

Оглавление

1.	Общие положения и понятия теории баз данных (база данных, СУБД, банк данных, трехуровневая архитектура, внешняя, физическая и концептуальная схемы данных, физическая и логическая независимость данных, модель данных, развитие языков баз данных, логическая целостность данных, процедурно-ориентированные и проблемно-ориентированные СУБД). Эволюция методов хранения данных.....	6
2.	Система управления базами данных. Классификация СУБД по различным критериям. Достоинства и недостатки систем с многотерминальной архитектурой, систем с архитектурой файлового сервера, с архитектурой "клиент-сервер".	7
3.	OLTP-, DSS (OLAP)-системы. Их характеристики.....	8
4.	Типовая организация СУБД.....	10
5.	Функции СУБД. Понятие мягкого и жесткого сбоя. Журнализация данных. Содержимое журнала транзакций. Процедура восстановления данных. Протокол WAL. Понятие конвейерного и разнесенного параллелизма.	10
6.	Этапы обработки запроса в серверах баз данных.	12
7.	Уровни доступа к базам данных.....	12
8.	Встроенный SQL (правила описания и использования главных переменных в ESQL/C, средства обработки ошибок в ESQL/C, средства обработки NULL-значений в ESQL/C, курсоры в ESQL/C, средства динамического SQL в ESQL/C).....	12
9.	Доступ к базам данных на основе стандарта ODBC (архитектура ODBC, последовательность действий при разработке ODBC-программы).	18
10.	Доступ к базам данных посредством CGI-скриптов (методы передачи данных, спецификация CGI-скрипта). Доступ к базам данных с помощью языка PHP.	19
11.	Доступ к базам данных с использованием технологии ActiveXDataObjects (ADO) .21	21
12.	Общие понятия теории отношений. Понятие отношения. Свойства бинарных отношений (рефлексивность, иррефлексивность, симметричность, ассиметричность, антисимметричность, транзитивность). Классы отношений (эквивалентность, толерантность, квазипорядок, строгий порядок). Модель данных. Классические модели данных.	23
13.	Иерархическая модель базы данных и ее свойства. Сущность. Связь. Связи 1:1, 1:M, M:N. Основные понятия иерархической модели данных. Характеристика операторов манипулирования данными иерархической модели. Недостатки иерархической модели. .24	24
14.	Сетевая модель базы данных и ее свойства. Основные понятия сетевой модели данных: элемент данных, агрегат данных, запись, набор. Свойства типов записей и связей сетевой модели. Характеристика операторов манипулирования данными сетевой модели. Недостатки сетевой модели.....	27
15.	Системы, основанные на инвертированных списках.	28
16.	Реляционная модель данных. Отношение реляционной модели данных. Основные понятия реляционной модели. Определение домена, семантическая нагрузка понятия домена. Фундаментальные свойства отношений. Базовые свойства реляционной модели данных. Обоснование требования отсутствия кортежей-дубликатов, отсутствия упорядоченности кортежей, упорядоченности атрибутов. Требования структурной, манипуляционной, целостной частей реляционной модели данных. Возможные подходы для выполнения	28

