал). Это означает, что если требуется вытолкнуть во внешнюю память измененный объект БД, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи о его изменении. Таким образом, если во внешней памяти БД содержится объект, к которому применена некоторая команда модификации, то во внешней памяти журнала транзакций содержится запись об этой операции. Обратное неверно - если во внешней памяти журнала содержится запись о некотором изменении объекта, то во внешней памяти базы данных может и не быть самого измененного объекта. Обычно все изменения вносимые транзакцией в журнал не записываются сразу на диск, т.к. это требует больших временных затрат. Минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния БД, является выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении БД этой транзакцией. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце этой транзакции.

3) Своевременная очистка буферов оперативной памяти. Дополнительным условием выталкивания буферов является ограниченность объемов буферов БД и журнала транзакций. Периодически или при наступлении определенного события (например, количество страниц в dirty-списке превысило определенный порог, или количество свободных страниц в

буфере уменьшилось и достигло критического значения) система принимает так называемую контрольную точку. Принятие контрольной точки включает выталкивание во внешнюю память содержимого буферов БД и специальную физическую запись контрольной точки, которая представляет собой список всех осуществляемых в данный момент транзакций.

4) Повышение надежности долговременных носителей данных (в том числе дублирование данных средствами носителя (например, использование RAID массивов в режиме зеркала)).

### 6.3.3. Методы восстановления данных

Во всех трех случаях потерь данных основой восстановления является избыточность данных, обеспечиваемая журналом транзакций. Рассмотрим методы восстановления данных, используемые СУБД в каждом конкретном случае:

1) Выполнение индивидуального отката транзакции.

Для того чтобы можно было выполнить индивидуальный откат транзакции, все записи в журнале от данной транзакции связываются в обратный список. Началом списка для не закончившихся транзакций является запись о последнем изменении БД, произведенном данной транзакцией, а для закончившихся транзакций (индивидуальные откаты которых уже невозможны (каскадные откаты)) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит запись об первом изменении БД, произведенном данной транзакцией. В каждой записи имеется уникальный системный номер транзакции, чтобы можно было восстановить прямой список записей об изменениях БД данной транзакцией.

Индивидуальный откат транзакции выполняется следующим образом:

- просматривается обратный список записей, сделанных данной

транзакцией в журнале транзакций (т.е. список операций от последнего изменения к первому изменению);

- для каждой операции из списка формируется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE выполняется INSERT, и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта БД;
- откат транзакции выполняется как специальная транзакция, любая операция которой также журналируется - это выполняется так, потому что во время выполнения индивидуального отката может произойти сбой, при восстановлении после которого потребуются откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат;
- при успешном завершении отката в журнал заносится запись о конце транзакции.

## 2) Восстановление БД после мягкого сбоя.

Несмотря на протокол WAL, после мягкого сбоя не все физические страницы базы данных содержат измененные данные, т.к. не все "грязные" страницы базы данных были вытолкнуты во внешнюю память. Последний момент, когда гарантированно были вытолкнуты "грязные" страницы - это момент принятия последней контрольной точки. На рисунке 34 указаны возможные варианты состояния транзакций по отношению к моменту последней контрольной точки (tc) и к моменту сбоя (tf):

- T1 - транзакция успешно завершена до принятия контрольной точки. Все данные этой транзакции сохранены в долговременной памяти - как записи журнала, так и страницы данных, измененные этой транзакцией, поэтому для транзакции T1 никаких операций по восстановлению не требуется.
- T2 - транзакция начата до принятия контрольной точки и успешно завершена после контрольной точки, но до наступления сбоя. Записи

журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память по завершению транзакции. Страницы данных, измененные этой транзакцией, только частично вытолкнуты во внешнюю память (на момент принятия контрольной точки). Для данной транзакции необходимо повторить заново те операции, которые были выполнены после принятия контрольной точки, т.е. выполнить их прямой прогон по журналу транзакций от момента принятия контрольной точки до фиксации.

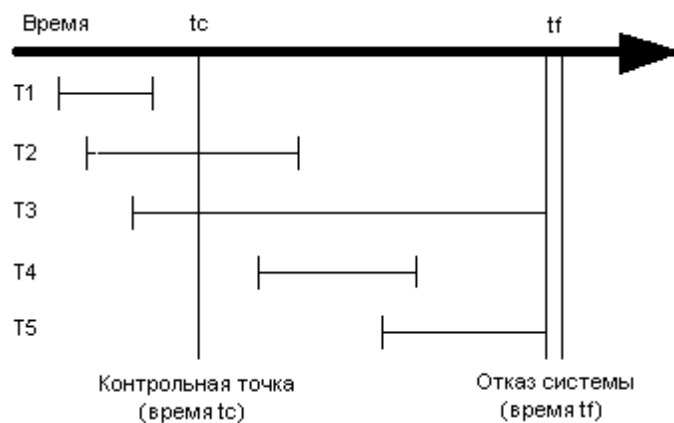


Рисунок 34. Варианты мягкого сбоя

- T3 - транзакция начата до принятия контрольной точки и не завершена в результате сбоя. Такую транзакцию необходимо откатить. Проблема, однако, в том, что часть страниц данных, измененных этой транзакцией, уже содержится во внешней памяти - это те страницы, которые были обновлены до принятия контрольной точки. Следов изменений, внесенных после контрольной точки в БД нет, т.к. записи журнала транзакций, сделанные до принятия контрольной точки, были вытолкнуты во внешнюю память, а записи журнала, которые были сделаны после контрольной точки, отсутствуют во внешней памяти журнала.
- T4 - транзакция начата после принятия контрольной точки и успешно

завершена до сбоя системы. Записи журнала транзакций, относящиеся к этой транзакции вытолкнуты во внешнюю память журнала. Изменения в БД, внесенные этой транзакцией, полностью отсутствуют во внешней памяти БД. Такую транзакцию необходимо повторить целиком (прогнать в прямом порядке по журналу транзакций).

- T5 - транзакция начата после принятия контрольной точки и не завершена в результате сбоя. Никаких следов этой транзакции нет ни во внешней памяти журнала транзакций, ни во внешней памяти БД. Для такой транзакции никаких действий предпринимать не нужно, ее как бы и не было вовсе, если о ней вспомнят, то ее запустят заново.

Восстановление системы после мягкого сбоя обычно осуществляется как часть процедуры перезагрузки системы. При перезагрузке системы транзакции T2 и T4 необходимо частично или полностью повторить, транзакцию T3 - частично откатить, для транзакций T1 и T5 никаких действий предпринимать не нужно. При перезагрузке система выполняет следующие действия:

- создается два списка транзакций UNDO (отменить) и REDO (повторить): в список UNDO заносятся все транзакции из последней записи контрольной точки (т.е. все транзакции, выполнявшиеся в момент принятия контрольной точки), список REDO остается пустым (для выше приведенного примера  $UNDO = \{T2, T3\}$ ,  $REDO = \{ \}$ ).
- начиная с записи контрольной точки, просматривается вперед журнал транзакций:
  - если в журнале транзакций обнаруживается запись о начале транзакции, то эта транзакция добавляется в список UNDO ( $UNDO = \{T2, T3, T4\}$ ,  $REDO = \{ \}$ );
  - если в файле регистрации обнаруживается запись COMMIT об окончании транзакции, то эта транзакция добавляется в список REDO ( $UNDO = \{T2, T3, T4\}$ ,  $REDO = \{T2, T4\}$ );
  - когда достигается конец журнала транзакций, оба списка анализируются, при этом из списка UNDO удаляются те транзакции,



которые попали в список REDO ( $UNDO = \{T3\}$ ,  $REDO = \{T2, T4\}$ );

- система просматривает журнал транзакций назад, начиная с момента контрольной точки и откатывая все транзакции из списка UNDO (для приведенного примера будут откатываться те операции транзакции T3, которые были выполнены до принятия контрольной точки);
- система просматривает журнал транзакций вперед, начиная с момента контрольной точки, и повторно выполняет все операции транзакций из списка REDO (для приведенного примера будут выполнены повторно все операции транзакции T4 и те операции транзакции T2, которые были выполнены после принятия контрольной точки).

### 3) Восстановление данных после жесткого сбоя.

При жестком сбое база данных на диске нарушается физически. Основой восстановления в этом случае является журнал транзакций и резервная копия БД. Восстановление данных при этом выполняется в следующем порядке:

- копирование БД из резервной копии на восстановленную систему;
- просмотр журнала транзакций для выявления всех транзакций, которые закончились успешно до наступления жесткого сбоя;
- по журналу транзакций в прямом направлении повторяются все успешно законченные транзакции (нет необходимости отката транзакций, прерванных в результате сбоя, т.к. изменения, внесенными этими транзакциями, отсутствуют после восстановления БД из резервной копии).

Наиболее плохим вариантом жесткого сбоя является ситуация, когда физически разрушены и БД, и журнал транзакций на диске. В этом случае единственное, что можно сделать - это восстановить состояние БД на момент последнего резервного копирования. Для того чтобы не допустить возникновения такой ситуации, саму БД и журнал транзакций обычно располагают на физически разных дисках, управляемых физически разными контроллерами.

## 7. ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ

В современном мире увеличение производительности обработки данных и знаний обычно достигается только в тех случаях, когда часть обработки выполняется с помощью компьютеров. Одним из вариантов достижения максимального прогресса в этой области, является создание и использование искусственного интеллекта, когда компьютер берет на себя не только однотипные, многократно повторяющиеся операции, но и решение сложноформализуемых (интеллектуальных) задач, которые до сих пор считаются прерогативой человека. Создание полноценного искусственного интеллекта открывает перед человечеством новые горизонты развития. В рамках создания подобных систем, основным предметом изучения являются мыслительные способности человека и способы их реализации техническими средствами.

### 7.1. Искусственный интеллект

Термин интеллект (*intelligence*) происходит от латинского *intellectus*, что означает ум, рассудок, разум; мыслительные способности человека. Обычно, под интеллектом понимается способность мозга решать задачи путем приобретения, запоминания и целенаправленного преобразования знаний в процессе обучения на опыте и адаптации к разнообразным обстоятельствам. Знания представляют собой информационную модель мира, в которой реальные объекты, их свойства и отношения между ними не только отображаются и запоминаются, но и могут мысленно целенаправленно преобразовываться, а формирование данной модели происходит в процессе обучения на опыте и адаптации к разнообразным обстоятельствам.

Термин искусственный интеллект (*artificial intelligence*) — ИИ (AI) можно рассматривать, как «умение рассуждать разумно», т.е. как свойство

компьютерных систем брать на себя отдельные функции интеллекта человека, например, выбирать и принимать оптимальные решения на основе ранее полученного опыта и рационального анализа внешних воздействий.

Решение любой задачи в рамках математики и кибернетики можно свести к двум основным этапам:

1) Нахождение алгоритма решения задачи (множества однотипных задач). Алгоритм (от имени узбекского математика Аль-Хорезми, который еще в IX веке предложил простейшие арифметические алгоритмы) – это точное предписание о выполнении в определенном порядке системы операций для решения любой задачи из некоторого данного класса (множества) задач. Поиск алгоритма для задач некоторого данного типа связан с тонкими и сложными рассуждениями, требующими большой изобретательности и высокой квалификации, поэтому принято считать, что подобного рода деятельность требует участия интеллекта человека, а задача отыскания алгоритма – интеллектуальная задача.

2) Решение задачи с использованием ранее найденного алгоритма. В данном случае процесс решения требует не рассуждений, а точного (машинального) выполнения инструкций алгоритма, т.е. не является интеллектуальным. Алгоритм могут в точности выполнить человек, вычислительная машина (должным образом запрограммированная) или робот, не имеющие ни малейшего представления о сущности самой задачи.

К неинтеллектуальным задачам можно отнести задачи, которые имеют стандартные методы (алгоритмы) решения, например, такие задачи, как решение системы линейных алгебраических уравнений, численное интегрирование дифференциальных уравнений и т.д.

К интеллектуальным задачам можно отнести задачи, в которых разбиение процесса поиска решения на отдельные элементарные шаги (этапы алгоритма) часто оказывается весьма затруднительным, даже если само их решение несложно. Примерами интеллектуальных задач являются распознавание образов, игры и управление в условиях неопределенности (шахматы),

планирование поведения, доказательство теорем и т.д.

Исходя из выше сказанного, интеллект можно определить как универсальный сверхалгоритм, который способен создавать алгоритмы решения конкретных задач. Деятельность мозга, направленная на решение интеллектуальных задач, называется мышлением, или интеллектуальной деятельностью. Характерными чертами интеллекта, проявляющимися в процессе решения задач, являются способность к обучению, обобщению, накоплению опыта (знаний и навыков) и адаптации к изменяющимся условиям в процессе решения задач. Благодаря этим качествам интеллекта мозг может решать разнообразные задачи, а также легко перестраиваться с решения одной задачи на другую. Таким образом, мозг, наделенный интеллектом, является универсальным средством решения широкого круга задач (в том числе неформализованных) для которых нет стандартных, заранее известных методов решения.

Существуют и другие, чисто поведенческие (функциональные) определения интеллекта:

- любая материальная система, с которой можно достаточно долго обсуждать проблемы науки, литературы и искусства, обладает интеллектом (А.Н. Колмогоров);
- определение А. Тьюринга – если в разных комнатах находятся люди и машина (причем они не могут видеть друг друга, но имеют возможность обмениваться информацией), тогда, если в процессе диалога между участниками игры людям не удастся установить, что один из участников — машина, то такую машину можно считать обладающей интеллектом.

Философские проблемы, связанные с ИИ:

1) Возможно ли создание ИИ в принципе?

История развития человечества знает попытки ответа на этот вопрос с древнейших времен и до наших дней: описание человекоподобных существ-автоматов (например, «Илиада» Гомера, сказки), идея создания механических машин для решения задач (Р. Луллий, средние века), универсальные языки

классификации наук (Лейбниц и Декарт, 18-й век), создание кибернетики (Н. Виннер), изучение "самовоспроизводящихся автоматов" (Дж. фон Нейман, 20-й век), написание компьютерных вирусов.

Принципиальная возможность автоматизации решения интеллектуальных задач с помощью ЭВМ обеспечивается свойством алгоритмической универсальности. Алгоритмическая универсальность ЭВМ означает, что на них можно программно реализовывать (т. е. представить в виде машинной программы) любые алгоритмы преобразования информации: вычислительные алгоритмы, алгоритмы управления, алгоритмы поиска доказательства теорем, алгоритмы составления мелодий и т.п. При этом процессы, порождаемые этими алгоритмами, являются потенциально осуществимыми, т.е. они осуществимы в результате конечного числа элементарных операций. Практическая осуществимость алгоритмов зависит от имеющихся технических средств, которые могут меняться с развитием техники (например, появление быстродействующих ЭВМ сделало практически осуществимыми такие алгоритмы, которые ранее были только потенциально осуществимыми (шахматы)). Однако свойство алгоритмической универсальности не ограничивается констатацией того, что для всех известных алгоритмов оказывается возможной их программная реализация на ЭВМ. Содержание этого свойства имеет и характер прогноза на будущее: всякий раз, когда в будущем какое-либо предписание будет признано алгоритмом, то независимо от того, в какой форме и какими средствами это предписание будет первоначально выражено, его можно будет задать также в виде машинной программы.

Не следует думать, что вычислительные машины и роботы могут в принципе решать любые задачи. Было строго доказано существование таких типов задач, для которых невозможен единый эффективный алгоритм, решающий все задачи данного типа; в этом смысле невозможно решение задач такого типа и с помощью вычислительных машин. Утверждение об алгоритмической неразрешимости некоторого класса задач является не просто признанием того, что такой алгоритм пока не известен и будет найден в

будущем, а представляет собой одновременно и прогноз что подобного рода алгоритм просто не существует. При решении алгоритмически неразрешимых задач человеком могут быть использованы следующие подходы:

- игнорирование (обход) задачи;
- сужение условий универсальности задачи, т.е. решение задачи только для определенного подмножества начальных условий;
- метод "научного тыка", т.е. случайное расширение множества доступных элементарных операций и направления решения.

2) Цель создания ИИ (нужен ли вообще человечеству ИИ, для чего и к чему это приведет)?

Наиболее вероятная цель создания ИИ – создание хранителя знаний и советника по решению интеллектуальных задач. Проводя аналогию с организацией знаний в современном мире – нет одного человека, который может быстро, точно и самостоятельно решить абсолютно любую задачу, но есть ряд узких специалистов, которые в состоянии решать задачи в своей частной области знаний, и их советы позволяют одному человеку принять правильный выбор из множества вариантов решения (повышение производительности, усилитель интеллекта). Исходя из такого прогноза развития ИИ, роль человека в будущем – решение более сложных интеллектуальных задач, на основе данных от ИИ-советников (хотя это тоже можно рассматривать как своего рода зависимость). Человек уже давно пользуется предметами, усиливающими его физические характеристики (например, средствами передвижения, орудиями труда, системами биозащиты и т.п.), поэтому в принципе возможно использование усилителей интеллекта. Примеры ИИ-советников, используемых в настоящее время: прогноз погоды, пилотируемые полеты, САПР.

3) Проблема безопасности ИИ для человека (будут ли у ИИ свои воля, желания и поступки, в том числе и такие, которые могут противоречить

человеку)?

Данная проблема также рассматривается человечеством с древних времен, поэтому можно сказать, что проблема безопасности и вопрос возможности создания ИИ неразрывно связаны. Особенно большой вклад в обсуждение этой проблемы внесли писатели-фантасты (К. Чапек (создатель термина «робот»), А. Азимов), которые в своих произведениях, кроме описания угроз, предлагают и решение проблемы безопасности. Например, попытка обобщения решения проблем безопасности в так называемых законах роботехники:

1) робот не может причинить вред человеку или своим бездействием допустить, чтобы человеку был причинен вред;

2) робот должен повиноваться командам, которые ему дает человек, кроме тех случаев, когда эти команды противоречат первому закону;

3) робот должен заботиться о своей безопасности, насколько это не противоречит первому и второму закону.

Эти размышления, в свою очередь приводят к возникновению новых вопросов:

- как ввести эти общие правила в ИИ, т.е. однозначно определить и запрограммировать подобные правила?
- сможет ли ИИ понять эти правила в нужных рамках (то, что человек считает как само разумеющееся), а не определить свою цель исходя из только логических рассуждений;
- как будет действовать ИИ при возникновении противоречий (например, если спасение жизни одного человека возможно только за счет потери другого).

Несмотря на перечисленные проблемы, данные законы являются довольно неплохим неформальным базисом проверки надежности системы безопасности для систем ИИ.

Один из интересных вариантов обеспечения безопасности ИИ – полное подчинение ИИ человеку-хозяину, т.е. фактически сделать конкретного

индивидуума ответственным за обучение и использование своего ИИ (что-то типа связки человек-собака, человек-автомобиль и т.п.). Также в плане обеспечения безопасности важен не тотальный запрет исследований в области ИИ, а контроль исследований со стороны всего человечества, т.к. неконтролируемый процесс создания ИИ можно сравнить с процессом создания оружия массового уничтожения.

В плане создания ИИ интересен подход, предложенный А. Тьюрингом: "Пытаясь имитировать интеллект взрослого человека, мы вынуждены много размышлять о том процессе, в результате которого человеческий мозг достиг своего настоящего состояния... Почему бы нам вместо того, чтобы пытаться создать программу, имитирующую интеллект взрослого человека, не попытаться создать программу, которая имитировала бы интеллект ребенка? Ведь если интеллект ребенка получает соответствующее воспитание, он становится интеллектом взрослого человека... Наш расчет состоит в том, что устройство, ему подобное, может быть легко запрограммировано... Таким образом, мы расчленим нашу проблему на две части: на задачу построения "программы-ребенка" и задачу "воспитания" этой программы". Данный подход используют практически все системы ИИ, т.к. невозможно практически заложить сразу все знания в такую сложную систему, а возможность обучения делает возможным выполнять усложнение системы постепенно, кроме того, только на этом пути проявятся перечисленные выше признаки интеллектуальной деятельности (накопление опыта, адаптация и т.п.).

## 7.2. Этапы и направления развития интеллектуальных систем

Исторически сложились два основных направления в ИИ:

1) Нейрокибернетика.

Нейрокибернетика ориентирована на программно-аппаратное моделирование структур, подобных структуре человеческого мозга. Основная



идея этого направления – единственный объект, способный мыслить, это человеческий мозг, поэтому любое «мыслящее» устройство должно каким-то образом воспроизводить его структуру. В рамках данного подхода объектом исследований являются структура и механизмы работы мозга человека, а конечная цель заключается в раскрытии тайн мышления. Необходимыми этапами исследований в этом направлении являются построение моделей на основе психофизиологических данных, проведение экспериментов с ними, выдвижение новых гипотез относительно механизмов интеллектуальной деятельности, совершенствование моделей и т. д.

Нервная система человека построена по иерархическому принципу, когда задача распределяется между несколькими уровнями. Высший уровень нервной системы (связанный с большими полушариями мозга) ставит лишь общую задачу, скажем, переложить книгу на стол. Этот уровень вообще не контролирует действие отдельных двигательных единиц, направленных на решение поставленной задачи. Детализация построения движений у человека происходит на уровнях более низких, чем командный уровень коры больших полушарий. Более того, в некоторых случаях (когда мы отдергиваем руку, прикоснувшись к горячему предмету, даже не успев осознать ситуацию) все управление формируется на нижележащих уровнях, связанных с различными отделами спинного мозга.

Для обработки только зрительной информации человеческий мозг использует достаточно сложную схему. Вначале свет попадает в глаз. Пройдя через всю оптическую систему глаза, фотоны попадают на сетчатку — слой светочувствительных клеток — палочек и колбочек. На этом уровне происходит первый этап обработки информации, поскольку, сразу за светочувствительными клетками находится обычно несколько слоев нервных клеток, которые выполняют сравнительно несложную обработку. Затем информация поступает по зрительному нерву в головной мозг человека, в так называемые "зрительные бугры". Далее зрительная информация поступает в отделы мозга, которые уже выделяют из нее отдельные составляющие —

горизонтальные, вертикальные, диагональные линии, контуры, области светлого, темного, цветного. На выше описанных этапах можно без труда смоделировать работу мозга, применяя различные графические фильтры. Постепенно образы становятся все более сложными и размытыми, но графический образ картины пройдет еще долгий путь, прежде чем достигнет уровня сознания. Причем на уровне сознания у нас будет не только зрительный образ, к нему примешаются, например, еще и звуки, запахи и вкусовые ощущения.

Физиологами давно установлено, что основной структурой человеческого мозга является большое количество ( $\sim 10^{21}$ ) связанных между собой и взаимодействующих нервных клеток – нейронов. Поэтому усилия нейрокибернетики сосредоточены на создании подобных элементов – нейронных сетей.

Первые нейросети были созданы в середине 1950-х гг., когда американский физиолог Ф. Розенблатт предложил модель зрительного восприятия и распознавания — перцептрон (perceptron). Перцептрон работает в двух режимах: в режиме обучения и в режиме распознавания.