требования целостности по ссылкам. Процедура каскадного удаления. Null-значения.	
Правила Кодда	30
17. Язык реляционной алгебры и его свойства. Операции реляционной алгебры. Запись операций реляционной алгебры на языке SQL	34
18. Реляционное исчисление на кортежах. Основные понятия. Запись операций реляционного исчисления средствами языка реляционной алгебры. Реляционное исчисление на доменах.....	35
19. Язык SQL. Стандартизация языка SQL. Структура операторов SQL: Select, Insert, Update, Delete. Подзапросы. Соединения. Виды соединений. Основные правила использования конструкций языка SQL. Представления. Оптимизация запросов SQL. Достоинства и недостатки языка SQL.....	38
20. Жизненный цикл базы данных. Основные этапы жизненного цикла. Понятие бизнес-процесса.....	42
21. Назначение методологии IDEF0. Виды связей в IDEF0. Основные понятия и конструкции диаграмм работ. Назначение методологии диаграмм потоков данных. Основные понятия и конструкции диаграмм потоков данных. Методология объектного проектирования на языке UML. Диаграммы прецедентов. Основные понятия и конструкции. Диаграммы деятельности. Основные понятия и конструкции. CASE-средства. Классификация CASE-средств.	46
22. Этапы проектирования баз данных. Задачи инфологического, логического, физического проектирования	54
23. Инфологическое проектирование. Сущности, атрибуты, связи. Способы представления сущности. Классификация атрибутов. Правила атрибутов. Понятие безусловной, условной, биусловной, рекурсивной связи. Фундаментальные виды связей. Формализация связи. Формализация связей 1:1, 1:M, M:N. Стержневая, ассоциативная, характеристическая, обозначающая сущности. Композиция связей. Понятие подтипа и супертипа. Взаимноисключающие связи. Получение реляционной схемы из ER-диаграммы.	56
24. Логическое проектирование реляционных баз данных. Аномалии операций с базой данных. Общие свойства нормальных форм. Виды нормальных форм. Условия нахождения отношений в первой нормальной форме. Негативные последствия нахождение отношения лишь в первой нормальной форме. Зависимость адекватности базы данных предметной области, легкости разработки и сопровождения базы данных, скорости выполнения основных операций от степени нормализации отношений базы данных.....	59
25. Функциональные зависимости и их свойства. Замыкание множества FD. Аксиомы Амстронга. Функционально полная и частичная зависимости неключевого атрибута от составного ключа. Минимальное множество функциональных зависимостей. Декомпозиция без потерь и функциональные зависимости. Теорема Хеза. Условия нахождения отношений во второй нормальной форме.....	60
26. Транзитивная зависимость. Условия нахождения отношений в третьей нормальной форме. Теорема Риссонена. Перекрывающиеся возможные ключи и нормальная форма Бойса-Кодда. Условия нахождения отношений в усиленной третьей нормальной форме. 61	
27. Многозначные зависимости. Лемма и теорема Фейджина. Тривиальные многозначные зависимости. Условия нахождения отношений в четвертой нормальной форме.....	62

28.	Зависимости проекции-соединения. Циклические ограничения (3D-ограничения). Условия нахождения отношений в пятой нормальной форме проекции-соединения.	63
29.	Ограничения целостности. Общие требования относительно ограничений целостности. Классификация ограничений целостности. Возможные подходы относительно удаления целевой сущности, на которую ссылается внешний ключ. Возможные подходы относительно обновления первичного ключа целевой сущности, на которую ссылается внешний ключ. Средства поддержания целостности информации в базе данных. Средства и способы задания ограничений целостности в языке SQL.	64
30.	Физическое проектирование. Задачи физического проектирования. Способы управления данными.	66
31.	Индексы. Создание индекса. Правила определение набора требуемых индексов. Задача рефакторинга.	66
32.	Организация индексов. Классификация индексов. Доступ с плотным индексом. Доступ с неплотным индексом. Решение проблемы переполнения при использовании разряженного индексного файла. Доступ посредством инвертированных файлов. Доступ посредством В-деревьев. В- и В+-деревья. Битовые индексы. Хеш-индексирование. Метод открытой адресации. Метод области переполнения. Метод многократного хеширования. Достоинства и недостатки различных методов индексирования.	67
33.	Методы организации хранения данных. Способы хранения отношений. Табличные пространства. Фрагментация. Кластеризация таблиц. Параметры проектирования физического уровня.	71
34.	Понятие транзакции. Свойства классических транзакций. Модели транзакций. Транзакции с контрольными точками. Многозвенные транзакции. Вложенные транзакции. Многоуровневые транзакции.	72
35.	Транзакции и параллелизм. Три проблемы, связанные с параллелизмом. Эффекты параллелизма. Управление транзакциями. Сериализация транзакций. Понятие смеси и графика транзакций. Виды конфликтов между транзакциями. Изолированность пользователей. Уровни изолированности.	74
36.	Методы управления транзакциями. Метод синхронизационных захватов. Метод гранулированных синхронизационных захватов. Решение на основе аппарата синхронизационных захватов проблем, связанных с параллелизмом. Распознавание тупиковых ситуаций. Разрушение тупиков. Метод временных меток. Метод выделения версий данных.	76
37.	Технология "клиент-сервер". Преимущества модели "клиент-сервер" в сравнении с традиционной моделью обработки данных. Логические компоненты приложений. Три модели архитектуры "клиент-сервер", их достоинства и недостатки. Трехзвенная архитектура модели "клиент-сервер". Сервер приложений. Традиционный подход к работе с сервером.	80
38.	Понятие активного сервера. Задачи активного сервера. Ограничения и утверждения. Процедуры и функции. Перегружаемые функции. Сигнатура. Хранимые процедуры. Средства организации хранимых процедур (язык SPL). Особенности написания хранимых процедур и функций в различных СУБД (Informix, Oracle, PostgreSQL). Триггеры. DDL- и DML-триггеры. Средства написания триггеров. Особенности написания триггеров в различных СУБД (Informix, Oracle, PostgreSQL). Механизм событий. Средства механизма событий. Активные базы данных и модели транзакций.	82