В наиболее простом виде перцептрон (см. рисунок 35) состоит из совокупности чувствительных (сенсорных) элементов (S-элементов), на которые поступают входные сигналы. S-элементы случайным образом связаны с совокупностью ассоциативных элементов (A-элементов), выход которых отличается от нуля только тогда, когда возбуждено достаточно большое число S-элементов, воздействующих на один A-элемент. A-элементы соединены с реагирующими элементами (R-элементами) связями, коэффициенты усиления ( $v$ ) которых переменны и изменяются в процессе обучения. Взвешенные комбинации выходов R-элементов составляют реакцию системы, которая указывает на принадлежность распознаваемого объекта определенному образу. В режиме распознавания машине предъявляются некоторые объекты (обычно отличные от тех, что использовались для обучения), и она должна их классифицировать, по возможности, правильно. Если распознаются только два

образа, то в персептроне устанавливается только один R-элемент, который обладает двумя реакциями — положительной и отрицательной. Если образов больше двух, то для каждого образа устанавливают свой R-элемент, а выход каждого такого элемента представляет линейную комбинацию выходов A-элементов:

$$R_j = \Theta_j + \sum_{i=1}^n v_{ij} x_i,$$

где  $R_j$  — реакция j-го R-элемента;  $x_i$  — реакция i-го A-элемента;  $v_{ij}$  — вес связи от i-го A-элемента к j-му R элементу;  $\Theta_j$  — порог j-го R-элемента.

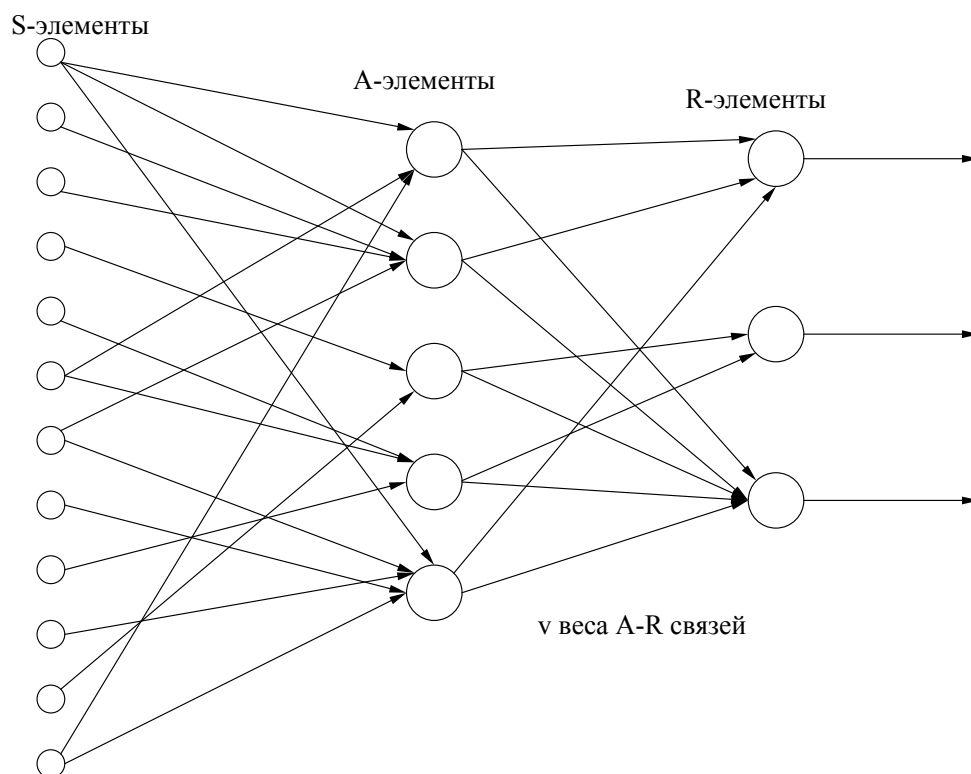


Рисунок 35. Персептрон.

Аналогично записывается уравнение i-го A-элемента:

$$x_i = \Theta_i + \sum_{k=1}^S y_k.$$

Здесь сигнал  $y_k$  может быть непрерывным, но чаще всего он принимает только два значения: 0 или 1. Сигналы от S-элементов подаются на входы A-

элементов с постоянными весами равными единице, но каждый А-элемент связан только с группой случайно выбранных S-элементов.

В режиме обучения некто (человек, машина, робот или природа), играющий роль учителя, предъявляет машине объекты и о каждом из них сообщает, к какому понятию (классу) он принадлежит. По этим данным строится решающее правило, являющееся, по существу, формальным описанием понятий. В процессе обучения изменяются веса  $v_i$  А-элементов. В частности, если применяется система подкрепления с коррекцией ошибок, прежде всего, учитывается правильность решения, принимаемого перцептроном. Если решение правильно, то веса связей всех сработавших А-элементов, ведущих к R-элементу, выдавшему правильное решение, увеличиваются, а веса неиспользованных А-элементов остаются неизменными. Можно оставлять неизменными веса использованных А-элементов, но уменьшать веса неиспользованных. В некоторых случаях веса сработавших связей увеличивают, а неиспользованных — уменьшают. После процесса обучения перцептрон сам, без учителя, начинает классифицировать новые объекты.

Появление машины, способной обучаться понятиям и распознавать предъявляемые объекты, оказалось чрезвычайно интересным не только физиологам, но и представителям других областей знания и породило большой поток теоретических и экспериментальных исследований (1960-е – 1980-е гг.). На сегодняшний день, кроме модели перцептронов, предложено большое число моделей нейросетей, например, модель нейронной сети с обратным распространением ошибки (back propagation), сети Хопфилда и Хэмминга и т.д.

Следующим толчком к развитию нейросетей послужила разработка транспьютеров – компьютеров с большим количеством процессорных элементов, обрабатывающих данные параллельно (1980-е, Япония).

На данный момент можно выделить три подхода к созданию нейросетей:

- аппаратный – создание специальных компьютеров и нейрочипов, моделирующих сети нейронов;

- программный – моделирование нейросетей на универсальных ЭВМ;
- гибридный – совмещение первых двух подходов.

## 2) «Кибернетика черного ящика».

В основу этого подхода положен следующий принцип – не имеет значения как устроено «мыслящее» устройство, главное, чтобы оно реагировало на заданные входные воздействия как человеческий мозг. Изобретения человека не всегда слепо копируют природу, например, колесо и ЭВМ в нынешнем виде не существуют в природе, но успешны и эффективны в применении. Кроме того, даже значительные успехи физиологии (и нейрокибернетики) не позволяют на данный момент понять, как протекают интеллектуальные процессы у человека, и это заставляет искать иные пути решения задачи ИИ, чем моделирование мозга. Фактически целью работ в этом направлении является создание алгоритмического и программного обеспечения вычислительных машин, позволяющего решать интеллектуальные задачи не хуже человека.

В этом направлении можно выделить следующие этапы развития:

- конец 1950-х гг. – исследование модели лабиринтного поиска; в данном подходе задача представляется как некоторое пространство состояний в виде графа, и в этом графе производится поиск оптимального пути от входных данных к результирующим; самыми первыми интеллектуальными задачами, которые стали решаться при помощи ЭВМ, были логические игры (шашки, шахматы), движение в лабиринтах (игрушки типа "электронной мыши" Клода Шеннона);
- начало 1960-х гг. – исследования в области эвристического программирования – программирования основанного на использовании эвристик или правил, теоретически необоснованных, но позволяющих упростить решение или сократить число переборов в пространстве поиска;
- конец 1960 гг. – к решению задач ИИ подключают методы

математической логики, в частности были предложены метод резолюций (Робинсон) и обратный вывод (Ю.С. Маслов), создан язык Пролог и программы доказательства теорем (однако большинство практических задач не сводятся к набору аксиом и только к классической логике);

- 1973 г. – «доклад Лайтхилла», обзор состояния разработок ИИ по заказу Британского совета научных исследований; уровень разработок определен как разочаровывающий, снижение финансирования программ исследований ИИ;
- середина 1970 гг. – на смену поиска универсального алгоритма мышления пришла идея моделирования конкретных знаний специалистов-экспертов; в США появились первые коммерческие экспертные системы (MYCIN в области медицины и DENDRAL – в области химии), заинтересованность Пентагона в создании программ на принципах ИИ;
- начало 1980-х гг. – проект вычислительных машин V поколения (Япония), расширение группы специалистов в области ИИ;
- начало 1990-х – коммерциализация ИИ, создание промышленных экспертных систем.

На сегодня эти два выше указанных направления фактически смешались между собой. Также можно выделить особое направление, ориентированное на создание смешанных интерактивных интеллектуальных систем, основанных на симбиозе возможностей естественного и искусственного интеллекта.

Основными задачами исследований в области ИИ на сегодня считаются:

1) Представление знаний и разработка систем, основанных на знаниях (knowledge-based systems) – разработка моделей представления знаний, создание баз знаний.

2) Программное обеспечение систем ИИ (software engineering for AI) – разработка языков для решения интеллектуальных задач (LISP, PROLOG, SMALLTALK, РЕФАЛ и др.), а также создание специальных пакетов прикладных программ (например, KEE, ARTS, G2) и оболочек экспертных

систем (например, KAPPA, EXSYS, M1, ЭКО)

3) Разработка естественно-языковых интерфейсов и машинный перевод (natural language processing) - при достаточно формальной обработке и классификации основных грамматических правил и приемов пользования словарем можно создать вполне удовлетворительный алгоритм для перевода, например, научного или делового текста. Для некоторых языков такие системы были созданы еще в конце 60-г. Однако для того, чтобы связно перевести достаточно большой разговорный текст, необходимо понимать его смысл. Работы над такими программами ведутся уже давно, но до полного успеха еще далеко. Имеются также программы, обеспечивающие диалог между человеком и машиной на урезанном естественном языке. Анализ текста разбивается на ряд подзадач:

- морфологический анализ – анализ слов в тексте;
- синтаксический анализ – разбор состава предложений и грамматических связей между словами;
- семантический анализ – анализ смысла частей предложения на основе некоторых знаний о предметной области текста;
- прагматический анализ – анализ смысла предложений.

4) Интеллектуальные роботы (robotics) - вся интеллектуальная деятельность человека направлена, в конечном счете, на активное взаимодействие с внешним миром посредством движений (русский физиолог И. М. Сеченов: "... все бесконечное разнообразие внешних проявлений мозговой деятельности сводится окончательно лишь к одному явлению — мышечному движению"). Точно так же элементы интеллекта робота служат, прежде всего, для организации его целенаправленных движений. В этом состоит отличие роботов от чисто компьютерных систем ИИ, которые решают интеллектуальные задачи, носящие абстрактный или вспомогательный характер и обычно не связаны ни с восприятием окружающей среды с помощью искусственных органов чувств, ни с организацией движений исполнительных механизмов. Можно выделить три поколения роботов:

- роботы с жесткой схемой управления (программируемые манипуляторы) – большинство промышленных роботов;
- адаптивные роботы с сенсорными устройствами – малая часть промышленных роботов, высокая стоимость;
- самоорганизующиеся или интеллектуальные роботы (проблемы машинного зрения, обработки и хранения трехмерной информации) – цель развития робототехники.

5) Обучение и самообучение (machine learning) – методы и алгоритмы, ориентированные на автоматическое накопление и формирование знаний на основе анализа и обобщения данных.

6) Распознавание образов (pattern recognition) – описание классов объектов через определенные значения значимых признаков и вывод о наиболее подходящем классе для введенного объекта; на данный момент это фактически отдельная наука.

7) Новые архитектуры компьютеров (new hardware platform and architectures) – разработка спецпроцессоров, основанных на архитектуре отличной от фон Неймановской (например, Пролог- и Лисп-машины).

8) Игры и машинное творчество – традиционно в этой области представлены программы, реализующие интеллектуальные игры (например, шахматы), также в настоящее время существуют и успешно применяются программы (в том числе и самообучаемые), позволяющие машинам играть в деловые или военные игры, имеющие большое прикладное значение. Машинное творчество охватывает сочинение компьютером музыки, стихов, рассказов и т.п.

### 7.3. Экспертные системы

Экспертная система - это программное средство, использующее экспертные знания для обеспечения высокоэффективного решения



неформализованных задач в узкой предметной области, причем эти результаты не должны уступать по качеству и эффективности решениям, получаемым человеком-экспертом. Следует отметить, что технология ЭС существенно расширяет круг практически значимых задач, решаемых на компьютерах, решение которых приносит значительный экономический эффект.

Общими свойствами таких неформализованных задач являются следующие:

- неполнота, неоднозначность, противоречивость и динамичность исходных данных и знаний о решаемой задаче (другими словами, эти задачи недостаточно хорошо понимаются или изучены на данный момент);
- отсутствие точного алгоритмического решения задачи или невозможность его использования из-за ограниченности ресурсов и времени вычислений (большая размерность пространства решения);
- возможность поиска решения для этих задач с помощью механизма символических рассуждений.

ЭС не отвергают и не заменяют традиционного процедурного подхода к разработке программ, ориентированного на решение формализованных задач, т.к. в настоящее время использование ЭС для решения формализованных задач (т.е. задач имеющих точно определенный алгоритм решения, например, поиск максимального значения в списке) очень неэффективно.

Основу ЭС составляет база знаний (БЗ) о предметной области, которая накапливается в процессе построения и эксплуатации ЭС. Накопление и организация знаний – основа построения ЭС. Отличие ЭС от традиционных программ и определяет их основные свойства:

- использование символического способа представления и вывода знаний (символ — это строка знаков, соответствующая содержанию некоторого понятия, символы объединяют, чтобы выразить отношения между ними), а также использование механизма автоматического рассуждения и эвристического поиска решения (исключение лишних вычислений, а не

последовательный перебор всех вариантов);

- сложность решаемых задач и использование либо сложных конструкций правил, либо большого их количества;
- применение для решения проблем высококачественного опыта, который представляет уровень мышления наиболее квалифицированных экспертов в данной (достаточно узкой) области, что ведёт к творческим, точным и эффективным решениям;
- наличие прогностических возможностей, при которых ЭС выдаёт ответы не только для конкретной ситуации, но и показывает, как изменяются эти ответы в новых ситуациях, с возможностью подробного объяснения каким образом новая ситуация привела к изменениям;
- обеспечение такого нового качества, как институциональная память, за счёт входящей в состав ЭС базы знаний, которая разработана в ходе взаимодействий со специалистами (экспертами) организации, и представляет собой совокупный опыт этой группы людей;
- возможность использования ЭС для обучения и тренировки персонала, обеспечивая новых сотрудников обширным багажом опыта и стратегий (т.к. кроме выдачи ответов ЭС должна уметь пояснить свои действия и проверить их правильность);
- ЭС могут ошибаться, т.к. они разработаны так, чтобы вести себя как человек-эксперт, а также ЭС, подобно людям, имеют потенциальную возможность учиться на своих ошибках.

Преимуществами использования ЭС являются:

- постоянство – компетентность человека-эксперта ослабевает со временем, а перерыв в его профессиональной деятельности может серьёзно отразиться на его профессиональных качествах;
- лёгкость передачи или воспроизведения - передача знаний от одного человека другому это долгий и дорогой процесс, а передача информации ЭС представляет собой простой процесс копирования программы или файла данных;

- устойчивость и воспроизводимость результатов – человек-эксперт может принимать в похожих ситуациях разные решения из-за эмоциональных факторов, а результаты ЭС в данном случае будут стабильны;
- стоимость – человек-эксперт обходится очень дорого, а ЭС, наоборот, сравнительно недорого (их разработка действительно дорога, но они дешёвы в эксплуатации).

Вместе с тем даже разработка ЭС не позволяет полностью отказаться от эксперта-человека, т.к. человеческая компетенция явно превосходит искусственную. Однако в ряде ситуаций возможен симбиоз ЭС и человека-эксперта средней квалификации (при этом ЭС используется для усиления и расширения его профессиональных возможностей человека).

К недостаткам ЭС по отношению к человеку-эксперту можно отнести слабость позиций в следующих направлениях:

- здравый смысл - в дополнение к широкому техническому знанию, человек-эксперт имеет здравый смысл;
- творческий потенциал - человек-эксперт может реагировать творчески на необычные ситуации;
- обучение - человек-эксперт автоматически адаптируется к изменению среды;
- узкие рамки - ЭС не работают, если решения не существует или когда проблема лежит вне области их компетенции.

Состав участников построения и эксплуатации ЭС следующий:

- эксперт - человек, способный ясно выражать свои мысли и пользующийся репутацией специалиста, умеющего находить правильные решения проблем в конкретной предметной области; эксперт использует свои приёмы и ухищрения, чтобы сделать поиск решения более эффективным, и ЭС моделирует все его стратегии;
- инженер знаний — человек, как правило, имеющий познания в информатике и в области ИИ и знающий, как надо строить ЭС; инженер знаний опрашивает экспертов, организует знания, решает, каким образом

они должны быть представлены в ЭС, и может исправлять программный код ЭС; отсутствие среди участников разработки инженеров по знаниям (т.е. замена их программистами) либо приводит к неудаче процесс создания ЭС, либо значительно удлиняет его; если проводить аналогию с СБД, то инженер знаний соответствует администратору БД.

- программист по разработке инструментальных средств (оболочки ЭС) – человек, который обладает квалификацией для построения набора программных средств ЭС; средство построения ЭС — это программное средство, используемое инженером знаний или программистом для построения ЭС, этот инструмент отличается от обычных языков программирования тем, что обеспечивает удобные способы представления сложных высокоуровневых понятий;
- клерк – человек, который может пополнять ЭС данными (например, статистикой продаж билетов);
- конечный пользователь — это человек, не относящийся к выше перечисленным группам, который использует уже построенную ЭС для решения своих задач.

Как уже отмечалось выше, основой любой ЭС является совокупность знаний, структурированная в целях упрощения процесса принятия решения. Эта информация принимает форму фактов и правил. Факты и правила в ЭС не всегда либо истинны, либо ложные. Иногда существует некоторая степень неуверенности в достоверности факта или точности правила. Если это сомнение выражено явно, то оно называется "коэффициентом доверия". Коэффициент доверия — это число, которое означает вероятность или степень уверенности, с которой можно считать данный факт или правило достоверным или справедливым. Многие правила ЭС являются эвристиками, то есть эмпирическими правилами или упрощениями, которые эффективно ограничивают поиск решения. Если алгоритмический метод гарантирует корректное или оптимальное решение задачи, то, в свою очередь, эвристический метод даёт приемлемое решение в большинстве случаев.

ЭС по своему способу построения и функционирования можно разделить на статические и динамические ЭС.

Статические ЭС используются в тех приложениях, где можно не учитывать изменения окружающего мира за время решения задачи. Структура статической ЭС состоит из следующих основных блоков (см. рисунок 36):

- база знаний (БЗ) - содержит факты (данные) и правила (или другие представления знаний), использующие эти факты как основу для принятия решений;
- база данных (БД) – содержит данные (обычно набор констант и статистики) необходимые для функционирования механизма логических выводов;
- механизм логических выводов (МЛВ) - модуль, который на основании имеющихся в БЗ знаний способен делать логические выводы; МЛВ состоит из интерпретатора, определяющего как применять правила для вывода новых знаний на основе информации, хранящейся в БЗ, и диспетчера, устанавливающего порядок применения этих правил;
- модуль приобретения знаний (МПЗ) – механизм получения знаний от эксперта, поддержки БЗ и при необходимости ее дополнения;
- модуль советов и объяснений (МСО) - механизм, который не только способен давать заключения, но и представлять различные комментарии к этому заключению и объяснять его мотивы;
- пользовательский интерфейс (ПИ) - интерфейс для получения и модификации знаний эксперта, а также для правильной передачи ответов пользователю;

Следует особо подчеркнуть важность модуля советов и объяснений в составе ЭС:

- без него пользователю трудно будет понять заключение, полученное при консультации или решении какого-либо вопроса;
- этот механизм важен для эксперта, он позволяет определить, как работает система и выяснить, как используются предоставленные им знания.

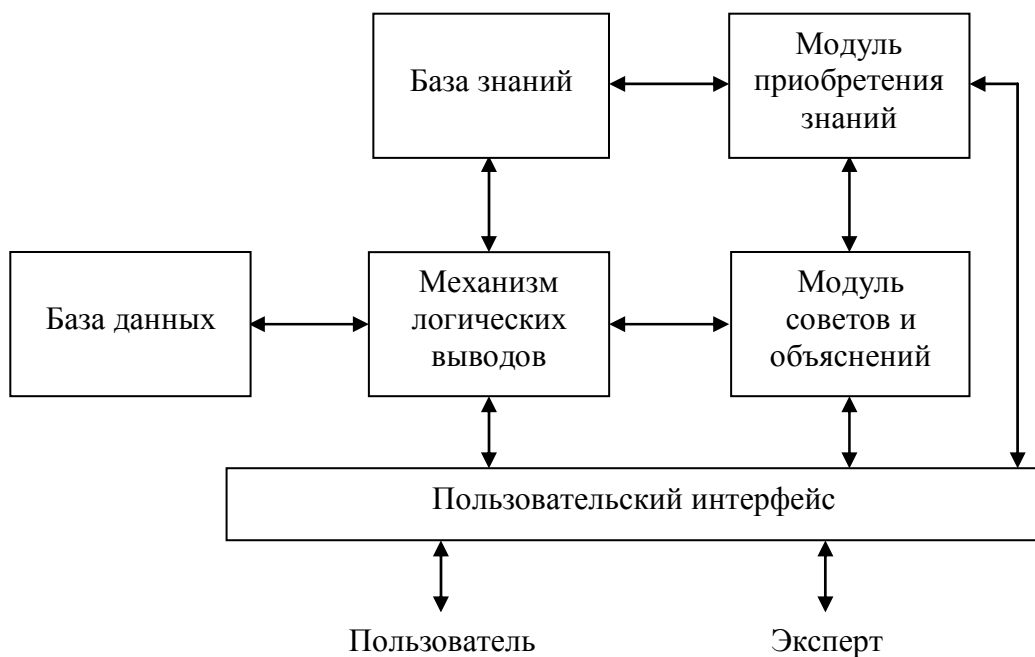


Рисунок 36. Структура статической ЭС.

Основной момент в организации ЭС – ответ на два вопроса:

1) Какие знания следует представлять (использовать) в ЭС?

Состав знаний ЭС определяется следующими факторами:

а) Проблемной средой.

В соответствии с общей схемой статической ЭС для ее функционирования требуются следующие знания:

- знания о процессе решения задачи (управляющие знания), используемые МЛВ;
- знания о языке общения и способах организации диалога, используемые ПИ;
- знания о способах представления и модификации знаний, используемые МПЗ;
- поддерживающие структурные и управляющие знания, используемые МСО.

б) Архитектурой ЭС.

С учетом архитектуры ЭС знания целесообразно делить на следующие классы: интерпретируемые и неинтерпретируемые знания.