39. Классификация информационных систем. Характеристики OLTP-систем, характеристики DSS-систем. Концепции хранилища данных, OLAP-анализа, DataMining. Классификация аналитических систем (DSS-систем) по функциональности, по архитектуре.	
87	
40. Принципы построения хранилищ данных. Требования к хранилищу данных. Архитектура хранилища данных. Основные его компоненты. Категории данных хранилища данных. Метаданные. ETL-процесс. Процедуры этапа преобразования данных в хранилище данных. Проблемы очистки данных. Подходы и методы решения проблем очистки данных. Процедуры ITL. Подходы к построению хранилищ данных.....	90
41. OLAP-технология. Многомерный анализ данных. Основные понятия OLAP-технологии. Тест FASMI. Операции OLAP-технологии. Классификация OLAP-средств по архитектуре. Модели хранилища данных. Многомерная модель хранилищ данных (MOLAP). Реляционная модель хранилищ данных (ROLAP). Схема "звезда". Схема "снежинка". Расширения языка SQL для OLAP-анализа данных. Классификация OLAP-средств и их примеры. Продукты класса Desktop OLAP.....	92
42. Интеллектуальный анализ данных (Data Mining). Требования к обнаруживаемым знаниям. Технологии Data Mining. Машинное обучение. Модели Data Mining. Задача классификации и регрессии. Задача поиска ассоциативных правил. Задача кластеризации.	94
43. Распределенные базы данных. Свойства распределенных баз данных. Технологии распределенных баз данных (особенности обработки и оптимизации запросов, управление одновременным доступом, протоколы обеспечения надежности, технологии тиражирования данных). Протокол двухфазной фиксации данных. Схемы репликации данных.	95
44. Объектно-реляционные свойства СУБД. Сложные типы данных. Коллекции. Наследование при работе с базами данных. Определенные пользователем типы данных. Функции приведения.	96
45. NoSQL базы данных. Обеспечение согласованности данных. Теорема CAP. Модель ключ-значение. Основные операции в базах данных ключ-значение. Документная модель данных. Выбор уровня денормализации при использовании документных баз данных. Столбцовая модель данных. Основное назначение использования столбцовых баз данных. Графовая модель данных. Типы графовых моделей данных. Достоинства и недостатки NoSQL баз данных.	97
46. Основные принципы, лежащие в основе темпоральных баз данных. Понятие времени в темпоральных базах данных. Модели, используемые в темпоральных базах данных (TRM, HDM).	100
47. Безопасность баз данных. Проблема безопасности. Построения модели угроз. Основные термины и понятия, связанные с информационной безопасностью. Защита баз данных. Внутренние и внешние источники угроз. Принципы построения защищенных систем баз данных. Угрозы безопасности баз данных. Организационно-административные и компьютерные средства защиты базы данных.....	100
48. Идентификация и аутентификация. Атаки, специфичные для баз данных. Авторизация. Модель политики безопасности. Дискреционная модель безопасности. Ролевая модель безопасности.	104
49. Мандатная модель безопасности. Модель многоуровневой безопасности Белла Лападула и ее основные свойства. Структура метки доступа. Последовательность шагов управления мандатным доступом в Oracle. Принцип многоэкземплярности.	106

50. Дополнительные инструменты защиты базы данных. Представления. Аудит.
Резервное копирование и восстановление. Виды резервного копирования. Методы
восстановления. Метод наката. Метод отката. Контрольные точки.....107