Интерпретируемые знания – это такие знания, которые способен интерпретировать МЛВ. Интерпретируемые знания можно разделить на:

- предметные знания - содержат данные о предметной области и способах преобразования этих данных при решении поставленных задач; в этих знаниях можно выделить описатели (содержат определенную информацию о предметных знаниях, такую, как коэффициент определенности правил и данных, меры важности и сложности) и собственно предметные знания, которые делят на факты (определяют возможные значения сущностей и характеристик предметной области) и исполняемые утверждения (содержат информацию о том, как можно изменять описание предметной области в ходе решения задач);
- управляющие знания – делятся на фокусирующие (описывают, какие знания следует использовать в той или иной ситуации; содержат сведения о наиболее перспективных объектах или правилах, которые целесообразно использовать при проверке соответствующих гипотез) и решающие (содержат информацию, используемую для выбора способа интерпретации знаний, подходящего к текущей ситуации; применяются для выбора стратегий или эвристик, наиболее эффективных для решения данной задачи);
- знания о представлении - содержат информацию о том, каким образом (в каких структурах) в системе представлены интерпретируемые знания.

По отношению к предметным знаниям знания о представлении и знания об управлении являются метазнаниями. Качественные и количественные показатели экспертной системы могут быть значительно улучшены за счет использования метазнаний, т.е. знаний о знаниях. Метазнания не представляют некоторую единую сущность, они могут применяться для достижения

различных целей:

- метазнания в виде стратегических метаправил используются для выбора релевантных правил;
- метазнания используются для обоснования целесообразности применения правил из области экспертизы;
- метазнания используются для обнаружения синтаксических и семантических ошибок в предметных правилах;
- метазнания позволяют системе адаптироваться к окружению путем перестройки предметных правил и функций;
- метазнания позволяют явно указать возможности и ограничения системы, т.е. определить, что система знает, а что не знает.

Неинтерпретируемые знания – знания, для которых МЛВ не может интерпретировать их структуру и содержание. Неинтерпретируемые знания подразделяются на:

- вспомогательные знания - хранят информацию о лексике и грамматике языка общения, информацию о структуре диалога; эти знания обрабатываются естественно-языковой компонентой, но ход этой обработки МЛВ не осознает, так как этот этап обработки входных сообщений является вспомогательным для работы ЭС;
- поддерживающие знания – используются при создании системы и при выполнении объяснений; они выполняют роль описаний (обоснований) как интерпретируемых знаний, так и действий системы, и подразделяются на технологические (содержат сведения о времени создания описываемых ими знаний, об авторе знаний и т.п.) и семантические (содержат смысловое описание знаний, например, о причинах ввода знаний, о назначении знаний, описывают способ использования знаний и получаемый эффект) знания.

в) Потребностями и целями пользователей.

Зависимость состава знаний от требований пользователя проявляется в



следующем:

- какие задачи (из общего набора задач) и с какими данными хочет решать пользователь;
- каковы предпочтительные способы и методы решения;
- при каких ограничениях на количество результатов и способы их получения должна быть решена задача;
- каковы требования к языку общения и организации диалога;
- какова степень общности (конкретности) знаний о проблемной области, доступная пользователю;
- каковы цели пользователей.

г) Языком общения.

Состав знаний о языке общения зависит как от языка общения, так и от требуемого уровня понимания.

2) Каким образом знания следует представлять в ЭС?

Вопросы организации знаний обычно рассматриваются вне выбранной модели знаний. Можно выделить следующие проблемы организации знаний:

а) Организация знаний по уровням представления и по уровням детальности.

Для того чтобы ЭС могла управлять процессом поиска решения, приобретать новые знания и объяснять свои действия, она должна уметь не только использовать свои знания, но и обладать способностью понимать и исследовать их (т.е. экспертная система должна иметь знания о том, как представлены ее знания о проблемной среде). Все знания можно разделить на уровни:

- уровень 0 - знания о проблемной среде;
- уровень 1 - метазнания, т.е. знания о том, как представлены в модели системы знания об уровне 0 и какие средства используются для представления знаний уровня 0; метазнания играют существенную роль

при управлении процессом решения, при приобретении и объяснении действий системы; знания уровня 1 не содержат ссылок на знания уровня 0 (т.е. метазнания независимы от проблемной среды);

- уровень 2 - содержит сведения о знаниях уровня 1, т.е. знания о представлении базовых понятий метазнаний.

Число уровней представления может быть больше приведенного количества. Разделение знаний по уровням представления обеспечивает расширение области применимости системы. Выделение уровней детальности позволяет рассматривать знания с различной степенью подробности. Количество уровней детальности во многом определяется спецификой решаемых задач, объемом знаний и способом их представления. Как правило, выделяется не менее трех уровней детальности, отражающих соответственно общую, логическую и физическую организацию знаний. Введение нескольких уровней детальности обеспечивает дополнительную степень гибкости системы, так как позволяет производить изменения на одном уровне, не затрагивая другие. Изменения на одном уровне детальности могут приводить к дополнительным изменениям на этом же уровне, что оказывается необходимым для обеспечения согласованности структур данных и программ. Однако наличие различных уровней препятствует распространению изменений с одного уровня на другие.

#### б) Организация знаний в рабочей памяти МЛВ.

Рабочая память МЛВ ЭС предназначена для хранения данных обработки знаний. Данные в рабочей памяти могут быть структурированы по следующим позициям:

- данные однородны или разделяются на уровни по типам данных (на каждом уровне рабочей памяти хранятся данные соответствующего типа; выделение уровней усложняет структуру ЭС, но делает систему более эффективной, например, можно выделить уровень планов, уровень агенды (упорядоченного списка правил, готовых к выполнению) и

уровень данных предметной области (уровень решений));

- данные изолированные (рабочая память состоит из множества независимых простых элементов) или связанные (рабочая память состоит из нескольких (при нескольких уровнях) сложных элементов (например, объектов), один сложный элемент соответствует множеству простых, объединенных в единую сущность; связь между отдельными объектами указывается явно, например заданием семантических отношений; для ускорения поиска и сопоставления, данные в рабочей памяти могут быть связаны не только логически, но и ассоциативно);
- данные являются переменными (например, характеристики некоторого объекта) и константами (например, значения этих характеристик);

в) Организация знаний в БЗ.

Показателем интеллектуальности системы с точки зрения представления знаний считается способность системы использовать в нужный момент необходимые (релевантные) знания. Системы, не имеющие средств для определения релевантных знаний, неизбежно сталкиваются с проблемой "комбинаторного взрыва". В проблеме доступа к знаниям можно выделить три аспекта:

- связность знаний и данных - основной способ, обеспечивающий ускорение поиска релевантных знаний; считается, что знания следует организовывать вокруг наиболее важных объектов (сущностей) предметной области - все знания, характеризующие некоторую сущность, связываются и представляются в виде отдельного объекта, при подобной организации знаний, если системе потребовалась информация о некоторой сущности, то она ищет объект, описывающий эту сущность, а затем уже внутри объекта отыскивает информацию о данной сущности; в объектах выделяют два типа связей между элементами: внешние (отражают взаимозависимости, существующие между объектами в области экспертизы; делятся на логические (выражают семантические

отношения между элементами знаний) и ассоциативные (предназначены для обеспечения взаимосвязей, способствующих ускорению процесса поиска релевантных знаний)) и внутренние (объединяют элементы в единый объект и предназначены для выражения структуры объекта);

- механизм доступа к знаниям - задача этого механизма состоит в том, чтобы по некоторому описанию сущности, имеющемуся в рабочей памяти, найти в БЗ объекты, удовлетворяющие этому описанию; нахождение объектов в общем случае проходит в два этапа: на первом этапе, соответствующем процессу выбора по ассоциативным связкам, совершается предварительный выбор в базе знаний потенциальных кандидатов; на втором этапе путем выполнения операции сопоставления потенциальных кандидатов с описаниями кандидатов осуществляется окончательный выбор искомым объектов;
- способ сопоставления - операция сопоставления может использоваться не только как средство выбора нужного объекта из множества кандидатов; она может быть использована для классификации (неизвестный объект может быть сопоставлен с некоторыми известными образцами, что позволит классифицировать неизвестный объект как такой известный образец, при сопоставлении с которым были получены лучшие результаты), подтверждения (при поиске сопоставление используется для подтверждения некоторых кандидатов из множества возможных) и декомпозиции (если осуществлять сопоставление некоторого известного объекта с неизвестным описанием, то в случае успешного сопоставления будет осуществлена частичная декомпозиция описания); выделяют следующие формы операции сопоставления: синтаксическое (соотносят формы (образцы), а не содержание объектов; если при этом образцы оказываются идентичными, то сопоставление успешно выполнено; результат синтаксического сопоставления является бинарным: образцы сопоставляются или не сопоставляются), параметрическое (вводится параметр, определяющий степень сопоставления), семантическое

(соотносятся не образцы объектов, а их функции) и принуждаемое (один сопоставляемый образец рассматривается с точки зрения другого; в отличие от других типов сопоставления здесь всегда может быть получен положительный результат - вопрос состоит только в силе принуждения) сопоставления.

Еще один важный вопрос – выбор метода поиска для решения задачи в ЭС. Методы решения задач, основанные на сведении их к поиску, зависят от:

1) Особенности предметной области, в которой решается задача

Особенности предметной области с точки зрения методов решения можно характеризовать следующими параметрами:

- размер, определяющий объем пространства, в котором предстоит искать решение;
- изменяемость области, характеризует степень изменяемости области во времени и пространстве (статические и динамические области);
- полнота модели, описывающей область, характеризует адекватность модели, используемой для описания данной области; если модель не полна, то для описания области используют несколько моделей, дополняющих друг друга за счет отражения различных свойств предметной области;
- определенность данных о решаемой задаче, характеризует степень точности (ошибочности) и полноты (неполноты) данных; точность (ошибочность) является показателем того, что предметная область с точки зрения решаемых задач описана точными или неточными данными; под полнотой (неполнотой) данных понимается достаточность (недостаточность) входных данных для однозначного решения задачи.

2) Требований, предъявляемых пользователем к решению.

Требования пользователя к результату задачи, решаемой с помощью поиска, можно характеризовать:

- количеством решений - следующие основные значения: одно решение,

несколько решений, все решения;

- свойствами результата - задает ограничения, которым должен удовлетворять полученный результат или способ его получения; например, для системы, выдающей рекомендации по лечению больных, пользователь может указать требование не использовать некоторое лекарство.
- способом получения результата.

Существующие методы решения задач, используемые в ЭС, можно классифицировать следующим образом:

- методы поиска в одном пространстве - методы, предназначенные для использования в следующих условиях: области небольшой размерности, полнота модели, точные и полные данные;
- методы поиска в иерархических пространствах - методы, предназначенные для работы в областях большой размерности;
- методы поиска при неточных и неполных данных;
- методы поиска, использующие несколько моделей, предназначенные для работы с областями, для адекватного описания которых одной модели недостаточно.

Сложность задачи, решаемой в ЭС, может варьироваться от простых задач (т.е. задач малой размерности с неизменяемыми определенными данными и отсутствием ограничений на результат и способ его получения) до сложных задач (задач большой размерности с изменяемыми, ошибочными и неполными данными и произвольными ограничениями на результат и способ его получения). Каким-либо одним методом нельзя решить весь спектр задач, т.к. одни методы превосходят другие только по некоторым из перечисленных параметров, поэтому перечисленные методы при необходимости должны применяться комбинированно для того, чтобы позволить решать задачи, сложность которых возрастает одновременно по нескольким параметрам.

Существует более высокий класс приложений, где требуется учитывать динамику изменения окружающего мира за время исполнения приложения.

Такие экспертные системы получили название динамических ЭС и их обобщённая структура будет иметь вид, приведённый на рисунке 37. По сравнению со статической ЭС в динамическую вводится ещё два компонента:

- подсистема моделирования внешнего мира;
- подсистема сопряжения с внешним миром.

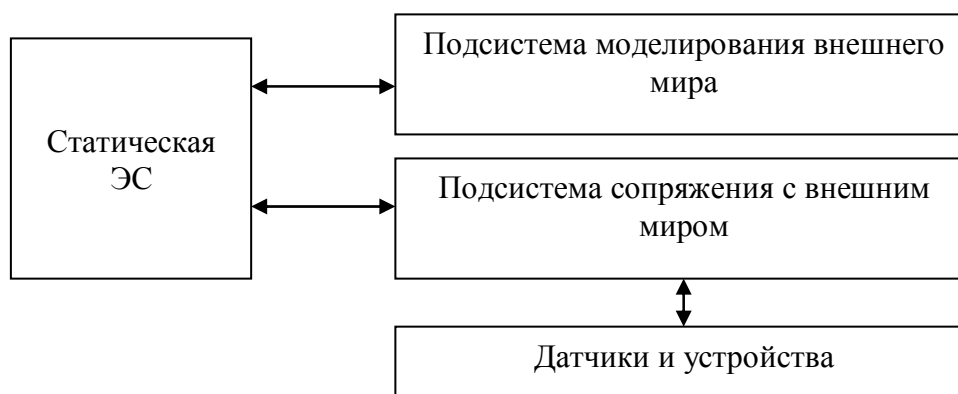


Рисунок 37. Структура динамической ЭС.

Динамические ЭС осуществляет связи с внешним миром через систему контроллеров и датчиков. Кроме того компоненты БЗ и МЛВ существенно изменяются, чтобы отразить временную логику происходящих в реальном мире событий.

В эксплуатации ЭС можно выделить два основных режима:

1) Обучение (приобретение знаний) ЭС.

В этом режиме общение с ЭС осуществляет эксперт (при помощи инженера знаний). Используя МПЗ, эксперт описывает проблемную область в виде совокупности фактов и правил, т.е. "наполняет" ЭС знаниями, которые позволяют ей самостоятельно решать задачи из проблемной области. Этому режиму при традиционном подходе к программированию соответствуют этапы: алгоритмизации, программирования и отладки, выполняемые программистом. Таким образом, в отличие от традиционного подхода в случае ЭС разработку

программ осуществляет не программист, а эксперт, не владеющий программированием.

## 2) Решение задач (консультация) пользователей.

В режиме консультаций общение с ЭС осуществляет конечный пользователь, которого интересует результат, а также способ получения результата. В зависимости от назначения ЭС пользователь может:

- не быть специалистом в данной предметной области, и в этом случае он обращается к ЭС за результатом, который не умеет получить сам;
- быть специалистом, и в этом случае он обращается к ЭС с целью ускорения получения результата, возлагая на ЭС рутинную работу.

В отличие от традиционных программ, ЭС при решении задачи не только исполняют предписанную алгоритмом последовательность операций, но и сама предварительно формирует её. Хорошо построенная ЭС имеет возможность самообучаться на решаемых задачах, пополняя автоматически свою БЗ результатами полученных выводов и решений.

Использовать ЭС следует только тогда, когда разработка ЭС возможна и оправдана, а методы инженерии знаний соответствуют решаемой задаче. Разработку ЭС можно считать возможной и оправданной, если выполнимы следующие требования:

- существуют эксперты в данной области, причем они сходятся в оценке предлагаемого решения, иначе нельзя будет оценить качество разработанной ЭС;
- эксперты способны выразить на естественном языке и объяснить используемые ими методы, в противном случае трудно рассчитывать на то, что знания экспертов будут "извлечены" и вложены в ЭС;
- задача может быть естественным образом решена посредством манипуляции с символами (т.е. с помощью символических рассуждений), а не манипуляций с числами, как принято в математических методах и в традиционном программировании; задача должна иметь эвристическую, а



не алгоритмическую природу, т.е. ее решение должно требовать применения эвристических правил;

- задача не должна быть слишком трудной для эксперта, и ее решение должно занимать у эксперта не больше нескольких дней, в противном случае решение задачи на ЭС также может быть слишком долгим;
- задача хотя и не должна быть выражена в формальном виде, но все же должна относиться к достаточно узкой, "понятной" и структурированной области, т.е. должны быть выделены основные понятия, отношения и известные (хотя бы эксперту) способы получения решения задачи; также важно, чтобы решение задачи не в значительной степени использовало "здравый смысл" (т.е. широкий спектр общих сведений о мире и о способе его функционирования, которые знает и умеет использовать любой нормальный человек), так как подобные знания пока не удается (в достаточном количестве) вложить в системы ИИ;
- задача должна быть практически значимой, чтобы оправдать затраты на разработку ЭС.

Обобщенная технология разработки ЭС включает в себя шесть этапов:

- этап идентификации – предназначен для определения цели разработки, задач, подлежащих решению, экспертов и типов пользователей ЭС;
- этап концептуализации – предназначен для выполнения содержательного анализа предметной области, выделения основных понятий и их взаимосвязей, определения методов решения задач;
- этап формализации – предназначен для выбора программных средств разработки ЭС, определения способов представления всех видов знаний, формализации основных понятий;
- этап выполнения (наиболее важный и трудоёмкий) – предназначен для наполнения экспертом с помощью инженера знаний БЗ ЭС ("извлечение" знаний из эксперта; организация знаний, обеспечивающая эффективную работу ЭС; представление знаний в виде, понятном для ЭС);
- этап тестирования – предназначен для проверки с помощью эксперта и

инженера знаний компетентности ЭС с использованием диалоговых и объяснительных средств; процесс тестирования продолжается до тех пор, пока эксперт не решит, что система достигла требуемого уровня компетентности.

- этап опытной эксплуатации – предназначен для проверки пригодности ЭС для конечных пользователей; по результатам этого этапа возможна существенная модернизация ЭС.

Процесс создания ЭС не сводится к строгой последовательности этих этапов, так как в ходе разработки приходится неоднократно возвращаться на более ранние этапы и пересматривать принятые там решения.

При разработке ЭС, как правило, используется концепция "быстрого прототипа". Суть этой концепции состоит в том, что разработчики не пытаются сразу построить конечный продукт, а совершенствуют его путем выпуска ряда прототипов. Прототипы должны удовлетворять двум противоречивым требованиям:

- прототип должен решать типичные задачи;
- время и трудоемкость разработки прототипа должны быть весьма незначительны.

Прототип должен продемонстрировать пригодность методов инженерии знаний для данного приложения. В случае успеха эксперт с помощью инженера по знаниям расширяет знания прототипа о проблемной области. При неудаче может потребоваться разработка нового прототипа или разработчики могут прийти к выводу о непригодности методов ЭС для данного приложения. По мере увеличения знаний прототип может достигнуть такого состояния, когда он успешно решает все задачи данного приложения. Преобразование прототипа ЭС в конечный продукт обычно приводит к перепрограммированию ЭС на языках низкого уровня, обеспечивающих как увеличение быстродействия ЭС, так и уменьшение требуемой памяти. Трудоемкость и время создания ЭС в значительной степени зависят от типа используемого инструментария. Различают следующие прототипы ЭС на разных этапах разработки (примерная

классификация):

- демонстрационный – доказательство работоспособности выбранного подхода, БЗ содержит 50 – 100 правил, срок разработки 1 – 3 месяца;
- исследовательский – изучение и тестирование системы, внушающие доверие результаты; БЗ содержит 200 – 500 правил; срок разработки 1 – 2 года;
- опытной эксплуатации – высокое качество результатов при достаточной надежности, тестирование в условиях реальной эксплуатации; БЗ включает 500 – 1000 правил, 2 – 3 года разработки;
- промышленный – высокое качество, надежность, быстроедействие и эффективность работы; БЗ включает 500-1000 правил, 2 – 4 года разработки;
- коммерческий – промышленный прототип на коммерческой основе, БЗ содержит более 3000 правил; 6 лет разработки.

В настоящее время технология ЭС используется для решения различных типов задач (интерпретация, предсказание, диагностика, планирование, конструирование, контроль, отладка, инструктаж, управление ) в самых разнообразных проблемных областях, таких, как финансы, нефтяная и газовая промышленность, энергетика, транспорт, фармацевтическое производство, космос, металлургия, горное дело, химия, образование, целлюлозно-бумажная промышленность, телекоммуникации и связь и др. Коммерческий успех ЭС определен в настоящее время не только решением сложноформализуемых задач, но и следующими причинами:

- интегрированностью – наличие инструментальных средств ИИ, легко интегрирующихся с другими информационными технологиями и средствами (CASE, СУБД, контроллерами, концентраторами данных и т.п.);
- открытостью и переносимостью – программные средства ЭС разрабатываются с соблюдением стандартов, обеспечивающих открытость и переносимость;

- использованием языков традиционного программирования - переход от средств, реализованных на языках ИИ (Lisp, Prolog и т.п.), к средствам, реализованным на языках традиционного программирования (C, C++ и т.п.), что упрощает обеспечение интегрированности с традиционными программами;
- архитектурой клиент-сервер - поддержка распределенных вычислений, что позволило децентрализовать приложения, повысить надежность и общую производительность;
- естественно-языковым интерфейсом – снижение затрат на обучение конечных пользователей ЭС;
- объектно-ориентированными средствами ИИ - сокращение сроков разработки приложений, увеличение эффективности использования ИС, упрощение и ускорение работы эксперта, повторная используемость информационного и программного обеспечения (объекты, классы, правила).

Примеры крупномасштабных экспертных систем:

- MICIN — ЭС для медицинской диагностики, разработана группой по инфекционным заболеваниям Стенфордского университета; может ставить соответствующий диагноз, исходя из представленных ей симптомов, и рекомендует курс медикаментозного лечения любой из диагностированных инфекций; БЗ состоит из 450 правил;
- PUFF — ЭС для анализа нарушений дыхания, данная система представляет собой MICIN, из которой удалили данные по инфекциям и вставили данные о легочных заболеваниях;
- DENDRAL — ЭС для распознавания химических структур, первые версии данной системы появились еще в 1965 году в Стенфордском университете; пользователь дает системе DENDRAL некоторую информацию о веществе, а также данные спектроскопии (инфракрасной, ядерного магнитного резонанса и масс-спектрометрии), и та в свою очередь выдает диагноз в виде соответствующей химической структуры;

- PROSPECTOR — ЭС, созданная для содействия поиску коммерчески оправданных месторождений полезных ископаемых;
- G2 – базовый программный продукт фирмы Gensym Corp. (США), представляющий собой графическую, объектно-ориентированную среду для построения и сопровождения динамических ЭС реального времени, предназначенных для мониторинга, диагностики, оптимизации, планирования и управления динамическим процессом.

## 8. ЗНАНИЯ

В системах ИИ знания являются основным объектом формирования, обработки и исследования. БЗ, наравне с БД, - необходимая составляющая программного комплекса ИИ. Машины, реализующие алгоритмы ИИ, называются машинами, основанными на знаниях, а подраздел теории ИИ, связанный с построением ЭС, - инженерией знаний.

В ЭВМ знания так же, как и данные, отображаются в знаковой форме - в виде формул, текста, файлов, информационных массивов и т.п. Поэтому можно сказать, что знания - это особым образом организованные данные.

Свойства знаний, отличающие их от данных:

- внутренняя интерпретируемость – кроме самой информации, в памяти ЭВМ должна храниться информация о некоторой протоструктуре информационных единиц (подобно метаданным в СУБД);
- структурированность - информационные единицы должны обладать гибкой структурой; при этом должна существовать возможность рекурсивной вложенности одних информационных единиц в другие и произвольного установления между отдельными информационными единицами отношений типа "часть-целое", "род-вид" или "элемент-класс";
- связность - между информационными единицами должна быть предусмотрена возможность установления связей различного типа; эти связи могут характеризовать отношения между информационными единицами (например, связь "причина-следствие" – декларативные знания; связь "аргумент-функция" - процедурные знания, связанные с вычислением определенных функций); различают отношения структуризации (иерархии информационных единиц), функциональные отношения (несут процедурную информацию, позволяющую вычислять одни информационные единицы через другие), каузальные отношения (задают причинно - следственные связи) и семантические отношения

(соответствуют всем остальным отношениям);

- семантическая метрика - отношение, характеризующее ситуационную близость информационных единиц (ассоциативную связь между информационными единицами); такое отношение дает возможность выделять в информационной базе некоторые типовые ситуации (например, "покупка", "управление машиной") и позволяет находить знания, близкие к некоторым, уже найденным;
- активность – знания активны, появление в базе фактов или описаний событий, установление связей может стать источником активности системы.

В системах ИИ знания в основном можно разделить на две группы: факты (фактические знания) и правила (знания для принятия решений).

В рамках работы со знаниями можно выделить следующие направления:

- представление знаний - в рамках этого направления решаются задачи, связанные с формализацией и представлением знаний в памяти системы ИИ, (разработка специальных моделей представления знаний и языков описания знаний);
- манипулирование знаниями - в рамках данного направления разрабатываются способы пополнения знаний на основе их неполных описаний, создаются методы достоверного и правдоподобного вывода на основе имеющихся знаний, предлагаются модели рассуждений, опирающихся на знания и имитирующих особенности человеческих рассуждений.

Манипулирование знаниями очень тесно связано с представлением знаний, и разделить эти два направления можно лишь условно.

## 8.1. Модели представления знаний

Проблема представления знаний оказывает существенное влияние на

характеристики и свойства системы. Для возможности оперирования знаниями из реального мира с помощью ЭВМ, необходимо осуществить их моделирование (по аналогии с построением концептуальных и логических моделей БД). При выполнении этого процесса следует учесть два важных условия задания модели знаний:

- однородность представления знаний - приводит к упрощению механизма управления логическим выводом и управлением знаниями;
- простота понимания знаний - доступность понимания представления знаний и экспертам, и пользователем системы, в противном случае затрудняется приобретение знаний и их оценка.

Различают три типа моделей представления знаний:

1) Формальные модели представления знаний.

Система ИИ в определенном смысле моделирует интеллектуальную деятельность человека и, в частности, - логику его рассуждений. В упрощенной форме логические построения человека сводятся к следующей схеме: из одной или нескольких посылок (которые считаются истинными) следует сделать "логически верное" заключение (вывод, следствие). Очевидно, для этого необходимо, чтобы и посылки, и заключение были представлены на понятном языке, адекватно отражающем предметную область, в которой проводится вывод. В обычной жизни это наш естественный язык общения, в математике, например, это язык определенных формул и т.п. Наличие же языка предполагает, во-первых, наличие алфавита (словаря), отображающего в символической форме весь набор базовых понятий (элементов), с которыми придется иметь дело, и, во-вторых, набор синтаксических правил, на основе которых, пользуясь алфавитом, можно построить определенные выражения. Логические выражения, построенные в данном языке, могут быть истинными или ложными. Некоторые из этих выражений, являющиеся всегда истинными, объявляются аксиомами. Они составляют ту базовую систему посылок, исходя из которой и пользуясь определенными правилами вывода, можно получить заключения в виде новых выражений, также являющихся истинными. Если



перечисленные условия выполняются, то говорят, что система удовлетворяет требованиям формальной теории и считается формальной системой. Логические модели, построенные на основе на исчислении высказываний или исчислении предикатов, являются классическими примерами формальных систем.

## 2) Неформальные модели представления знаний.

В отличие от формальных моделей, в основе которых лежит строгая математическая теория, неформальные модели такой теории не придерживаются. Каждая неформальная модель годится только для конкретной предметной области и поэтому не обладает той универсальностью, которая присуща моделям формальным. Логический вывод в формальных системах строг и корректен, а вывод в неформальных системах во многом определяется самим исследователем, который и отвечает за его корректность. К неформальным моделям относятся:

- модель семантической сети;
- модель, основанная на использовании фреймов;
- продукционная модель.

## 3) Интегрированные модели представления знаний.

Интегрированные МПЗ совмещают в себе модели различных типов.

Выбор модели представления знаний при построении системы ИИ является ответственным этапом и выполняется инженером знаний.

### 8.1.1. Модель на базе логики

Одним из способов представления знаний является язык математической логики – логики предикатов, позволяющий формально описывать понятия предметной области и связи между ними. В отличие от естественного языка, который очень сложен, язык логики предикатов использует только такие конструкции естественного языка, которые легко формализуются.

Логика предикатов - это формальная языковая система, которая оперирует с предложениями на естественном языке в пределах синтаксических правил этого языка. Формально логическая модель задается четверкой вида:

$$M = \langle T, F, A, P \rangle.$$

Здесь,  $T$  – алфавит (множество базовых элементов различной природы),  $F$  – множество формул (задается в виде синтаксических правил),  $A$  – множество аксиом,  $P$  – множество правил вывода. Существуют процедура  $\Pi(X)$ , которая позволяет определить принадлежит или не принадлежит ее аргумент множеству  $X$  (в качестве  $X$  могут использоваться  $T, F, A$  и  $P$ ).

Для знаний, входящих в БЗ, можно считать, что множество  $A$  образуют все информационные единицы, которые введены в БЗ извне, а с помощью правил вывода из них выводятся новые производные знания. Другими словами формальная система представляет собой генератор порождения новых знаний, образующих множество выводимых в данной системе знаний. Это свойство логических моделей делает их притягательными для использования в базах знаний. Оно позволяет хранить в базе лишь те знания, которые образуют множество  $A$ , а все остальные знания получать из них по правилам вывода.

Фактически язык логики предикатов использует слова, которые делятся на следующие группы:

- термы – описывают понятия и объекты изучаемой предметной области;
- предикаты – задают свойства этих объектов и понятий, а также их поведение и отношения между ними.

При логическом программировании пользователь описывает предметную область совокупностью предложений в виде логических формул, а ЭВМ, манипулируя этими предложениями, строит необходимый для решения задач вывод.

Термы - это символы (или комбинация символов), являющиеся наименьшим значимым элементом языка. К термам относятся:

- константы - применяются для обозначения конкретных объектов реального мира (например, «студент», «птица», «5» и т.п.);

- переменные - используются для обозначения некоторого из возможных объектов реального мира или их совокупности (например, некто, вещь и т.п.);
- функции - последовательность из нескольких констант или переменных, заключенных в круглые скобки, следующие за особым функциональным символом - функтором (оператором, который после воздействия на объект возвращает некоторое значение); например, сумма (1,2), увеличить\_на\_единицу(X).

Предикат - это логическая функция, которая выражает отношение между своими аргументами и принимает значение «истина», если это отношение имеется, или «ложь», если оно отсутствует. Заключенная в скобки последовательность из n термов, перед которой стоит предикатный символ, называется n-арным предикатом, который принимает значения «истина» или «ложь» в соответствии со значением термов, являющимися его аргументами. Например,

студент (Иванов)

владелец (X, «А 1820 МС»)

Приведенные выше предикаты получили название атомарных предикатов и соответствуют наиболее простым предложениям нашего разговорного языка - нераспространенным предложениям. В обычном языке из нераспространенных предложений с помощью соединительных местоимений, союзов, и других частей речи строят более сложные конструкции - сложные предложения. В логике предикатов сложными предложениями естественного языка соответствуют предикатные формулы, которые образуются из атомарных предикатов и логических связок (расположены по уменьшению приоритета):

$\neg$  - «не»

$\wedge$  - «и»;

$\vee$  - «или»;

$\rightarrow$  - «если»;

$\leftrightarrow$  - «тогда и только тогда».

Пример предикатной формулы, соответствующей сложному предложению «Некто является птицей, если некто имеет крылья и владеет гнездом»:

является (X, птица)  $\leftarrow$  имеет (X, крылья), владеет (X, гнездо)

где является(), имеет() и владеет() - атомарные предикаты, а - X – переменная, при подстановке в которую конкретного значения (например, X:=«орёл») можно оценить значение предикатной формулы (ложь или истина). Здесь для истинности предиката является() должна быть истинной связка имеет() и владеет(). Иногда можно определить значения предиката не делая подстановок констант, а используя кванторы общности ( $\forall$  - «для всех») и существования ( $\exists$  - «существует по крайней мере одно»), например «Любой некто является птицей если некто имеет крылья и владеет гнездом»:

$(\forall X)$  [являться (X, птица)  $\leftarrow$  имеет (X, крылья), владеет (X, гнездо)]

Кванторы  $\forall$  и  $\exists$  могут использоваться и для любого числа переменных. Рассмотрим их различное использование на примере двухместного предиката изучает (S, P), который описывает отношение «S изучает P» (S – переменная, которая задает студента, а P - предмет):

- $(\forall S) (\forall P)$  изучает (S, P) - все студенты изучают все предметы;
- $(\exists S) (\forall P)$  изучает (S, P) – существует по крайней мере один студент, который изучает все предметы;
- $(\forall S) (\exists P)$  изучает (S, P) - каждый студент изучает по крайней мере один предмет;
- $(\exists S) (\exists P)$  изучает (S, P) - существует по крайней мере один студент, который изучает по крайней мере хоть один предмет.

Для построения составных предикатных формул языка логики предикатов первого порядка (или правильно построенных формул (ППФ или логических формул)), задан следующий набор правил:

- термом является либо константа, либо переменная, либо кортеж из n термов, перед которым стоит функтор;

- предикат - это кортеж из  $n$  термов, перед которым стоит предикатный символ;
- атомарный предикат является логической формулой;
- если  $F$  и  $G$  - логические формулы, то  $(F)$ ;  $F,G$ ;  $F \vee G$ ;  $\neg F$ ;  $F \rightarrow G$ ;  $F \leftrightarrow G$  - также являются логическими формулами;
- если  $F(X)$  - логическая формула, то оба выражения  $(\forall X) F(X)$ ,  $(\exists X) F(X)$  является логическими формулами;
- все результаты, получаемые повторением конечного числа перечисленных выше правил являются логическими формулами;

Например, ППФ предложения «Все люди смертны» будет записано в виде:  $(\forall X) [\text{человек}(X) \leftarrow \text{смертен}]$ .

В логической модели определен специальный механизм работы со знаниями - логический вывод. Логический вывод - это процесс получения из множества правильно построенных формул ( $S$ ) некоторой новой ППФ ( $S'$ ) путем применения одного или нескольких правил вывода. Наиболее простой метод логического вывода использует только одно правило вывода, называемое резолюцией, которое применяют к логическим формулам вида:

факт:  $A$   
 отрицание:  $\neg(A_1, \dots, A_n)$   
 импликация:  $A \leftarrow B_1, \dots, B_m$ ,

где  $A_i$  ( $i = 1, n$ ) и  $B_j$  ( $j = 1, m$ ) - произвольные предикаты.

Для  $S = \{ S_1, S_2 \}$ , для которых выполняются следующие условия:

$S_1$  (отрицание):  $\neg A$ ,  
 $S_2$  (импликация):  $A \leftarrow B$ ,

предикат  $A$  из  $S_1$  совпадает с предикатом  $A$  левой части  $S_2$ ,

можно по выше указанному правилу резолюций вывести новую ППФ вида:

$S'$ :  $\neg B$

При этом ППФ  $S_1$  и  $S_2$  называются родительскими предложениями, а  $S'$  -

резольвентой, которая получается в результате применения резолюции к  $S_1$  и  $S_2$ .

Резолюция в этом простейшем случае соответствует правилу вывода modus tollens, которое записывается в виде:

$$\neg B \leftarrow \neg A, A \leftarrow B$$

и соответствует следующему умозаключению: допуская, что НЕ А и А ЕСЛИ В, выводим НЕ В.

Применяя правило вывода modus tollens в случае, когда:

$$S_1 \text{ (отрицание): } \neg A$$

$$S_2 \text{ (факт): } A,$$

применение правила резолюции даст в качестве резольвенты пустое отрицание  $S'$ :  $\square$ , что будет означать противоречие, т.е. допуская, что: НЕ А и А выводим противоречие ( $\neg(\neg A) = A$ ).

Для более общего случая, когда родительские предложения имеют вид (причем предикат  $A_k$  из отрицания  $S_1$  совпадает с предикатом из левой части  $S_2$ ):

$$S_1: \neg(A_1, \dots, A_k, \dots, A_n)$$

$$S_2: A_k \leftarrow (B_1, \dots, B_m) \quad (\text{где } 1 < k < n)$$

в качестве резольвенты будет получено следующее отрицание:

$S'$ :  $\neg(A_1, \dots, A_{k-1}, B_1, \dots, B_m, A_{k+1}, \dots, A_n)$ , т.е. в  $S_1$  будет выполнена замена  $A_k$  на  $(B_1, \dots, B_m)$ .

Например, допуская, что « $\neg$ (темно, зима, холодно) и что зима  $\leftarrow$  январь», выводим, что  $\neg$ (темно, январь, холодно).

В случае, если родительские предложения имеют вид ( $A_k$  является одним из предикатов из  $S_1$ ):

$$S_1 \text{ (отрицание): } \neg(A_1, \dots, A_k, \dots, A_n)$$

$$S_2 \text{ (факт): } A_k$$

в качестве резольвенты будет получено следующее отрицание:

$$S': \neg(A_1, \dots, A_{k-1}, A_{k+1}, \dots, A_n), \text{ т.е. из } S_1 \text{ будет удален } A_k.$$

Например. допуская, что « $\neg$  (темно, зима, холодно) и что зима» выводим, что « $\neg$  (темно, холодно)».

Рассмотренные выше правила могут быть применены и к большому количеству родительских предложений. Например, для следующих знаний:

$S_{11}$ : получает (студент, стипендию)  $\leftarrow$  сдает (успешно, сессию, студент)

$S_{12}$ : сдает (успешно, сессию, студент)

нужно ответить на вопрос «Получает ли студент стипендию?». Для решения данной проблемы можно воспользоваться доказательством от противного, т.е. представить вопрос в виде отрицания «Студент не получает стипендию», тогда получив в качестве резольвенты пустое множество мы опровергнем это отрицание (соответственно докажем противоположное). Тогда решение можно задать следующими шагами (используя последовательную подстановку):

1)  $S_1$ :  $\neg$  получает (студент, стипендию)

$S_{11}$ : получает (студент, стипендию)  $\leftarrow$  сдает (успешно, сессию, студент)

Резольвента:  $S'$ :  $\neg$  сдает (успешно, сессию, студент).

2)  $S_2$ :  $\neg$  сдает (успешно, сессию, студент)

$S_{12}$ : сдает (успешно, сессию, студент)

Резольвента:  $S'$ :  $\square$  (пустое множество)

Таким образом за два шага была доказана противоречивость отрицания « $\neg$  получает (студент, стипендию)», что означает ответ «да» на заданный вопрос (или подтверждение высказывания «получает (студент, стипендию)»).

Подобный логический вывод, который порождает последовательность отрицаний, называется резолюцией сверху вниз.

В общем виде резолюция сверху вниз записывается не только с помощью констант, но и с использованием переменных и функций. Тогда для выполнения метода резолюций необходимо использовать унификаторы, иначе в левой части импликации и в отрицании может не оказаться одинаковых предикатов. Унификатором называется множество присваиваний вида

$$\Theta = \{X_1 := t_1, \dots, X_n := t_n\}$$

где  $X_i$  - переменная, а  $t_i$  – терм, применение которых к двум выражениям дает одинаково общие примеры. На практике унификаторы определяют, сравнивая по очереди соответствующие аргументы предикатов и выписывая те присваивания термов переменным, которые сделали бы эти аргументы одинаковыми. Например, для предикатов (данные предикаты называются родительскими)

получает (X, стипендию)

получает (студент, Y)

с помощью унификатора  $\Theta = \{ X:= студент, Y:= стипендия \}$  можно получить предикат «получает (студент, стипендию)».

Общий алгоритм решения задач с использованием логической модели можно задать следующими этапами:

1) Сформулировать знания и допущения о предметной области в виде множества ППФ.

2) Выразить конкретную задачу, поставленную на предметной области, как запрос об одном или нескольких отношениях. Обычно запрос ставится как исходное отрицание.

3) Решить задачу строя доказательство от противного. Для этого строится вывод сверху вниз, начиная с исходного отрицания, и порождает последовательность отрицаний  $D_1, D_2, \dots, D_n$ . Если может быть построен вывод, заканчивающийся отрицанием  $D_n = \square$  (противоречие), то этот вывод, называемый успешным выводом, сразу дает решение поставленной задачи.

Пример решения задачи нахождения значения факториала.

Этап 1) Формулировка знаний о вычислении факториала с помощью ППФ:

$S_1 : F(0, 1)$

$S_2 : F(x, y) \leftarrow F(x-1, y_1), y = y_1 * x$

Этап 2) Формулировка запроса для решения (найти 3!):

$D_1 : \neg F(3, z)$

Этап 3) Работа системы логического вывода на основе метода резолюции



(по шагам):

1)  $D_1: \neg F(3, z)$

$$S_2 : F(x, y) \leftarrow F(x-1, y_1), y = y_1 * x$$

Резольвента  $D_2: \neg F(2, z/3)$  и унификатор  $\Theta = \{x:=3, y:=z, y_1:=y/x\} = \{x:=3, y_1:= z/3\}$

2)  $D_2: \neg F(2, z/3)$

$$S_2 : F(x, y) \leftarrow F(x-1, y_1), y = y_1 * x$$

Резольвента  $D_3: \neg F(1, z/6)$  и унификатор  $\Theta = \{x:=2, y:=z/3, y_1:=y/x\} = \{x:=2, y_1:= z/6\}$

3)  $D_3: \neg F(1, z/6)$

$$S_2 : F(x, y) \leftarrow F(x-1, y), y = y * x$$

Резольвента  $D_4: \neg F(0, z/6)$  и унификатор  $\Theta = \{x:=1, y:=z/6, y_1:=y/x\} = \{x:=1, y_1:= z/6\}$

4)  $D_4: \neg F(0, z/6)$

$$S_1 : F(0, 1)$$

Резольвента  $\square$  и унификатор  $\Theta = \{z/6:=1\} = \{z:=6\}$

Полученное на шаге 4 противоречие подтверждает отрицание  $D_1$ , а стало быть, вывод является успешным и дает решение для предиката « $F(3,z)$ » с унификатором  $\Theta = \{z:=6\}$ , т.е. « $F(3, 6)$ » и ответ  $z = 6$ .

Логическая модель является классическим примером формальных систем (т.е. систем, которые основаны на строгой математической теории). Такие системы хорошо исследованы, имеют надежные модели логического вывода и дают гарантии непротиворечивости вывода, алгоритмической разрешимости (для исчисления высказываний) и полуразрешимости (для исчислений предикатов первого порядка). Основные недостатки таких систем:

- "закрытость" (т.е. негибкость), т.к. модификация и расширение здесь всегда связаны с перестройкой всей системы, что для практических систем сложно и трудоемко;
- сложный учет происходящих изменений.

Поэтому формальные системы как модели представления знаний используются редко (причем в более сложном виде (модификации и расширения), чем описанная здесь модель) и только в тех предметных областях, которые хорошо локализируются и мало зависят от внешних факторов.

### 8.1.2. Семантические сети

Семантическая сеть - это система знаний, имеющая определенный смысл в виде целостного образа сети, узлы которой соответствуют понятиям и объектам, а дуги - отношениям между объектами. Семантические сети относятся к неформальным моделям знаний, т.к. не основаны на строгой математической теории. Сама по себе семантическая сеть является моделью памяти и не раскрывает, каким образом осуществляется представления знаний, поэтому семантические сети должны рассматриваться как метод представления знаний с возможностями структурирования этих знаний, процедурами их использования и механизмом вывода. Само название «семантическая сеть» акцентирует внимание на смысле (семантика – смысловая сторона языка, "семантический" – связанный со смыслом), т.е. значениях тех слов, предложений, ситуаций, состояний, которые входят в модель предметной области и которые необходимо описать адекватно нашему (человеческому) пониманию таким образом, чтобы с этим смыслом можно было работать, т.е. моделировать процессы, рассуждения, выводы, сравнения, сопоставления и т.п.

Семантическая сеть представляет собой ориентированный граф с помеченными (поименованными) дугами и вершинами. Основными элементами сети являются:

- 1) Вершины – соответствуют таким семантическим единицам, как:
  - понятия - сведения об абстрактных или физических объектах предметной области (реального мира);
  - события - действия происходящие в реальном мире, которые

определяются путем указания типа действия или указания ролей, которые играют объекты в этом действии;

- свойства – средство уточнения понятий и событий, например, применительно к понятиям они описывают их особенности и характеристики (цвет, размер, качество), а применительно к событиям – продолжительность, время, место;

2) Дуги - отображают многообразие семантических отношений, которые условно можно разделить на следующие общие классы:

- лингвистические отношения - отображают смысловую взаимосвязь либо между несколькими событиями, либо между событиями и понятиями или свойствами; лингвистические отношения разделяются на глагольные (время, вид, род, залог, наклонение), атрибутивные (цвет, размер, форма) и падежные (ролевые - кто, что, где);
- логические отношения - это операции, используемые в исчислении высказываний (алгебра логики): дизъюнкция, конъюнкция, инверсия, импликация;
- теоретико-множественные отношения - это отношение подмножества, отношение части целого, отношение множества и элемента (множество, подмножество, объединение, дополнение, пересечение), например, «входит в», «часть из».
- квантифицированные отношения - это логические кванторы общности и существования («любой», «существует»);

Семантические сети можно классифицировать следующим образом:

- по количеству типов отношений – однородные (с одним типом отношений) и неоднородные (с различными типами отношений);
- по типам отношений – бинарные (только между двумя объектами) и n-арные (есть отношения, связывающие n понятий);
- по типу описания – сценарные, функциональные, иерархические и т.п.

В основном семантические сети используются для:

1) Описания структуры понятий.

Основой для определения любого понятия является множество его отношений с другими понятиями. Обязательными отношениями являются:

- класс, которому принадлежит данное понятие;
- свойства, выделяющие понятие из всех понятий данного класса;
- примеры (экземпляры) данного понятия.

В итоге связи понятий образуют структуру, в общем случае сетевую, в которой используется как минимум теоретико-множественные отношения.