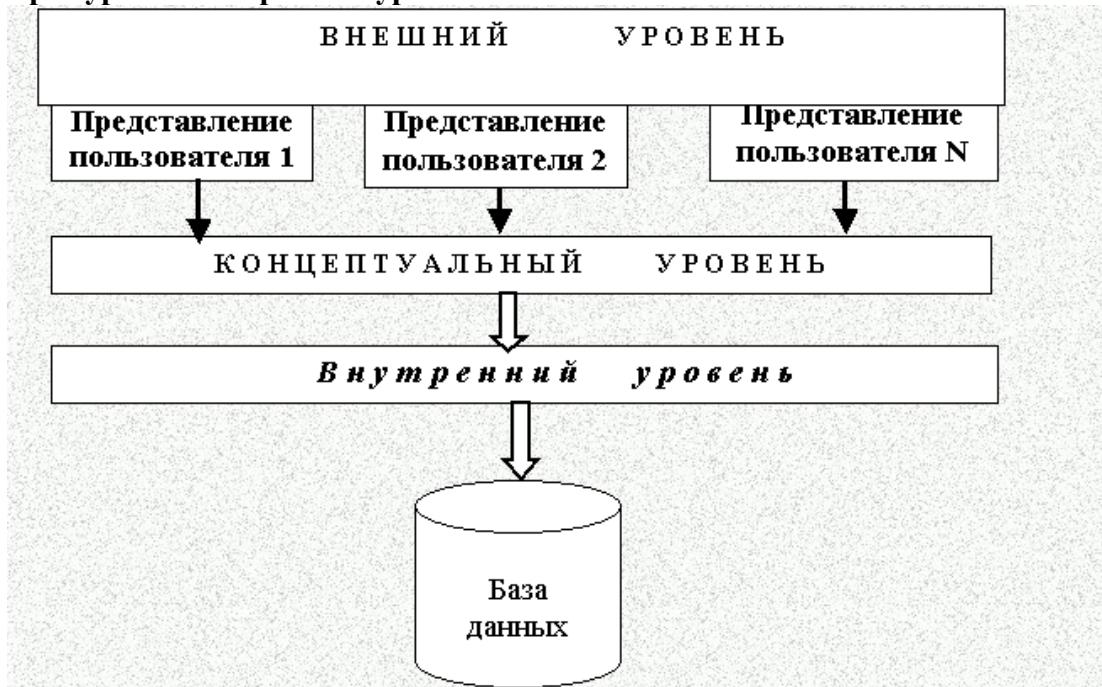
1. Общие положения и понятия теории баз данных (база данных, СУБД, банк данных, трехуровневая архитектура, внешняя, физическая и концептуальная схемы данных, физическая и логическая независимость данных, модель данных, развитие языков баз данных, логическая целостность данных, процедурно-ориентированные и проблемно-ориентированные СУБД). Эволюция методов хранения данных.

База данных – совокупность логически связанных данных и описаний этих данных, доступ к которым осуществляется с помощью системы управления базами данных (СУБД)

СУБД (система управления базами данных) – средство централизованного управления базой данных как социальным ресурсом в интересах всей совокупности пользователей и представляющая собой набор программных средств, предназначенных для организации, контроля и администрирования содержащейся информации.

Банк данных – разновидность информационной системы, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации на основе концепции БД и СУБД.

Трехуровневая архитектура



Верхний уровень – отражает представление конечного пользователя о конфигурации данных.

Внешняя схема данных описывает то, как видят различные группы пользователей архитектуру базы данных.

Концептуальный уровень – это объединяющее представление данных, используемых всеми пользовательскими приложениями, работающими с данной базой.

Концептуальная схема предназначена для абстрактного представления всей базы данных. Разработка концептуальной схемы данных называется концептуальным проектированием базы данных, которое включает в себя: оценку и учет потребностей пользователей, и выделение для них соответствующих прав и определение для них нужных элементов управления и взаимодействия с БД.

Внутренний уровень – служит для адаптации концептуальной модели к конкретной СУБД.

Внутренняя схема данных – это полное описание физической реализации базы данных. При помощи внутренней схемы данных осуществляется настройка. С ее помощью можно достичь

оптимальной производительности СУБД и обеспечить экономное использование места на носители информации.