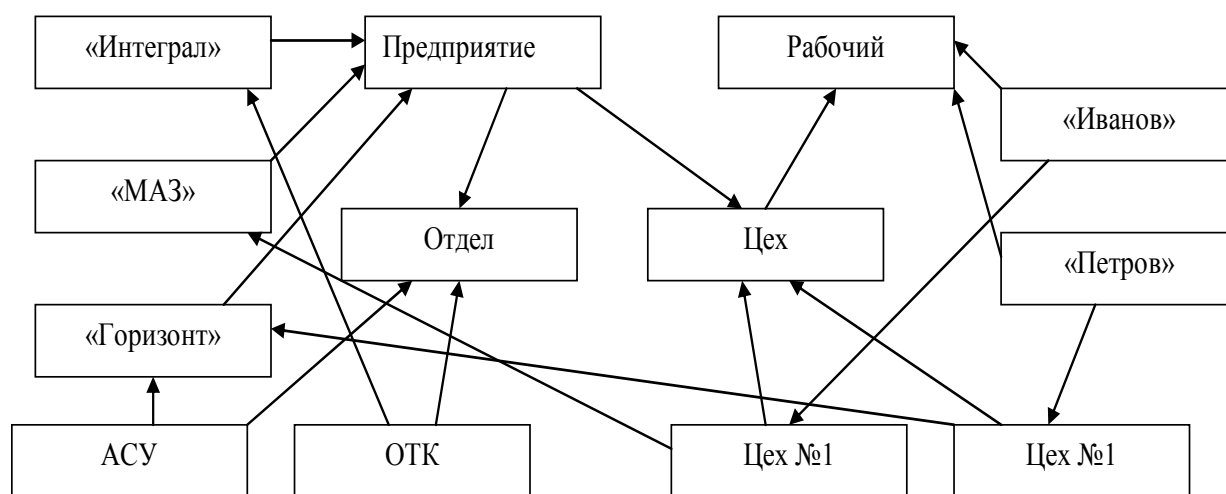


Рисунок 38. Представление структуры понятий семантической сетью.

Например, на рисунке 38 приведена семантическая сеть, в которой описаны классы и их экземпляры: «Предприятие» («Интеграл», «МАЗ», «Горизонт»), «Отдел» («АСУ», «ОТК»), «Цех» («Цех №1», «Цех №1»), «Рабочий» («Иванов», «Петров»), причем классы «Отдел» и «Цех» являются свойствами класса «Предприятие», а «Рабочий» - свойством класса «Цех».

## 2) Описания структуры событий.

При представлении событий предварительно выделяются простые отношения, которые характеризуют основные компоненты события. В первую очередь из события выделяется действие, которые обычно описываются глаголом. Далее определяются:

- объекты, которые выполняют действия;

- объекты, над которыми эти действия выполняются.

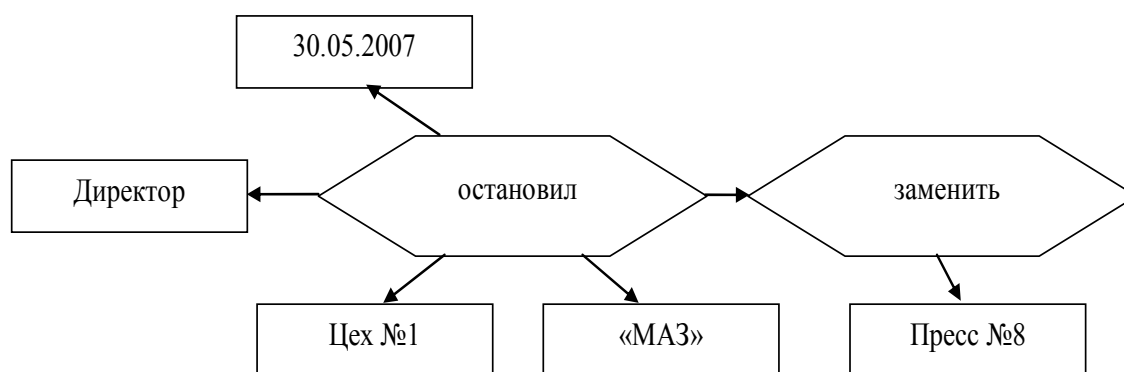


Рисунок 39. Представление событий семантической сетью.

Все связи понятий, событий и свойств с действием (глаголом) называют падежами или падежными отношениями, которые относятся к классу лингвистических отношений. Обычно рассматривают следующие падежи:

- агент - предмет, являющийся инициатором действия;
- объект - предмет, подвергающийся действию;
- источник - размещение предмета перед действием;
- приемник - размещение предмета после действия;
- время - момент выполнения действия;
- место - место проведения действия;
- действие – характер манипулирования над объектом;
- цель – действие другого события.

Например, для рисунка 39, семантическая сеть описывает событие «Директор «МАЗа» остановил 30.05.2007 цех №1 чтобы заменить пресс №8», здесь использованы следующие падежи:

- агент - директор;
- объект – цех №4;
- время – 30.05.2007;
- место – «МАЗ»;
- действие – остановил;

- цель – заменить.

Использование семантической сети для вывода возможно по двум вариантам, из-за того, что БЗ и механизм вывода заданы в самой структуре семантической сети:

1) Метод сопоставления, состоит в выполнении следующих действий:

- строится семантическая сеть, отражающая структуру запроса, где целевой объект заменяется символом «?»;
- вывод обеспечивается за счет сопоставления общей сети БЗ и сети для запроса.

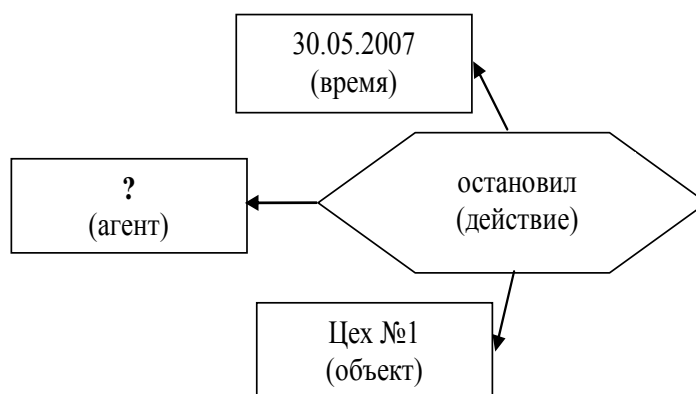


Рисунок 40. Запрос в семантической сети.

Например, на рисунке 40 сформирован запрос «Кто остановил 30.05.2007 цех №1», при выполнении поиска должен быть найден фрагмент сети, соответствующий данной сети, тогда объект, совпадающий с целевым объектом, и будет ответом на запрос («?» = «директор»).

2) Метод перекрестного поиска, при котором осуществляется поиск отношения между понятиями (например, «директор» и «цех №1»), здесь ответ на запрос формируется путем обнаружения вершины, в которой пересекаются дуги, идущие из заданных вершин (для указанного случая – «остановил»).

Преимуществами семантической сети являются:

- описание понятий и событий производится на уровне, очень близком к

естественному языку;

- обеспечивается возможность сцепления различных фрагментов сети;
- отношение между понятиями и событиями образуют достаточно небольшое и хорошо формализованное множество;
- для каждой операции над данными и знаниями можно выделить из полной сети, представляющей всю семантику (или все знания), некоторый ее участок, который охватывает необходимые в данном запросе смысловые характеристики.

Недостатки семантической сети:

- с увеличением размеров сети существенно увеличивается время поиска, теряется наглядность;
- неоднозначность описания и сложность внесения изменений;
- отсутствие формального аппарата установления противоречивости описания.

Основное применение семантические сети находят в системах обработки естественных языков, а также в системах распознавания образов, в которых они используются для хранения знаний о структуре, форме и свойствах физических объектов.

### 8.1.3. Фреймовые модели

В сложных семантических сетях, включающих множество понятий, процесс обновления узлов и контроль связей между ними становится затруднительным. При этом количество опосредованных родовидовых связей между понятиями резко возрастает. Основная идея фреймового подхода к представлению знаний заключается в том, что все, что касается понятия или ситуации, не «размывается по сети», а представляется во фрейме.

Впервые понятие фрейма (frame – рамка, каркас, структура) было введено М. Минским в 1975 году. М. Минский дал такое определение: «Фрейм – это

структура данных, представляющая стереотипную ситуацию, вроде присутствия внутри жилой комнаты или сбора на вечеринку. К каждому фрейму присоединяется несколько видов информации. Часть информации о том, как использовать фрейм. Часть о том, чего можно ожидать далее. Часть о том, что следует делать, если ожидания не подтвердятся». В основе этого понятия лежат представления гештальтпсихологии, занимающейся изучением восприятия человеком внешнего мира в форме целостных фрагментов. Наиболее часто центральным понятием такого фрагмента является объект внешнего мира с его наиболее характерными свойствами и т.п. По Минскому, фрейм – это структура данных, содержащая минимально необходимую информацию для представления класса объектов (явлений или процессов), которая однозначно определяет эти объекты. Еще фрейм называют структурой данных для представления стереотипных ситуаций.

Фрейм можно рассматривать как фрагмент семантической сети, предназначенный для описания понятий со всей совокупностью присущих им свойств. Различают следующие виды фреймов:

прототипы (протофреймы) - содержит знания, общие для всех частных случаев;

фрейм-примеры (или фрейм-экземпляры) - содержат знания, отличающие частный случай от общего.

Каждый узел во фреймовой системе имеет вид, приведенный на рисунке 41.

Имя фрейма			
Имя слота	Значение слота	Способ ввода значений	Присоединенная процедура

Рисунок 41. Структура фрейма.



В качестве имен фреймов могут фигурировать имена объектов, событий процессов и т.п. Слотами выступают характерные свойства или атрибуты описываемых объектов. В качестве значений слотов могут выступать:

имена других фреймов;

фасеты («агрегат», «диапазон», «по умолчанию» и др.);

имена процедур;

конкретные значения слотов.

Имя слота в некоторых случаях может быть расширено служебной информацией, например:

- имя пользователя, определяющего фрейм;
- дату определения или модификации фрейма;
- комментарий.

Также дополнительно со слотом может быть связан указатель наследования и указатель типа данных.

Указатель наследования показывает, какую информацию об атрибутах слотов во фрейме верхнего уровня наследуют слоты с теми же именами во фрейме нижнего уровня. Типичные указатели наследования:

- S (тот же) - слот наследуется с теми же значениями данных;
- U (уникальный) - слот наследуется, но данные в каждом фрейме могут принимать любое значение;
- I (независимый) - слот не наследуется.

Типом данных, включаемых в слот, могут быть:

- FRAME (указатель) - указывает имя фрейма верхнего уровня.
- АТОМ (переменная);
- TEXT (текстовая информация);
- LIST (список);
- LISP (процедура).

Таким образом, существует возможность ссылки из одного фрейма в один или несколько других, что позволяет вводить структурированные связи между фреймами. При этом возникает однородная семантическая сеть со сложными

вершинами, допускающими дальнейшую развертку. Заметим, что семантическую сеть можно переделать во фреймовое описание и наоборот. В случае рекурсивного вложения фреймов друг в друга, порождается иерархическая система фреймов.

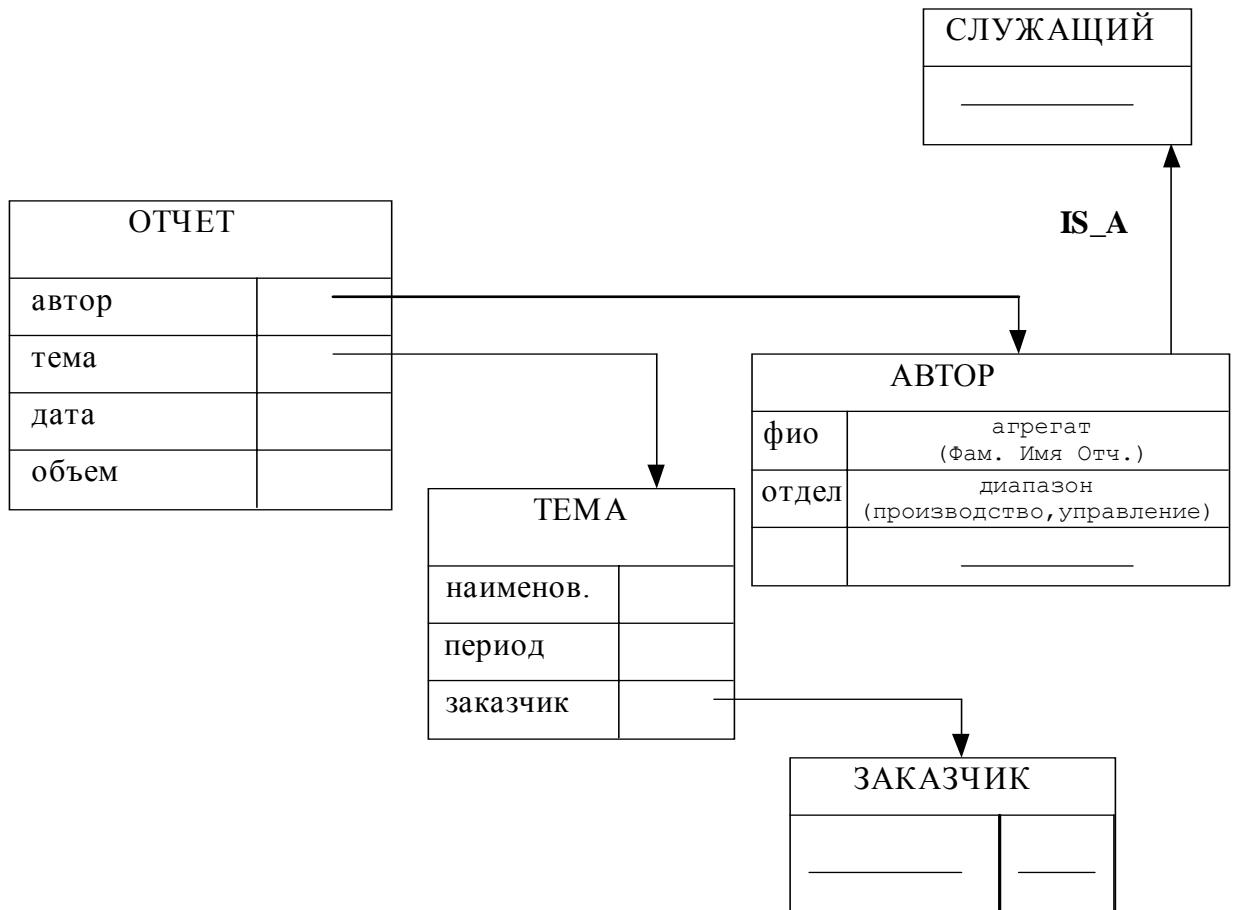


Рисунок 42. Сеть фреймов.

С каждым слотом может быть связана одна или несколько процедур, которые выполняются, когда изменяются значения слотов. Чаще всего со слотами связываются процедуры:

- если-добавлено - выполняется, когда новая информация помещается в слот;
- если-удалено - выполняется при удалении информации из слота;
- если-нужно - выполняется, когда запрашивается информация из слота, а он пуст.

Эти процедуры могут следить за присписыванием информации к данному

узлу и проверять, что при изменении значения производятся соответствующие действия.

Пример организации знаний с использованием фреймов приведен на рисунке 42.

Для создания нового отчета (например, по запросу, «Иванову И.И. создать отчет «Отчет 2» по теме «А» объемом 2 страницы») в такой сети будут выполнены следующие действия:

- создается новый экземпляр фрейма-прототипа «Отчет», который будет называться «Отчет 2»; слоты этого экземпляра заполняются либо значениями по умолчанию (например, объем отчета равен 2 страницы), либо с использованием присоединенных процедур;
- для слота «Автор» фрейма «Отчет 2» процедура «если-добавлено» выполняет поиск автора «Иванов И.И.» среди всех экземпляров авторов и устанавливает на этого автора ссылку (если найден), в противном случае генерируется сообщение об ошибке (возможны другие варианты: либо инициируется создание нового экземпляра «Автор», либо это поле будет заполнено значением по умолчанию из слота «Автор» фрейма-прототипа); для автора также формируется сообщение с требованием о создании текста отчета (это сообщение будет сформировано только при успешном заполнении остальных слотов фрейма «Отчет 2»);
- для слота «Тема» фрейма «Отчет 2» процедура «если-добавлено» выполняет поиск темы «А» среди всех экземпляров тем и устанавливает на эту тему ссылку (если найден), в противном случае генерируется сообщение об ошибке (или будут использованы другие варианты заполнения подобно выше рассмотренному слоту «Автор»);
- для слота «Дата» фрейма «Отчет 2» процедура «если-добавлено» анализирует значение даты в запросе и если оно не задано, то активизируется процедура «если-нужно» слота «Дата», которая формирует необходимое значение (например, устанавливается ближайшая допустимая дата - 01.06.2006);

- если для слота «Объем» фрейма «Отчет 2» не заданы присоединенные процедуры, то из слота «Объем» фрейма-прототипа «Отчет» будет выбрано значение по умолчанию (например, 2 страницы);
- при успешном выполнении всех выше указанных действий, формируется уведомление для автора отчета «Иванов И.И., подготовьте отчет «Отчет 2» по теме «А» к 30.06.2006 объемом 2 страницы», если по каким-либо причинам фамилия Иванов будет удалена из слота «Автор», то система автоматически отправит ему сообщение, что данный отчет не требуется.

Для организации процесса вывода на фреймах также используются механизмы наследования информации и присоединённых процедур. Такой подход позволяет объединять возможности сетевого и иерархического представления знаний.

К достоинствам фреймового представления знаний следует следующее:

- обеспечение эффективной реализации процедур вывода;
- возможность логических скачков, т.е. немонотонного вывода;
- возможность образования семантических сетей фреймов, что даёт большую экономию памяти при представлении информации за счёт наследования свойств фреймов более высоких уровней во фреймах более низких уровней;
- обеспечение хорошего соответствия реальной действительности;
- возможность комбинирования различных моделей представления знаний, объединяя их достоинства и компенсируя их недостатки.

К недостаткам фреймового представления относят:

- каждый фрейм представляет собой достаточно сложный фрагмент знаний, поэтому удаление или включение нового фрейма – весьма болезненная процедура, так как должна предусматривать и удаление всех составляющих элементов, которые могут быть составными частями других фреймов;
- достаточно сложно осуществлять на фреймах представление временных процессов;

- отсутствует формальная теория вывода на фреймах, поэтому на инженере знаний целиком лежит ответственность за корректность организации иерархии фреймов и их заполнения.

Свое наиболее яркое выражение представление знаний в виде фреймов получило в системах объектно-ориентированного программирования, поэтому очень часто языки фреймового типа называют объектно-ориентированными. Примерами таких языков является Smalltalk, языки FMS, FRL, KRL, являющиеся надстройками над LISP-системами.

#### 8.1.4. Продукционные модели

Если проследить за построением фраз у человека, то можно заметить, что значительная часть мыслей оформляется в виде правил типа

«Если (некоторое условие), то (некоторое следствие)».

Например, «Если прогнозируют дождь, то возьми зонт», «Закаляйся, если хочешь быть здоров» и т.п. Несмотря на то, что такая форма представления знаний чрезвычайно популярна для человека, она является достаточно сложной для реализации в виде жестких алгоритмических схем, так как знания многообразны (как по уровням, так и по спектру), допускают неоднозначное толкование, имеют ассоциативный характер использования. Поэтому знания сложно разложить их на причины и следствия и описывать их в логической модели знаний (на языке исчисления высказываний и предикатов). Кроме того знания постоянно меняются, пополняются, модифицируются и приспособляются к условиям среды.

Все выше сказанное (фактически простота и однородность описания знаний) обусловило широкое распространение в системах ИИ именно продукционной модели - модели, в которых знания представляются с помощью правил вида:

ЕСЛИ - ТО (явление - реакция).

Термин «продукция» был введен американским логиком Е. Постом в 40-х годах в работах по обоснованию и формализации алгоритмических систем. Продукцией называется правило вида

$$\frac{t_1, t_2, \dots, t_n}{t},$$

где  $t_1, t_2, \dots, t_n$  – посылки, а  $t$  – заключение, при этом  $t_1, t_2, \dots, t_n, t$  – слова некоторого алфавита. Вся совокупность правил, хранящаяся в БЗ, называется системой продукций. В простейшем случае и посылки и заключение могут быть простыми высказываниями. Главное отличие продукции от логических формул – они допускают более широкий спектр интерпретации.

В наиболее общем виде под продукцией понимается выражение

$\langle (i), Q, N, P \rightarrow A, D \rangle$ , где:

- $i$  – уникальное имя продукции, с помощью которого продукция выделяется из всего множества (в качестве имени может выступать любая цепочка символов или цифр, но чаще – лексема, отражающая суть (смысл) данной продукции);
- $Q$  – сфера применения данной продукции (такие сферы легко выделяются в системах знаний человека по принципу "полки", например на первой полке хранятся знания о математике, на второй – о физике и т.п.);
- $N$  – условие применимости ядра продукции (если  $N$  принимает значение «истина», то ядро продукции активизируется, в противном случае продукция не может быть применена);
- $P \rightarrow A$  – ядро продукции; допускается форма: «если  $P$ , то  $A$ , иначе  $B$ »; ядра могут быть детерминированными (могут быть либо однозначными, либо альтернативными (связка «или») - нужен свой механизм выбора альтернативы из нескольких возможных (например: если  $P$ , то чаще надо делать  $A1$ , реже  $A2$ )) или недетерминированными (т.е.  $A$  может и не выполняться при истинности  $D$ ; ядро продукции в этом случае интерпретируется: «если  $P$ , то, возможно,  $A$ », а для формирования оценок используется вероятностный или какие-либо другие подходы (например,

«если P, то с вероятностью p реализовать A»;

- D - постусловия продукции, они активизируются лишь в случае, если ядро продукции реализовалось, и описывают действия и процедуры, которые необходимо выполнить после реализации ядра.

Состав продукционной системы, необходимый для вывода на знаниях, задают следующие компоненты:

- база правил - набор правил, используемых как БЗ; фактически это модель предметной области;
- рабочая память – динамическая область памяти, где хранятся предпосылки, касающиеся отдельных задач, а также результаты выводов, получаемых на основе этих предпосылок; фактически это текущее состояние предметной области;
- механизм логического вывода – порядок использования правил БЗ в соответствии с содержимым рабочей памяти (аналог рассуждениям человека).

Таким образом, обобщенный механизм работы продукционной машины можно описать следующим образом:

- задание модели текущего состояния предметной области;
- интерпретация текущего состояния предметной области на рабочую память и выработка вариантов решения;
- выбор по какому-либо способу варианта решения и выдача его на выход системы для пользователя;
- изменение состояния рабочей памяти, путем выполнения действий и процедур, рекомендованных в послесловиях.

Заметим, что при этом управление процессом вывода осуществляется путем изменения состояния рабочей области и не затрагивает базу правил. Цикл управления (выработки вариантов решения) состоит из выполнения четырёх основных функций:

- сопоставление - образец правила из базы правил сопоставляется с имеющимися фактами в рабочей памяти;

- выбор - если в конкретной ситуации могут быть применены сразу несколько правил, то из них выбирается наиболее подходящее по заданному критерию (разрешение конфликтов);
- срабатывание - если образец правила при сопоставлении совпал с какими-либо фактами из рабочей памяти, то соответствующее правило срабатывает (в каждом цикле может срабатывать только одно правило, сработавшие правила больше не рассматриваются);
- действие – рабочая память изменяется путем добавления в неё заключения сработавшего правила, если в правой части содержится ещё и указания на какие-либо действия, то они выполняются настолько, насколько это возможно.

Основных стратегий вывода в продукционной машине две:

I) Прямой вывод (вывод от ситуации к цели – по известным фактам отыскивается заключение, которое следует из этих фактов) в простом варианте состоит из следующих шагов:

- в рабочую память заносятся предпосылки (некоторые исходные условия (явления), для которых необходимо найти некоторые заранее не известные рекомендации);
- выполняется цикл просмотра базы правил и для каждого не сработавшего правила (продукции) производится сопоставление образцов из условной части правила с образцами, хранимыми в рабочей памяти, по следующим правилам:
  - если все необходимые образцы имеются в рабочей памяти, то условная часть правила считается истинной (срабатывание правила), в противном случае – ложной (переход к следующему правилу);
  - если правило срабатывает, то его результирующая часть (заключение) добавляется в рабочую память, при этом правило помечается как сработавшее, а текущий цикл просмотра базы правил завершается и начинается следующий;



- если ни одно правило на данном цикле не сработало, то работа вывода завершается;

- все результаты, которые содержатся в рабочей памяти по останову вывода, в порядке времени записи в рабочую память выдаются пользователю, причем последнее заключение определяет основную рекомендацию, а все предыдущие поясняют ход рассуждений.

Могут быть варианты этой стратегии, например, возвратная стратегия, когда одна и та же продукция применяется до тех пор, пока это требуется, а не только один раз.

Пример №1 прямого вывода на продукциях:

Исходная база правил состоит из следующих правил, пронумерованных в порядке размещения в БЗ:

- 1) если «вода в реке поднимается», то «рыба выходит на мель»;
- 2) если «вода в реке опускается», то «рыба уходит в глубину»;
- 3) если «лето влажное», то «вода в реке поднимается»;
- 4) если «лето сухое», то «вода в реке опускается»;
- 5) если «рыба выходит на мель», то «следует применять легкую блесну»;
- 6) если «рыба уходит в глубину», то «следует применять тяжелую блесну».

Необходимо дать рекомендации по рыбалке, если лето выдалось влажное. Рассмотрим состояние рабочей памяти на каждом шаге работы вывода:

Шаг 0) в рабочую память помещаются исходные условия:

«лето влажное»

Шаг 1) первый просмотр базы правил – срабатывание правила №3:

«лето влажное»

«вода в реке поднимается»

Шаг 2) второй просмотр базы правил – срабатывание правила №1:

«лето влажное»

«вода в реке поднимается»

«рыба выходит на мель»

Шаг 3) третий просмотр базы правил – срабатывание правила №5:

«лето влажное»

«вода в реке поднимается»

«рыба выходит на мель»

«следует применять легкую блесну»

Шаг 4) четвертый просмотр базы правил – нет сработавших правил – останов вывода:

«лето влажное»

«вода в реке поднимается»

«рыба выходит на мель»

«следует применять легкую блесну»

Шаг 5) вывод и интерпретация результатов – рекомендовано «следует применять легкую блесну», т.к. «рыба выходит на мель», т.к. «вода в реке поднимается», т.к. задано «лето влажное».

Пример №2 прямого вывода на продукциях:

Исходная база правил состоит из следующих правил, пронумерованных в порядке размещения в БЗ:

1) если «намерение – отдых» и «дорога ухабистая», то «использовать джип»;

2) если «место отдыха – горы», то «дорога – ухабистая».

Необходимо дать рекомендации по отдыху, если место отдыха – горы.

Рассмотрим состояние рабочей памяти на каждом шаге работы вывода:

Шаг 0) в рабочую память помещаются исходные условия:

«намерение – отдых»

«место отдыха – горы»

Шаг 1) первый просмотр базы правил – срабатывание правила №2:

«намерение – отдых»

«место отдыха – горы»

«дорога – ухабистая»

Шаг 2) второй просмотр базы правил – срабатывание правила №1:

«намерение – отдых»  
«место отдыха – горы»  
«дорога – ухабистая»  
«использовать джип»

Шаг 3) третий просмотр базы правил – нет сработавших правил – останов  
вывода:

«намерение – отдых»  
«место отдыха – горы»  
«дорога – ухабистая»  
«использовать джип»

Шаг 4) вывод и интерпретация результатов – рекомендовано  
«использовать джип», т.к. «дорога – ухабистая», т.к. задано «намерение –  
отдых» и «место отдыха – горы».

II) Обратный вывод (подтверждение заданной цели при определенных  
начальных условиях, т.е. вывод в данном случае управляется целью) в простом  
варианте состоит из следующих шагов:

- в рабочую память заносятся предпосылки (некоторые исходные условия (явления)) и цель, для которой необходимо найти подтверждение;
- выполняется цикл просмотра базы правил и для каждого целевого правила (т.е. правила, в результирующей части которого записана текущая цель) производится сопоставление образцов из условной части этого правила с образцами, хранимыми в рабочей памяти, по следующим правилам:

- если все необходимые образцы целевого правила имеются в рабочей памяти, то условная часть правила считается истинной (подтверждение цели), в противном случае – ложной (переход к следующей цели);

- если выполняется переход к следующей цели, то условная часть (за исключением тех её фрагментов, которые уже находятся в рабочей

памяти) неподтвержденного целевого правила добавляется в рабочую память как новая цель, а текущий цикл просмотра базы правил завершается и начинается следующий;

- если произошло подтверждение текущей цели, то в качестве цели выбирается предыдущая неподтвержденная цель из рабочей памяти;

- если нет изменений в рабочей области или не осталось неподтвержденных целей, то работа вывода завершается;

- все результаты, которые содержатся в рабочей памяти по останову вывода, в порядке времени записи в рабочую память выдаются пользователю, если осталась хоть одна цель, значит подтверждение первоначальной цели на текущих данных невозможно.

Пример №1 обратного вывода на продукциях:

Исходная база правил состоит из следующих правил, пронумерованных в порядке размещения в БЗ:

- 1) если «вода в реке поднимается», то «рыба выходит на мель»;
- 2) если «вода в реке опускается», то «рыба уходит в глубину»;
- 3) если «лето влажное», то «вода в реке поднимается»;
- 4) если «лето сухое», то «вода в реке опускается»;
- 5) если «рыба выходит на мель», то «следует применять легкую блесну»;
- 6) если «рыба уходит в глубину», то «следует применять тяжелую блесну».

Необходимо подтвердить рекомендацию «следует применять легкую блесну», если лето выдалось влажное. Рассмотрим состояние рабочей памяти на каждом шаге работы вывода:

Шаг 0) в рабочую память помещаются исходные условия и цель (цель помечена «?»):

«лето влажное»

«следует применять легкую блесну»?

Шаг 1) первый просмотр базы правил – попытка подтверждения целевого правила №5 (подтверждение невозможно, задание новой цели):

«лето влажное»

«следует применять легкую блесну»?

«рыба выходит на мель»?

Шаг 2) второй просмотр базы правил – попытка подтверждения целевого правила №1 (подтверждение невозможно, задание новой цели):

«лето влажное»

«следует применять легкую блесну»?

«рыба выходит на мель»?

«вода в реке поднимается»?

Шаг 3) третий просмотр базы правил – попытка подтверждения целевого правила №3 (подтверждение возможно, возврат к предыдущей цели):

«лето влажное»

«следует применять легкую блесну»?

«рыба выходит на мель»?

«вода в реке поднимается»

Шаг 4) четвертый просмотр базы правил – попытка подтверждения целевого правила №1 (подтверждение возможно, возврат к предыдущей цели):

«лето влажное»

«следует применять легкую блесну»?

«рыба выходит на мель»

«вода в реке поднимается»

Шаг 5) пятый просмотр базы правил – попытка подтверждения целевого правила №5 (подтверждение возможно, больше целей нет – останов вывода):

«лето влажное»

«следует применять легкую блесну»

«рыба выходит на мель»

«вода в реке поднимается»

Шаг 6) вывод и интерпретация результатов – целей после останова вывода нет – значит «следует применять легкую блесну» подтверждено, остальные значения рабочей памяти показывают ход рассуждений при

подтверждении цели.

Пример №2 обратного вывода на продукциях:

Исходная база правил состоит из следующих правил, пронумерованных в порядке размещения в БЗ:

1) если «намерение – отдых» и «дорога ухабистая», то «использовать джип»;

2) если «место отдыха – горы», то «дорога – ухабистая».

Необходимо подтвердить рекомендации «использовать джип», если место отдыха – горы. Рассмотрим состояние рабочей памяти на каждом шаге работы вывода:

Шаг 0) в рабочую память помещаются исходные условия и цель (цель помечена «?»):

«намерение – отдых»

«место отдыха – горы»

«использовать джип»?

Шаг 1) первый просмотр базы правил – попытка подтверждения целевого правила №1 (подтверждение невозможно, задание новой цели):

«намерение – отдых»

«место отдыха – горы»

«использовать джип»?

«дорога ухабистая»?

Шаг 2) второй просмотр базы правил – попытка подтверждения целевого правила №2 (подтверждение возможно, возврат к предыдущей цели):

«намерение – отдых»

«место отдыха – горы»

«использовать джип»?

«дорога ухабистая»

Шаг 3) третий просмотр базы правил – попытка подтверждения целевого правила №1 (подтверждение возможно, больше целей нет – останов вывода):

«намерение – отдых»

«место отдыха – горы»

«использовать джип»

«дорога ухабистая»

Шаг 4) вывод и интерпретация результатов – целей после останова вывода нет – значит «использовать джип» подтверждено, остальные значения рабочей памяти показывают ход рассуждений при подтверждении цели.

В целом обратный вывод аналогичен прямому, но возникают дополнительные проблемы ограничения конфликтных наборов правил и выбора алгоритма разрешения конфликтов. Есть и другие проблемы, например, оценки условий в условной части. Все эти проблемы решаются по-разному в рамках соответствующих инструментальных систем продукционного типа.

Процесс вывода можно рассматривать как поиск решения в пространстве состояний. Такое пространство может быть задано в виде графа состояний, тогда, в общем виде, различают стратегии поиска в глубину и ширину.

Поиск в глубину основан на полном исследовании одного варианта до изучения других вариантов. Поиск выполняется по несложной методике (см. рисунок 43), например:

- всегда первой исследуется неизвестная левая ветвь дерева состояний;
- если процесс поиска заходит в тупик, то происходит возврат наверх в последний пункт выбора, где имеются неизученные альтернативные варианты движения;
- поиск повторяется для нового варианта;

Поиск в глубину лучше всего работает при решении проблемы, где все пути поиска от вершины дерева до его основания имеют одинаковую длину.

Стратегия поиска в ширину предусматривает переход в первую очередь к вершинам, ближайшим к стартовой вершине (т.е. отстающим от нее на одну связь), затем к вершинам, отстающим на две связи, затем на три и т. д., пока не будет найдена целевая вершина (см. рисунок 44).





существует указаний на то, какая именно дорога приводит к цели.

Обе приведенные выше стратегии поиска предполагают последовательный перебор возможных вариантов, однако поиск будет более эффективным, если некоторый механизм в пунктах выбора сам сможет делать наиболее желательный выбор, как правило такой выбор основывается на эвристиках поиска.

Поскольку в общем случае в производственной системе невозможно построить сразу все пути решения задачи (проблема выбора (активизации) правил), движение в пространстве состояний осуществляется в соответствии с эвристиками, составляющими содержание критериев:

- принцип "стопки книг" - наиболее часто используемые продукции актуализируются первыми; этот принцип целесообразно применять, если продукции относительно независимы друг от друга;
- принцип наиболее длинного условия - первыми выбирается продукция, у которой наиболее "длинное" условие выполнимости ядра; этот принцип целесообразен, когда продукция хорошо структурированы по отношению "частное-общее";
- принцип метапродукций - в систему продукции вводятся специальные правила метапродукции, задачей которых является разрешение ситуаций в условиях конфликтного набора правил;
- принцип приоритетного выбора - на продукции вводятся статические (формируются заранее, чаще всего на основе экспертных оценок) или динамические (вырабатываются в самом процессе функционирования производственной системы) приоритеты.

Популярность производственных моделей определяется следующими факторами:

- подавляющая часть человеческих знаний может быть записана в виде продукции;
- за небольшим исключением удаление или добавление продукции не приводит к изменениям в основных продукциях;

- при необходимости системы продукций могут реализовать любые алгоритмы и, следовательно, способно отражать любое процедурное знание, доступное ЭВМ;
- наличие в продукциях указателей на сферу применения продукций позволяет эффективно организовывать память, сократив время поиска в ней необходимой информации; классификация сфер может быть многоуровневой, что еще более повышает эффективность поиска знаний, т.к. позволяет наследовать информацию о базе знаний;
- при объединении систем продукций и сетевых представлений получают средства, обладающие большой вычислительной мощностью (могут использоваться также композиции с другими представлениями знаний (фреймы, объектно-ориентированные модели));
- естественный параллелизм в системе продукций, асинхронность их реализации делают производственные системы удобной моделью ЭВМ новой архитектуры, в которой идеи параллельности и асинхронности является центральной.

Производственные модели имеют и недостатки:

- при большом числе продукций становится сложной проверка непротиворечивости системы продукций, и это заставляет при добавлении новых продукций тратить много времени на проверку непротиворечивости новой системы;
- из-за присущей системе недетерминированности (неоднозначного выбора выполняемой продукции из множества активизированных продукций) возникают принципиальные трудности при проверке корректности работы системы; считается, что если число продукций достигает тысячи, то вероятность нормальной работы системы невелика, поэтому число продукций, с которым, как правило, работают современные интеллектуальные системы, не превышает тысячи (если их требуется больше, то осуществляется декомпозиция предметной области и задача разбивается на более мелкие).

- производственные модели – неформальные модели, в них нет строгой теории, и в основном используются эвристики; причина неудач создания теории кроется в расплывчатости самого понятия продукции;
- при переходе от статических систем к динамическим (адаптивным или системам производств реального времени, т.е. меняющим состав производств в системе или перестраивающих алгоритм управления выбором производств из фронта готовых производств в зависимости от текущих ситуаций) теряется быстрота срабатывания (решение этой проблемы видят в специальных средствах аппаратной поддержки).

## 8.2. Приобретение знаний

Приобретением знаний называется выявление знаний из источников знаний и преобразование их в необходимую форму, а также перенос в БЗ.

Источниками знаний для систем ИИ могут быть:

- объективизированные знания – знания, переведенные в такую форму (книги, архивные документы, содержимое других баз знаний и т. п.), которая делает их доступными для потребителя;
- экспертные знания - знания, которые имеются у специалистов, но не зафиксированы во внешних по отношению к нему хранилищах; экспертные знания являются субъективными;
- эмпирические знания – знания, которые могут добываться системой ИИ путем наблюдения за окружающей средой (если есть подходящие средства наблюдения); эмпирические знания также являются субъективными.

Ввод в БЗ объективизированных знаний не представляет особой проблемы, а выявление и ввод субъективных (особенно экспертных) знаний достаточно трудны.

Чтобы разработать методологию приобретения субъективных знаний,

получаемых от эксперта, надо понимать две формы репрезентации (представления) знаний:

- первая форма связана с тем, как и в каких моделях хранятся эти знания у человека-эксперта (при этом следует учитывать, что эксперт не всегда осознает полностью, как репрезентированы у него знания);
- вторая форма связана с тем, как инженер по знаниям, проектирующий некоторую систему ИИ, собирается описывать и представлять знания.

Когнитивные структуры знаний характерные для человека:

- представление класса понятий через его элементы (например, понятие "птица" репрезентируется рядом чайка, воробей, скворец, и т.д.);
- представление понятий класса с помощью базового прототипа, отражающего наиболее типичные свойства объектов класса (например, понятие "птица" репрезентируется прототипом нечто с крыльями, клювом, летает и т.п.) представление с помощью признаков (для понятия "птица", например, наличие крыльев, клюва, двух лап, перьев).

Кроме понятий репрезентируются и отношения между ними. В большинстве случаев, отношения между понятиями определяются процедурным способом, а отношения между составляющими понятием (т.е. определяющими структуру понятия) - декларативным способом.

При приобретении знаний важную роль играют так называемое поле знаний, в котором содержатся основные понятия, используемые при описании предметной области, и свойства всех отношений, используемых для установления связей между понятиями. Поле знаний связано с концептуальной моделью проблемной области, в которой еще не учтены ограничения, которые неизбежно возникают при формальном представлении знаний в БЗ. Получение формального представления знаний в БЗ без этапа концептуального описания в поле знаний существенно замедляет процесс формирования БЗ и может привести к ошибкам. Фактически, переход от описания некоторой области в поле знаний к описанию в БЗ аналогичен переходу от концептуальной модели БД к ее логической схеме.

Возможны три режима взаимодействия инженера знаний с экспертом-специалистом:

- протокольный анализ - заключается в фиксации (например, путем записи на диктофон словесных рассуждений эксперта во время решения проблемы и в последующем анализе этой информации; недостатком этого метода является то, что при анализе протоколов инженеру знаний нелегко отделить понятия, важные для включения в словарь предметной области, от тех, которые появились случайно, кроме того, в протоколах обнаруживаются пробелы, когда рассуждение эксперта как бы прерывается и продолжается уже на основе пропущенных шагов вывода (для устранения недостатков требуется проведения интервью);
- игровая имитация профессиональной деятельности - эксперт помещается в ситуации, похожие на те в которых протекает его профессиональная деятельность; наблюдая за его действиями в различных ситуациях, инженер знаний, формирует свои соображения об экспертных знаниях, которые впоследствии могут быть уточнены с экспертом в режиме интервью;
- интервью - инженер знаний ведет с экспертом активный диалог, направляя его в необходимую сторону;

Исходя из выше сказанного, интервью - важнейший метод приобретения знаний. Наиболее известны следующие стратегии проведения интервью:

- разбиение на ступени - эксперту предлагается назвать наиболее важные, по его мнению, понятия предметной области и указать между ними отношения структуризации (отношения типа "род-вид", "элемент-класс", "целое- часть" и т.п.); эти понятия используются на следующем шаге опроса как базовые, а сама стратегия нацелена на создание иерархии понятий предметной области, выделение в понятиях тесно связанных между собой групп (кластеров);
- репертуарная решетка – эта стратегия направлена на выявление характеристических свойств понятий, позволяющих отделять одни

понятия от других, и состоит в предъявлении эксперту троек понятий с предложением назвать признаки для каждой двух понятий, которые отделяли бы их от третьего; т.к. каждое понятие входит в несколько троек, то на основании такой процедуры происходит уточнение объемов понятий и формируются описания понятий, с помощью которых эти понятия могут идентифицироваться в БЗ.

- подтверждение сходства – эта стратегия состоит в том, что эксперту предлагается установить принадлежность каждой пары понятий из предметной области к некоторому отношению сходства (толерантности); для этого эксперту задается последовательность достаточно простых вопросов, цель которых заключается в уточнении того понимания сходства, которое вкладывает эксперт в утверждение о сходстве двух понятий предметной области.

Процесс взаимодействия инженера знаний с экспертом включает три основных этапа:

1) Подготовительный этап.

Данный этап обычно строится по следующей схеме:

- четкое определение инженером знаний задач проектируемой системы (сужение поля знаний) - определение, что на входе и выходе; определение режима работ, консультации, обучение и т.п.;
- выбор экспертов - определение количества экспертов; выбор уровня компетентности (не всегда хорошо выбирать самый высокий уровень сразу); определение способов и возможности заинтересовать экспертов в работе; тестирование экспертов (например, для определения типа личности (положительные типы: инициатор (быстро реагирует на перспективные проблемы, ощущает необходимость решения проблемы с элементами неопределенности), эрудит (наделен исключительной памятью, отличается повышенным вниманием к деталям и стремлением к упорядоченности), диагност (способен к быстрой оценке сильных и слабых сторон решения задачи); конфликтные типы: эстет (стремится

исследовать только проблемы, приводящие к изящным решениям), независимый (стремится только к индивидуальному решению проблем), фанатик (самоотверженно увлечен своей научной проблемой, того же требует и от окружающих));

- знакомство инженера знаний со специальной литературой в предметной области, чтобы не терять суть разговора за дополнительными пояснениями эксперта (правда, иногда «глупые» вопросы могут многое объяснить);
- знакомство инженера знаний и экспертов (инженер знаний должен настроиться на роль «ученика» (а не «экзаменатора»));
- знакомство эксперта с популярной литературой по искусственному интеллекту (необязательный этап);
- попытка инженера знаний создать прототип поля знаний - поле знаний первого приближения по априорным знаниям из литературы.

## 2) Этап установления "общего кода".

Участники интервью должны пытаться сократить «расстояние» между объектом (т. е. исследуемой предметной областью) и инженером знаний, для этого необходимо выработать словарную основу БЗ; уровень детализации; взаимосвязи между понятиями. В процессе извлечения знаний (желательна ведущая роль инженера знаний в диалоге) сначала желательно получить от эксперта поверхностные знания, и только потом переходить к глубинным структурам и более абстрактным понятиям.

## 3) Основной (гносеологический) этап.

На этом этапе происходит выяснение закономерностей, присущих предметной области, условий достоверности и истинности утверждений, структурирование за счет введения отношений и т. п. Данный этап обычно строится по следующей схеме:

- "накачка" поля знаний - в зависимости от предметной области

выбирается способ интервьюирования и выполняется протоколирование рассуждений эксперта (инженер знаний по возможности не должен пока вмешиваться в рассуждения);

- "домашняя работа" - попытка инженера знаний выделить некоторые причинно-следственные связи в рассуждениях эксперта; построение словаря предметной области (возможно, на карточках) и подготовка вопросов к эксперту; желательно построение инженером знаний цепочек от общего к частному (частный факт → обобщенный факт → эмпирический закон → теоретический закон);
- "подкачка поля зрения" - обсуждение с экспертом прототипа поля знаний и домашней работы, а также ответы на вопросы инженера знаний;
- формализация концептуальной модели и построение поля знаний второго приближения (цель - ясная и понятная модель проблемной области).

Для облегчения приобретения знаний от экспертов созданы специализированные программные системы (например, KRITON - использование стратегий репертуарной решетки и разбиения на ступени; SIMER и ДИАПС - основным методом приобретения знаний является автоматизированное интервьюирование эксперта, которое управляется знаниями, приобретенными системой).

### 8.3. Работа с качественными знаниями

При общении или формировании знаний в некоторой предметной области, часто возникают ситуации, когда невозможно точно количественно описать какие-либо объекты или явления (например, на вопрос, «Каков спрос в Минске на двухкомнатные квартиры?», можно получить качественный ответ - «высокий», «низкий», «ниже, чем в Москве» и т.п., который будет более ясным для понимания, чем некоторая количественная характеристика (например,



100000 квартир в год). Для успешной работы с качественно определенными данными необходимо использовать специальный математический аппарат, называемый теорией нечетких множеств, которую в 1965 г. предложил математик Л.А.Заде.

Математическая теория нечетких множеств, предложенная Л.А. Заде, позволяет описывать нечеткие понятия и знания, оперировать этими знаниями и делать нечеткие выводы. Основанная на этой теории новая методология построения компьютерных систем, существенно расширяет области применения компьютеров. Более того, такие системы можно создавать в любой области деятельности человека. Популярность этого математического аппарата привела в последнее время к созданию микропроцессоров, выполняющих операции над нечеткими множествами, нечетких компьютеров и программных систем (в том числе и систем ИИ).

Например, требуется определить нечеткие понятия степени успеваемости студентов: «отлично», «хорошо», «удовлетворительно» и «неудовлетворительно». С позиции точной количественной оценки, студент учится «хорошо», если его средний балл равен 4. Можно подойти к этому определению с другой позиции, и считать, что студент учится «хорошо», если его средний балл находится в некотором диапазоне (например, от 3.2 до 4.8). Однако, при приближении к краям этого интервала, наша оценка будет иметь определенную долю сомнения (действительно, студент со средним баллом 3.2 очень мало «похож» на «хорошиста»). Точно также можно оценить и другие категории успеваемости. Наиболее просто представить эти рассуждения графически (см. рисунок 45), где  $U$  – шкала оценок, а ось  $\mu_A$  – степень принадлежности (в процентах от 0 до 100). Тогда для определения, к какому классу успеваемости может относиться студент, остается рассчитать его средний балл и провести прямую параллельную оси  $\mu_A$  из этой точки. Пересечение этой прямой с кривой каждого класса и даст искомый процент соответствия (для приведенного примера, студент со средним баллом 2.75 будет отнесен к классу «удовлетворительно» с большей степенью, чем к классу

«неудовлетворительно»).

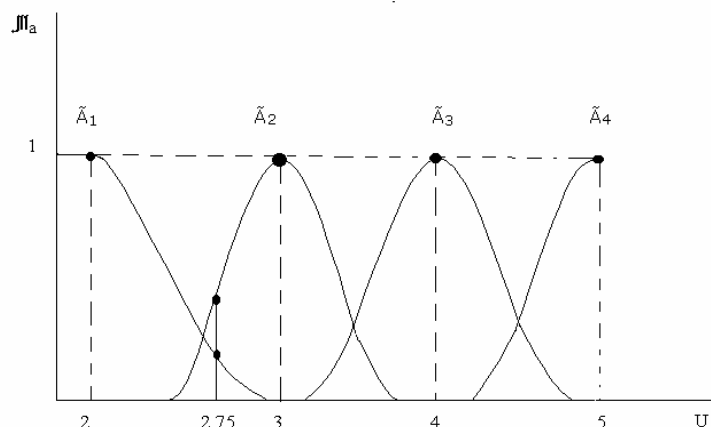


Рисунок 45. Функции принадлежности.

Формально, нечетким множеством  $\tilde{A}$  на множестве  $U$  называется совокупность пар

$$\tilde{A} = \langle \mu_A(u), u \rangle,$$

где  $\mu_A(u)$  - функция принадлежности нечеткого множества  $\tilde{A}$ , а  $u$  - носитель нечеткого множества  $\tilde{A}$ . Функция принадлежности  $\mu_A(u) \in [0,1]$ , т.к. удобнее выразить свое отношение уверенности в процентах (в данном случае абсолютная уверенность задается при  $\mu_A(u) = 1$  (т.е. 100%), а абсолютная неуверенность -  $\mu_A(u) = 0$ ).

Неопределенность знаний в системах ИИ задается с помощью лингвистических моделей, основанных на теории лингвистических переменных и теории приближенных рассуждений. Эти теории опираются на понятие нечеткого множества, систему операций над нечеткими множествами и методы построения функций принадлежности. Одним из основных понятий, используемых в лингвистических моделях, является понятие лингвистической переменной, значениями которой могут являться не числа, а слова или предложения некоторого искусственного либо естественного языка. Например, лингвистическая переменная "успеваемость"

$$X = \{\tilde{A}_1, \tilde{A}_2, \tilde{A}_3, \tilde{A}_4\}$$

может принимать четыре возможных значения:

- $\tilde{A}_1$  = «неудовлетворительно»;
- $\tilde{A}_2$  = «удовлетворительно»;
- $\tilde{A}_3$  = «хорошо»;
- $\tilde{A}_4$  = «отлично».

Для того чтобы работать с лингвистической переменной  $X$ , необходимо количественно определить все нечеткие множества  $\tilde{A}_i$  с помощью задания соответствующих функций принадлежности (см. рисунок 45). Для приведенного выше примера по успеваемости, для каждого класса успеваемости носителем нечеткого множества будет соответствующий средний балл (2, 3, 4, 5), а график для каждого класса – соответствующая функция принадлежности. Графики функций принадлежности пересекаются, потому что мы затрудняемся сказать точно, к какому нечеткому значению следует отнести определенный средний балл.

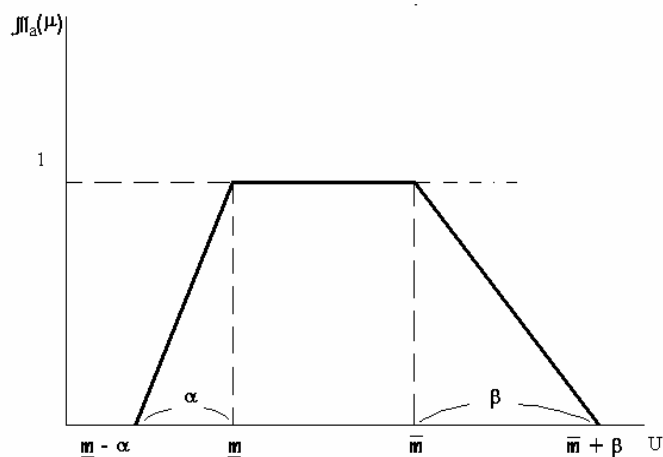


Рисунок 46. Упрощенная функция принадлежности.

В большинстве случаев функции принадлежности строятся субъективно по результатам опроса экспертов, поэтому они являются в некотором смысле «приближенными», т.е. не абсолютно адекватно отражающими явление или

объект, поэтому для упрощения вычислений выбирают такое описание функции принадлежности, для которого проще всего выполняются расчеты. Наиболее распространенный подход – выбор трапециевидной функции (см. рисунок 46), которая характеризуется четверкой значений  $(\underline{m}, \bar{m}, \alpha, \beta)$ .

Для работы с нечеткими данными можно задать нечеткие переменные  $(X)$ , значения которых определяются заданием функции принадлежности. Функция принадлежности для таких переменных может быть описана не только графически, а например, через совокупность пар вида  $(x, \mu_A(x))$ , где  $x \in U$  ( $U$  – базовая шкала):

$$X = \sum_{i=1}^n \frac{x_i}{\mu(x_i)};$$

Для работы с нечеткими множествами определены операции, которые подобны операциям с обычными множествами, но результатом их является нечеткое множество, например:

- объединение:  $\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$  или
 
$$\mu_{A \cup B} = 1 \text{ при } \mu_A(x) + \mu_B(x) \geq 1;$$

$$\mu_{A \cup B} = \mu_A(x) + \mu_B(x) - \text{иначе.}$$
- пересечение:  $\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$  или  $\mu_{A \cap B}(x) = \mu_A(x) * \mu_B(x)$ ;
- дополнение:  $\mu_{A'}(x) = 1 - \mu_A(x)$ ;
- разность:  $\mu_{A \setminus B}(x) = \mu_A(x) - \mu_B(x)$  при  $\mu_A(x) \geq \mu_B(x)$ ,
 
$$\mu_{A \setminus B}(x) = 0 - \text{иначе;}$$
- декартово произведение:  $\mu_A(x) = \min\{\mu_{A1}(x_1), \dots, \mu_{An}(x_n)\}$ ;
- нормализация:  $\mu_B(x) = \mu_A(x) / \max \mu_A(x)$ ;

Для упрощенного трапециевидного описания, нечеткие множества задаются четверкой значений (см. рисунок 46):

$$X_1 = (\underline{m}_1, \bar{m}_1, \alpha_1, \beta_1),$$

$$X_2 = (\underline{m}_2, \bar{m}_2, \alpha_2, \beta_2).$$

Для такого упрощенного описания также определены соответствующие операции, например, арифметические операции задаются в следующем виде:

- сложение  $X = X_1 + X_2$  :  $\underline{m} = \underline{m}_1 + \underline{m}_2$ ,  $\overline{m} = \overline{m}_1 + \overline{m}_2$ ,  $\alpha = \alpha_1 + \alpha_2$ ,  $\beta = \beta_1 + \beta_2$ ;
- вычитание  $X = X_1 - X_2$  :  $\underline{m} = \underline{m}_1 - \underline{m}_2$ ,  $\overline{m} = \overline{m}_1 - \overline{m}_2$ ,  $\alpha = \alpha_1 + \beta_2$ ,  $\beta = \beta_1 + \alpha_2$ ;
- умножение  $X = X_1 * X_2$  :  $\underline{m} = \underline{m}_1 * \underline{m}_2$ ,  $\overline{m} = \overline{m}_1 * \overline{m}_2$ ,

$$\alpha = \underline{m}_1 * \underline{m}_2 - (\underline{m}_1 - \alpha_1) * (\underline{m}_2 - \alpha_2),$$

$$\beta = (\overline{m}_1 + \beta_1) * (\overline{m}_2 + \beta_2) - \overline{m}_1 * \overline{m}_2;$$

- деление  $X = X_1 / X_2$  :  $\underline{m} = \underline{m}_1 / \underline{m}_2$ ,  $\overline{m} = \overline{m}_1 / \overline{m}_2$ ,

$$\alpha = (\underline{m}_1 * \beta_2 + \overline{m}_2 * \alpha_1) / (\overline{m}_2^2 + \overline{m}_2 * \beta_2),$$

$$\beta = (\underline{m}_2 * \beta_1 + \overline{m}_1 * \alpha_2) / (\underline{m}_2^2 - \underline{m}_2 * \alpha_2);$$

Для выполнения нечетких операций над знаниями большое значение получили следующие операции:

1) декартово произведение  $R = A \times B$ , которое отражает результат прямого перемножения нечетких множеств. В логике нечётких отношений произведение нечётких множеств моделирует продукцию вида "если А, то В", или  $A \rightarrow B$ . Получение произведения  $R$  выполняется следующими действиями, которые рассмотрим на приведенном ниже примере.

Пусть имеется множество  $A$  – маложирные молочные продукты (1 – сыворотка, 2 – молоко, 3 – творог, 4 – сливки), для которого определено нечеткое множество

$$A = 1 / 1,0 + 2 / 0,8 + 3 / 0,6 + 4 / 0,2.$$

Также имеется множество  $B$  – жирные молочные продукты (1 – сыр, 2 – сметана, 3 – масло, 4 – топленое масло), для которого определено нечеткое множество

$$B = 1 / 0,1 + 2 / 0,5 + 3 / 0,8 + 4 / 1.$$

Зададим продукцию «если  $A$  – маложирные, то  $B$  – очень жирные».

$R = A \times B$  будет представлять матрицу с элементами  $\mu_R(A_i, B_j)$ . Строки матрицы образуются следующим образом:

- для первой строки берется  $\mu_A$  ("числитель") первого элемента из  $A$  и поочередно сравнивается с каждым  $\mu_B$  ("числителем") из множества  $B$ ,

меньшее значение из сравниваемой пары заносится в соответствующий элемент строки;

- для второй строки берется  $\mu_A$  второго элемента из  $A$  и сравнивается с каждым  $\mu_B$  из  $B$ ;
- процесс повторяется для каждого элемента из  $A$ .

Полученная матрица будет иметь вид:

A\B	1	2	3	4
1	0,1	0,5	0,8	1,0
2	0,1	0,5	0,8	0,8
3	0,1	0,5	0,6	0,6
4	0,1	0,2	0,2	0,2

Таким образом, отношение  $R = A \times B$  выполняет операцию  $A \rightarrow B$ , т.е. отображает нечёткое отношение из  $A$  в  $B$ . Интерпретация, например, элемента (1,1) такой матрицы звучит как «если сыворотка – маложирный продукт, то сыр – очень жирный со степенью уверенности 0,1», а для элемента (1,4) – «если сыворотка – маложирный продукт, то топленое масло – очень жирный со степенью уверенности 1,0».

2) Для определения операции  $R \rightarrow S$  над матрицами нечётких отношений задана операция свёртки (композиции), которая обозначается  $R \circ S$ . Данная операция позволяет промоделировать операцию  $A \times B \rightarrow B \times C$ , или определить нечёткое отношение из  $A$  в  $C$  (или  $A \rightarrow C$ ) путем использования только двух продукций  $A \rightarrow B$  и  $B \rightarrow C$ . В левой части этого выражения стоит матрица  $R$  с элементами  $r_{ij} = \mu_R(A_i, B_j)$ , в правой части – матрица  $S$  с элементами  $s_{ij} = \mu_S(B_i, C_j)$ . Результатом свёртки будет матрица  $P = R \circ S$  с элементами  $p_{ij} = \mu_P(\mu_{Ri}, \mu_{Sj})$ . Алгоритм получения матрицы  $P$  – классический алгоритм произведения матриц (строка на столбец), при этом из каждой пары выбирается меньшая величина, а затем из этих меньших берется наибольшая, например:

$$p_{11} = \max \{ \min(r_{11}, s_{11}), \min(r_{12}, s_{21}), \min(r_{13}, s_{31}), \min(r_{14}, s_{41}) \},$$

...

$$p_{32} = \max \{ \min(r_{31}, s_{12}), \min(r_{32}, s_{22}), \min(r_{33}, s_{32}), \min(r_{34}, s_{42}) \},$$

...

Данную операцию рассмотрим на следующем примере.

Пусть А – множество жителей разных регионов (1 - индус, 2 - русский, 3 - ненец, 4 – африканец), для которого определено нечеткое множество «южане»:

$$F = 1/0,9 + 2/0,6 + 3/0,0 + 4/1,0.$$

Пусть также дано В – множество различных типов климата (1 - умеренный, 2 - теплый, 3 - субтропики, 4 – тропики), для которого определено нечеткое множество «жаркий климат»:

$$G = 1/0,3 + 2/0,5 + 3/0,9 + 4/1,0.$$

Для воспроизведения правила  $(F \rightarrow G)$  "если человек – южанин, то он живет в жарком климате", необходимо выполнить операцию произведения  $R = F \times G$ .

Полученная матрица R будет иметь вид:

F\G	1	2	3	4
1	0,3	0,5	0,9	0,9
2	0,3	0,5	0,6	0,6
3	0,0	0,0	0,0	0,0
4	0,3	0,5	0,9	1,0

Полученные результаты можно трактовать следующим образом, например, для индуса ( $F = 1$ ): 0,3 – его склонность пожить в умеренном климате ( $G = 1$ ), 0,5 – в теплом ( $G = 2$ ), 0,9 – в субтропиках ( $G = 3$ ) и 0,9 – в тропиках ( $G = 4$ ), а ненец ( $F = 3$ ) не хочет жить ни в одном из указанных климатов.

Пусть также дано С – множество различных типов зданий по расчетной минимальной температуре (1 – до  $-50^{\circ}\text{C}$ , 2 - до  $-20^{\circ}\text{C}$ , 3 - до  $-10^{\circ}\text{C}$ , 4 – до  $0^{\circ}\text{C}$ ), для которого определено нечеткое множество «здание для жаркого климата»:

$$H = 1/0,0 + 2/0,2 + 3/0,7 + 4/1,0.$$

Тогда определение отношения  $(G \rightarrow H)$  «если климат - жаркий, то строить здание для жаркого климата» отобразится произведением  $S = G \times H$ :

G\H	1	2	3	4
1	0,0	0,2	0,3	0,3
2	0,0	0,2	0,5	0,5
3	0,0	0,2	0,7	0,9
4	0,0	0,2	0,7	1,0

Свертка отношений R и S позволяет определить, как относятся южане (F) к зданиям для жаркого климата (H) или  $F \rightarrow H$ :

$$R \circ S = \begin{pmatrix} 0,3 & 0,5 & 0,9 & 0,9 \\ 0,3 & 0,5 & 0,6 & 0,6 \\ 0,0 & 0,0 & 0,0 & 0,0 \\ 0,3 & 0,5 & 0,9 & 1,0 \end{pmatrix} \circ \begin{pmatrix} 0,0 & 0,2 & 0,3 & 0,3 \\ 0,0 & 0,2 & 0,5 & 0,5 \\ 0,0 & 0,2 & 0,7 & 0,9 \\ 0,0 & 0,2 & 0,7 & 1,0 \end{pmatrix} =$$

FH	1	2	3	4
1	0,0	0,2	0,7	0,9
2	0,0	0,2	0,6	0,6
3	0,0	0,0	0,0	0,0
4	0,0	0,2	0,7	1,0

Полученные результаты можно трактовать следующим образом, например, для индуса ( $F = 1$ ): 0,0 – это возможность использовать здание типа 1, 0,2 – типа 2, 0,7 – типа 3 и 0,9 – типа 4, а ненец ( $F = 3$ ) не может жить в здании южного типа.

Нечёткий вывод, применяемый в нечетких системах ИИ, чаще всего основан на правиле заключения (*modus ponens*) и в его основе лежат операции нечеткого перемножения и свертки, выполняемы следующим образом:

1) Вначале определяется нечёткое отношение  $A \rightarrow B$  и формируется матрица  $R = A \times B$ .

2) Далее следует свёртка  $S = A' \circ R$ , где  $A'$  – нечеткое множество.

Заключение, полученное в результате этих операций, также будет нечетким и в общем случае будет иметь вид:



$$\frac{A', A \rightarrow B'}{B''}$$

В настоящее время нечеткая логика заняла важное место в ИИ, сделавшись самостоятельным направлением – нечёткие экспертные системы, нечеткие алгоритмы, нечёткие системы принятия решений. Существует общая тенденция к переопределению известных алгоритмов, в том числе вычислительных (при этом точные вычисления рассматриваются как вырожденный случай) через нечёткую алгебру.

Основные недостатки нечеткой логики в системах ИИ:

- субъективность нечётких оценок;
- отсутствие методов проверки нечётких знаний на непротиворечивость;
- трудности с получением точных результатов, когда это необходимо.

## 9. ЯЗЫКИ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Для автоматизации построения систем ИИ характерны следующие направления:

- разработка систем ИИ путем прямого использования широко распространенных языков обработки символьной информации;
- расширение базисных языков ИИ до систем представления знаний за счет использования специализированных библиотек и пакетов;
- создание языков представления знаний, специально ориентированных на поддержку определенных формализмов, и реализация соответствующих трансляторов с этих языков.

В начальном этапе развития ИИ (до начала 1960-х гг.) использовались не специализированные универсальные языки программирования, недостатки которых в задачах ИИ (в основном сложность и трудоемкость разработки) были столь велики, что реализация систем ИИ была практически невозможной. Поэтому для устранения указанных недостатков (а также для реализации теоретических положений ИИ, которые разрабатывались параллельно), были разработаны языки и системы обработки символьной информации общего назначения для задач ИИ:

1) ЛИСП (LISP) – разработан в Стенфорде (Дж. Маккарти) в начале 1960-х гг., предназначался только для автоматизации программирования (замена ФОРТРАНа), но постепенно были сформированы основные принципы, которые привели язык к успеху (LISP 1.5 – 1978 г.) в области создания систем ИИ:

- использование единого спискового представления для программ и данных;
- применение выражений для определения функций;
- скобочный синтаксис языка.

Язык LISP реализован для большинства классов ЭВМ в виде современных диалектов, а также созданы специализированные LISP-машины. Дальнейшее развитие языка идет по пути его стандартизации (Standard Lisp,

Common Lisp), а также использования в качестве среды функционирования для построения концептуально новых языков.

2) СНОБОЛ (SNOBOL) – язык обработки строк, в рамках которого была впервые реализована концепция поиска по образцу. Данный язык был одной из первых практических реализаций развитой продукционной системы (СНОБОЛ-IV, 1980 г.).

3) РЕФАЛ – создан в ИПМ АН СССР (1968 г.). В основы языка положено понятие рекурсивной функции, определенной на множестве произвольных символьных выражений. Базовой структурой данных этого языка являются двунаправленные списки. В языке обработка символов задана ближе к продукционным представлениям, используется концепция поиска по образцу, характерная СНОБОЛу. В настоящее время используется уже третье поколение языка. Использование – для автоматизации построения трансляторов, построения систем аналитических преобразований, в качестве инструментальной среды для реализации языков представления знаний.

4) ПРОЛОГ (PROLOG) – разработан в Марсельском университете (1971 г.). Данный язык предлагает такую парадигму мышления, в рамках которой описание решаемой задачи представляется в виде слабо структурированной совокупности отношений (это удобно, если число отношений не слишком велико и каждое отношение описывается небольшим числом альтернатив, иначе программа становится сложной для понимания и модификации). Популярность языка связана с математическим обоснованием логических основ языка, кроме того, ПРОЛОГ был выбран в качестве базового для машины вывода в японском проекте машин V поколения.

Работы в области языков общего назначения для задач ИИ, рассмотренных выше, сыграли важную роль в создании:

- языков, ориентированных на программирование поисковых задач – примеры таких языков - PLANNER, CONNIVER – надстройка над LISP-средой; для них характерно, кроме свойств LISP, представление данных в

виде произвольных списковых структур, развитые методы сопоставления образцов, поиск с возвратами и вызов процедур по образцу;

- языков представления знаний (ЯПЗ) первого поколения – например, KRL, FRL, KL-ONE, для этих языков характерными чертами были двухуровневое представление данных (абстрактная модель предметной области в виде иерархии множеств понятий и конкретная модель ситуации как совокупность взаимосвязанных экземпляров этих понятий), представление связей между понятиями и закономерностей предметной области в виде присоединенных процедур, семантический подход сопоставлению образцов и поиску по образцу.

Современные требования к ЯПЗ (необходима разумная компромиссная реализация данных требований):

- наличие простых и вместе с тем достаточно мощных средств представления сложно структурированных и взаимосвязанных объектов;
- возможность отображения описаний объектов на разные виды памяти ЭВМ;
- наличие гибких средств управления выводом, учитывающих необходимость структурирования правил работы механизма вывода;
- прозрачность системных механизмов для программиста, предполагающая возможность их доопределения и переопределения на уровне входного языка;
- возможность эффективной реализации;

## 9.1. РЕФАЛ

Название языка РЕФАЛ происходит от "РЕкурсивных Функций АЛгоригми-ческий язык".

Разработчики языка Рефал делят алгоритмические языки на две группы:

- 1) Языки операторного, или процедурного типа.

Для этих языков элементарными единицами программы являются операторы, т.е. приказы, выполнение которых сводится к четко определенному изменению четко определенной части памяти машины. Типичным представителем этой группы является язык машины Поста. Сюда же относятся машинные языки конкретных ЭВМ, а также массовые языки программирования (например, Фортран, Алгол, ПЛ/1). В естественных языках прообразом операторных языков является повелительное наклонение (императив, приказание).

2) Языки декларативного типа (или сентенциальные языки, от sentence — высказывание, предложение).

Программа на таком языке представляется в виде набора предложений (соотношений, правил, формул), которые машина, понимающая данный язык, умеет каким-то образом применять к обрабатываемой информации. Простейшим примером сентенциального языка, созданного с теоретическими целями является язык нормальных алгоритмов Маркова. В естественных языках прообразом для сентенциальных языков является изъявительное наклонение (описание, повествование), наиболее распространенное и по сути дела составляющее основу языка.

Язык РЕФАЛ является сентенциальным в своей основе, а вся информация в этом языке представляется в виде правил конкретизации. Каждое правило записывается в виде предложения, которое представляет собой продукцию с определенными синтаксисом и семантикой. Предложения в Рефал-программе отделяются друг от друга знаком § (параграф).

Каждое правило конкретизации определяет раскрытие смысла некоторого понятия через элементарные понятия. Операцию конкретизации можно также определить как переход от имени к значению. Введены два знака:  $k$  и  $\perp$ , которые называются конкретизационными скобками, и которые будут содержать объект, подлежащий конкретизации. Так, если  $x$  — некоторая переменная, то  $kx\perp$ . (конкретизация  $x$ ) будет изображать значение этой величины. Другой пример: объект  $k28+7\perp$  при правильном определении

операции сложения будет заменен на объект 35 ( $28 + 7 = 35$ ).

Выполнение конкретизации — переход от имени к значению — объявляется основной и, по существу, единственной операцией в языке Рефал. Эту операцию будет выполнять Рефал-машина (машина на логическом уровне, имитируемая соответствующим транслятором на универсальной ЭВМ). Поскольку правило конкретизации есть указание для замены одного объекта (слова в некотором алфавите) на другой, предложения языка Рефал должны состоять из левой части (заменяемый объект) и правой части (объект, заменяющий левую часть). Для разделения правой и левой части используется знак стрелки " $\rightarrow$ ". Например, предложение, выражающее тот факт, что значение переменной X есть 137, записывается в виде

$\S kX\perp\rightarrow 137$

Между знаком  $\S$  и первым знаком k можно вставлять последовательность знаков, которая будет служить номером предложения, или комментарием к нему, например:

$\S 1.1 kX\perp\rightarrow 137$

Рефал-машина, которая, используя предложения Рефал-программы, будет выполнять конкретизации, может быть описана следующим образом:

- объектом обработки является некоторое выражение (слово), которое находится в поле зрения машины;
- работа машины осуществляется по шагам, каждый из которых представляет выполнение одного акта, конкретизации:

- если программа машины состоит из единственного предложения, например,  $\S 1.1 kX\perp\rightarrow 137$ , а в поле зрения находится выражение  $kX\perp$ , тогда за один шаг машина заменит содержимое поле зрения на 137, после чего она остановится, т. к. знаков конкретизации больше нет;

- т.к. Рефал-программа содержит, набор (последовательность) предложений, может оказаться, что для выполнения данной конкретизации пригодно не одно, а несколько предложений (например, в поле памяти, кроме  $\S 1.1 kX\perp\rightarrow 137$ , может находиться еще предложение  $\S$

1.2  $kX\perp\rightarrow 274$ ); неоднозначность, которая отсюда может возникнуть, устраняется следующим образом: машина просматривает предложения в том порядке, в котором они расположены в Рефал-программе, и применяет первое из них, которое окажется подходящим.

- поле зрения может содержать сколько угодно конкретизационных скобок, причем они могут быть как угодно вложены друг в друга, в этом случае Рефал-машина начинает процесс конкретизации с первого из знаков  $k$ , в области действия которого (т.е. в последовательности знаков до парной скобки  $\perp$ ) нет ни одного знака  $k$ ; выражение, находящееся в области этого знака  $k$ , последовательно сравнивается с левыми частями предложений Рефал-программы, после того как будет найдено подходящее предложение, машина выполняет в поле зрения необходимую замену и переходит к следующему шагу конкретизации.

Пример выполнения Рефал-программы. Пусть дана программа следующего вида:

$kX\perp\rightarrow 137$

$kX\perp\rightarrow 274$

$kY\perp\rightarrow 2$

$k137+2\perp\rightarrow 139,$

а поле зрения содержит выражение

$kkX\perp+kY\perp\perp.$

Выполнение программы по шагам:

Шаг 1) Замене подлежит подвыражение  $kX\perp$  — получим в поле зрения  $k137 + kY\perp\perp.$

Шаг 2) Конкретизируется  $kY\perp$  — получим в результате применения третьего предложения  $k137 + 2\perp$

Шаг 3) Получим 139, не содержащее символов  $k$ , что означает останов машины.

Чтобы иметь возможность представлять обобщенные предложения, используются три типа переменных:

- $e$  — для представления выражений;
- $t$  — для представления термов;
- $s$  — для представления символов.

В простейшем случае переменные записываются в виде указателя типа ( $e$ ,  $t$ ,  $s$ ) и индекса; например,  $e_1$ ,  $e_2$  — переменные, пробегающие в качестве значений выражения.

Выражением в языке Рефал называется последовательность знаков, правильно построенная в соответствии с синтаксисом языка Рефал.

Терм языка Рефал — это либо символ, либо выражение в круглых или конкретизационных скобках. Выражения строятся из термов.

Пример, требуется написать программу, которая выполняет раскрытие скобок в алгебраических выражениях, построенных из букв с помощью скобок, знаков сложения "+" и умножения "\*". Решение данной задачи можно сформулировать так:

1) Если некоторое выражение  $e$  имеет вид  $e_1 + e_2$ , где  $e_1$ ,  $e_2$  — выражения, то для раскрытия скобок надо:

- раскрыть скобки в  $e_1$ ,
- раскрыть скобки в  $e_2$ ,
- полученные результаты сложить.

Это решение может быть записано в виде предложения:

$$\S 2.1 \ ke_1 + e_2 \perp \rightarrow ke_1 \perp + ke_2 \perp$$

2) Если же выражение  $e$  имеет вид  $e_1 * e_2$ , то, необходимо учитывать две возможности:

- хотя бы один из сомножителей есть сумма (например,  $e = (A + B) * C$ ),
- ни одно из выражений  $e_1$  или  $e_2$  не представимо в виде суммы (например,  $e = (A * B) * (C * L)$ ).



В первом случае надо описать законы дистрибутивности:

$$\S 2.2 k e_1 * (e_2 + e_3) \perp \rightarrow k e_1 * e_2 \perp + k e_1 * e_3 \perp,$$

$$\S 2.3 k (e_1 + e_2) * e_3 \perp \rightarrow k e_1 * e_3 \perp + k e_2 * e_3 \perp,$$

$$\S 2.4 k e_1 * (e_2 + e_3) * e_4 \perp \rightarrow k (e_1 * e_2 + e_1 * e_3) * e_4 \perp.$$

Во втором случае по аналогии со сложением имеем

$$\S 2.5 k e_1 * e_2 \perp \rightarrow k e_1 \perp * k e_2 \perp.$$

Осталось выразить возможность "снятия внешних скобок" и условие "терминальности" символов, что определяют предложения:

$$\S 2.6 k (e) \perp \rightarrow k e \perp,$$

$$\S 2.7 k s \perp \rightarrow s$$

(буквы не подлежат конкретизации).

Приведенные семь предложений § 2.1 - § 2.7 полностью решают данную задачу.

Пример, обработки на данной программе выражения

$$k(A + B) * (C + D) \perp$$

Последовательно получим в результате работы программы (слева указан номер правила, которое непосредственно привело к данному выражению):

$$\S 2.2 k(A + B) * C \perp + k(A + B) * D \perp$$

$$\S 2.3 k A * C \perp + k B * C \perp + k(A + B) * D \perp$$

$$\S 2.3 k A * C \perp + k B * C \perp + k A * D \perp + k B * D \perp$$

Далее ограничимся рассмотрением первого слагаемого:

$$\S 2.5 k A \perp * k C \perp + \dots$$

$$\S 2.7 A * k C \perp + \dots$$

$$\S 2.7 A * C + \dots$$

После аналогичной обработки остальных слагаемых получим искомое выражение

$$A * C + D * C + A * D + B * D.$$

Пример, обработки на данной программе выражения

$$kA + (B + C) \perp$$

Последовательно получим в результате работы программы (слева указан номер правила, которое непосредственно привело к данному выражению):

$$\S 2.1 kA \perp + k(B + C) \perp,$$

$$\S 2.7 A + k(D + C) \perp,$$

$$\S 2.6 A + kB + C \perp,$$

$$\S 2.1, 2.7 A + B + C.$$

Заметим, что если расположить правило  $\S 2.5$  перед правилами  $\S 2.2$  и  $\S 2.3$  в программе, то это приведет к ошибке, например, выражение  $A *(B+C)$  будет приведено к виду:  $A *B + C$ .

## ЛИТЕРАТУРА

1. ANSI X3.135—1992, American National Standard for Information Systems — Database Language — SQL, November, 1992.
2. Astrahan M.M., System R: A Relational Approach to Data Base Management //ACM Transactions on Data Base Systems. — 1976. — V1, 97, June.
3. Boyce R.F., Chamberlin D.D., King W.F., Hammer M.M. Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage //Communications ACM. — 1975. V.18, November. — P.621.
4. Chamberlin D.D., Gray J.N., Traiger L.L. Views, Authorization and Locking in a Relational Data Base System //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. — 1975.
5. Chamberlin D.D., Raymond F.B. SEQUEL: A Structured English Query Language. //Proc. ACM—SIGMOD. — 1974. — Workshop, Ann Arbor, Michigan, May.
6. Codd E.F. A Data Base Sublanguage Founded On The Relational Calculus //Proc. ACM—SIGFIDET/ — 1971. — Workshop, San Diego, Calif., Nov. P.35—68.
7. Codd E.F. Extending the Database Relation Model to Capture More Meaning. //ACM Transaction on Database Systems. 1979.— V.4, №4. — P.397—434. (Имеется перевод: Кодд Э.Ф. Расширение реляционной модели для лучшего отражения семантики //СУБД. — 1996. — №5—6. — С.163—192.)
8. Codd E.F. Further Normalization of the Data base Relational Model //Data Base Systems.— N.J.: Prentice—Hall, 1972. — P.33—64.
9. Codd E.F. Normalized Data Base Structure: A Brief Tutorial //Proc. of 1971 ACM—SIGFIDET Workshop on Data Description, Access and Control.— N.—Y.: ACM. — 1971. — P.1—17.
10. Codd E.F. Recent investigations in relational data base systems //Proc. IFIP Congress. — 1974. — North—Holland Pub. Co., Amsterdam. — P.1017—

- 1021.
- 11.Codd E.F. Relation Model of Data for Large Shared Data Banks //Comm. ACM. — 1970. — V.13, №.6. — P.377—383. (Имеется перевод: Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных //СУБД. — 1995. — №1. — С.145—160.)
  - 12.Eswaran K.P. Chamberlin D.D. Functional specifications of a subsystem for data base integrity //Proc. Very Large Data Base Conf., Framingham, Mass., Sept. — 1975. — P.48—68.
  - 13.Eswaran K.P., Gray J.N., Lorie R.A., Traiger I.L. The Notions of Consistency and Predicate Locks in a Data Base System //CACM. — 1976. — V.19, №11.
  - 14.Fagin R. Multivalued Dependencies and New Normal Form for Relational Databases //ACM TODS. — 1977. — V.2, №3.
  - 15.Fagin R.A. Normal Form for Relational Databases That is Based on Domains and Key //ACM Transactions on Database Systems. — 1981. — V.6, №3. — P.387—415.
  - 16.Gray J., Lorie R., Putzolu G., Traiger I. Granularity of Locks and Degrees of Consistency in a Shared Data Base //in Readings in Database Systems, Second Edition, Chapter 3, Michael Stonebraker, Ed., Morgan Kaufmann. — 1994.
  - 17.Heath I.J. Unacceptable File Operations in Relational Database //Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access, and Control. — San Diego, Calif. — 1971.
  - 18.Held G.D., Stonebraker M.R., Wong E. INGRES: A Relational Data Base System //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. — 1975.
  - 19.Meiton J., Simon A.R. Understanding The New SQL: A Complete Guide //Morgan Kaufmann. — 1993.
  - 20.Reisner P., Boyce R.F., Chamberlin D.D. Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL //Proceedings of AFIPS National ComputerConference, Anaheim, CA, May. — 1975.
  - 21.Smith J.M., Smith D.C. Database Abstractions: Aggregation and

- Generalization. //ACM Transactions on Database Systems. — 1977. — V.2, №2, June.— P.105—133. (Имеется перевод: Смит Д.М., Смит Д.К. Абстракции баз данных: Агрегация и обобщение //СУБД. — 1996. — №2. — С.141—160.)
- 22.Zloof M.M. Query By Example //Proceedings of AFIPS National Computer Conference, Anaheim, CA, May. — 1975.
- 23.Алиев Р.А., Абдикеев Н.М. и др. — Производственные системы с искусственным интеллектом, — М.: Радио и связь, 1990.—263 с.
- 24.Амамия М. Танака Ю. Архитектура ЭВМ и искусственный интеллект: Пер. с япон. — М.: Мир, 1993. — 400 с.
- 25.Атре Ш. Структурный подход к организации баз данных. — М.: Финансы и статистика, 1983. — 320 с.
- 26.Беренсон Х., Бернштейн Ф., Грэй Д., Мелтон Д., О'Нил Э., О'Нил П. Критика уровней изолированности в стандарте ANSI SQL //СУБД. — 1996. — №2. — С.45—60.
- 27.Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. — М.: Финансы и статистика, 1989. — 351 с.
- 28.Болотова Л.С., Комаров М.А., Смольянинов А.А. Системы искусственного интеллекта. Теоретические основы СИИ и формальные модели представления знаний: Учебное пособие/МИРЭА — М.: 1998. — 108 с.
- 29.Боуман Д, Эмерсон С., Дарновски М. Практическое руководство по SQL. — Киев: Диалектика, 1997.
- 30.Буч Г. Объектно—ориентированный анализ и проектирование. 2—е изд./ Пер. с англ. — М.: Издательство Бином, СПб: "Невский диалект", 1998 г. —560 с.
- 31.Васкевич Д. Стратегии клиент/сервер. — Киев: Диалектика, 1997.
- 32.Гаврилова Г.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. — СПб., «Питер», 2000. — 384 с.
- 33.Гилуа М.М. Множественная модель данных в информационных

- системах. — М.: Наука, 1992.
- 34.Голосов А.О. Аномалии в реляционных базах данных //СУБД. — 1986. — №3. — С.23—28.
- 35.Грабер М. Введение в SQL. — М.: Лори, 1996. — 379 с.
- 36.Грабер М. Справочное руководство по SQL. — М.: Лори, 1997. — 291 с.
- 37.Дейт К. Руководство по реляционной СУБД DB2. — М.: Финансы и статистика, 1988. — 320 с.
- 38.Дейт К.Дж. Введение в системы баз данных, 6—е издание. — К.; М.; СПб.: ИД «Вильямс», 2000. — 848 с.
- 39.Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. — М.: Мир, 1991. — 252 с.
- 40.Диго С.М. Проектирование и использование баз данных. — М.: Финансы и статистика, 1995. — 208 с.
- 41.Злуф М.М. Query—by—Example: язык баз данных //СУБД. — 1996. — №3. — С.149—160.
- 42.Змитрович А.И. Базы данных. — Мн., Из—во «Университетское», 1991.
- 43.Иванов А. Кремер Ю. Язык Smalltalk: концепция объектно-ориентированного программирования. "Компьютер пресс", 4, 1992 г. с. 21—31.
- 44.Информатика: Энциклопедический словарь для начинающих/ под ред. Поспелова Д.А., Борисов А.Н. и др. Обработка нечеткой информации в системах принятия решений,— М.: Радио и связь, 1989.—303 с.
- 45.Канолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. 2—е издание. — К.; М.; СПб.: ИД «Вильямс», 2000. — 1120 с.
- 46.Карпова Т.С. Базы данных: модели, разработка, реализация. — СПб., «Питер», 2001. — 304 с.
- 47.Кириллов В.В. Структуризованный язык запросов (SQL). — СПб.: ИТМО, 1994. — 80 с.
- 48.Кондрашина Е.Ю., Литвинцева Л.В., Поспелов Д.А. Представление

- знаний о времени и пространстве в интеллектуальных системах / Под ред. Д.А.Поспелова. — М.: Наука. Гл.ред.физ.—мат. лит. 1989 г. — 328 с.
- 49.Корнеев В.В., Гарев А.Ф. Базы данных. Интеллектуальная обработка информации. — СПб., «Корона», 2000.
- 50.Кузин Л.Т. Основы кибернетики, т. 2, — М.: Энергоатомиздат, 1994. — 350 с.
- 51.Кузнецов С.Д. Введение в системы управления базами данных //СУБД. — 1995. — №1,2,3,4, 1996. — №1,2,3,4,5.
- 52.Кузнецов С.Д. Дубликаты, неопределенные значения, первичные и возможные ключи и другие экзотические прелести языка SQL //СУБД. — 1997. — №3. — С.77—80.
- 53.Кузнецов С.Д. Неопределенная информация и трехзначная логика // СУБД. — 1997. — №5. — С.65—67.
- 54.Кузнецов С.Д. Операционные системы для управления базами данных // СУБД. — 1996. — №3. — С.95—102.
- 55.Кузнецов С.Д. Стандарты языка реляционных баз данных SQL: краткий обзор // СУБД. — 1996. — №2. — С.6—36.
- 56.Ладыженский Г.М. Системы управления базами данных — кратко о главном // СУБД. — 1995. — №1,2,3,4.
- 57.Левин Р., Дранг Д., Эделсон И./ Перевод с англ. — Практическое введение в технологию искусственного интеллекта и экспертных систем с иллюстрацией на Бейсике, — М.: Финансы и статистика, 1990.
- 58.Логический подход к искусственному интеллекту. / Тейз А., Грибомон П. и др. — М.: Мир. - 1990.
- 59.Мартин Д. Планирование развития автоматизированных систем. — М.: Финансы и статистика, 1984. — 196 с.
- 60.Мейер М. Теория реляционных баз данных. — М.: Мир, 1987. — 608 с.
- 61.Минский М. Фреймы для представления знаний. — М.:Мир, 1979.
- 62.Нагао М., Катаяма Т., Уэмура С. Структуры и базы данных. — М.: Мир, 1986. — 197 с.

63. Нечеткие множества и теория возможностей. Последние достижения/ под ред. Ягера Р.Р., — М.: Радио и связь, 1986. — 408 с.
64. Нильсон Н. Искусственный интеллект. — М.: Мир, 1973.
65. Нильсон Н. Принципы искусственного интеллекта: Пер. с англ. — М.: Радио и связь. 1985. — 376 с.
66. Оззу М.Т., Валдуриз П. Распределенные и параллельные системы баз данных // СУБД. — 1996. — №4. — С.4—26.
67. Озкарахан Э. Машины баз данных и управление базами данных. — М.: Мир, 1989.
68. Осуга С. Обработка знаний: Пер. с япон. 1989.— 293 с.
69. Понятие лингвистической переменной и его применение к принятию приближенного решения/ Пер. с англ.,— М.: Мир, 1976,—203 с.
70. Попов Э.В. Экспертные системы. — М.: Наука, 1987.
71. Поспелов Д.А. Логико-лингвистические модели в системах управления, — М.: Энергоиздат, 1981 г.— 230 с.
72. Поспелов Д.А. Моделирование рассуждений. Опыт анализа мыслительных актов. — М.: Радио и связь, 1989. — 1989. — 184 с.
73. Поспелов Д.А. Ситуационное управление. Теория и практика. — М.: Наука, 1986 г.— 284 с.
74. Построение экспертных систем / Под ред. Ф.Хейес—Рот, Д.Уотерман, Д.Ленат — М.: Мир, 1987.
75. Практическое введение в технологию искусственного интеллекта и экспертных систем / Р.Левин, Д.Дранг, Б.Эделсон: Пер. с англ. — М.: Финансы и статистика. 1991.— 239 с.
76. Представление и использование знаний: Пер. с япон./ Под ред. Х. Уэно, М. Исудзука. — М.: Мир, 1989. — 220 с.
77. Пржиялковский В. В. Абстракции в проектировании БД // СУБД. — 1998. — №1. — С.90—97.
78. Приобретение знаний: Пер. с япон. / Под ред. С.Осуги, Ю.Саэки. — М.: Мир, 1990. — 304 с.



79. Прохоров А, Определение оптимальной структуры базы данных // Informix magazine. Русское издание. — 1998. — Апрель.
80. Тиори Т., Фрай Д. Проектирование структур баз данных. В 2 кн., — М.: Мир, 1985. Кн. 1. — 287 с.: Кн. 2. — 320 с.
81. Ульман Д. Базы данных на Паскале. — М.: Машиностроение, 1990. — 386 с.
82. Ульман Д. Основы систем баз данных. — М.: Финансы и статистика, 1983. — 334 с.
83. Ульман Дж.Д., Уидом Дж. Введение в системы баз данных. — М., «Лори», 2000.
84. Форсайт Р. Экспертные системы. — М., ФиС, 1987.
85. Хаббард Д. Автоматизированное проектирование баз данных. — М.: Мир, 1984. — 294 с.
86. Хансен Г., Хансен Дж. Базы данных. Разработка и управление. — М., Издательство «Бином», 1999. — 704 с.
87. Харрингтон Дж. Л. Проектирование реляционных баз данных. Просто и доступно. — М., «Лори», 2000. — 231 с.
88. Хомоненко А.Д., Цыганков В.М. Базы данных. Учебник для высших учебных заведений. — СПб., «Корона», 2000.
89. Цаленко М.Ш. Моделирование семантики в базах данных. — М.: Наука, 1988.
90. Цикритизис Д., Лоховски Ф. Модели данных. — М.: Финансы и статистика, 1985. — 344 с.
91. Чамберлин Д.Д., Астрахан М.М., Эсваран К.П., Грифитс П.П., Лори Р.А., Мел Д.В., Райшер П., Вейд Б.В. SEQUEL 2: унифицированный подход к определению, манипулированию и контролю данных //СУБД. — 1996. — №1. — С.144—159.
92. Чаудхари С. Методы оптимизации запросов в реляционных системах //СУБД. — 1998. — №3. — С.22—36.
93. Чен П. Модель "сущность-связь" — шаг к единому представлению о

данных //СУБД. — 1995. — №3. — С.137—158.