Физическая независимость данных: возможность изменения способа хранения, расположения или переноса данных без влияния на логическую структуру БД и работу приложений, работающих с БД.

Логическая независимость данных: общая структура данных может быть изменена без изменения прикладных программ.

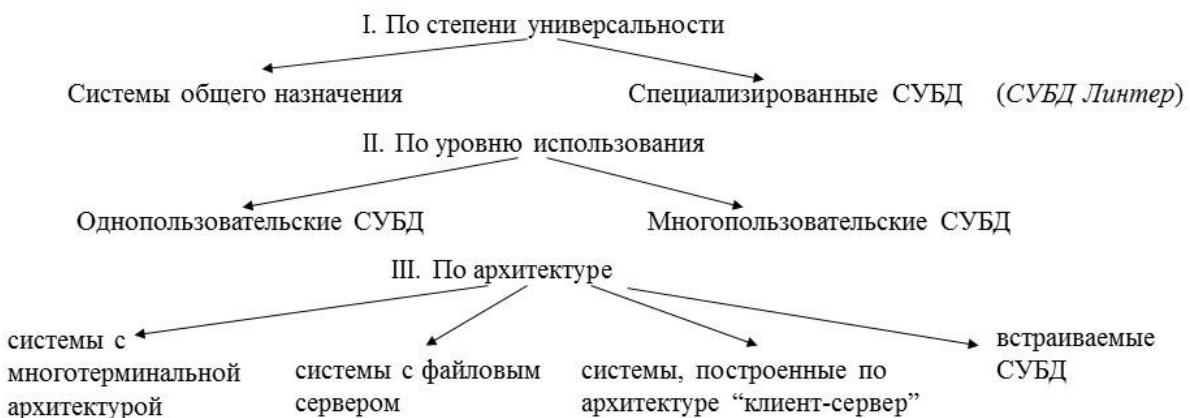
Модель данных – формализованное описание структур единиц информации и операций над ними в информационной системе. Модель данных делится на реляционную, иерархическую и сетевую.

Целостность базы данных - соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.
Логическая целостность – непротиворечивость данных в базе.

2. Система управления базами данных. Классификация СУБД по различным критериям. Достоинства и недостатки систем с многотерминальной архитектурой, систем с архитектурой файлового сервера, с архитектурой "клиент-сервер".

Система управления БД – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Классификация СУБД



Многотерминальная архитектура

Плюсы:

1. Ориентация на системы многопользовательского доступа
2. Не критичность к удаленности рабочих мест от сервера

Минусы:

1. Высокая стоимость администрирования и сопровождения
2. Проблема масштабируемости

Архитектура файлового сервера

Плюсы:

1. низкая стоимость разработки;

2. высокая скорость разработки;
3. невысокая стоимость обновления и изменения ПО.

Минусы:

1. Потенциально более высокая загрузка сети
2. Затрудненность или невозможность централизованного управления, обеспечения высокой надежности, высокой доступности и высокой безопасности.

Архитектура клиент-сервер

Плюсы:

1. Сохранность информации.
2. Устойчивость к сбоям.
3. Масштабируемость (способность к расширению).
4. Большая защищенность информации от несанкционированного доступа.
5. Меньшая нагрузка сети одним пользователем
6. Большая гибкость системы.

Минусы:

1. Дорогое техническое обеспечение;
2. Дорогие серверные операционные системы и клиентские лицензии;
3. Требуется администратор сети.

3. OLTP-, DSS (OLAP)-системы. Их характеристики.

В области информационных технологий выделяют два класса информационными систем (и соответственно два класса задач):

- OLTP-системы;
- DSS-системы.

OLTP-системы (OnLine Transaction Processing) – системы оперативной обработки транзакций. Основная функция подобных систем заключается в одновременном выполнении большого числа коротких транзакций от большого числа пользователей.

Системы OLTP характеризуются:

- поддержкой большого числа пользователей;
- малым временем отклика на запрос;
- относительно короткими запросами;
- участием в запросах небольшого числа таблиц.

Большая часть запросов OLTP-систем известна заранее еще на этапе проектирования системы. Критическими для OLTP-приложений являются скорость и надежность выполнения коротких запросов. Типичные примеры OLTP-систем: работа оператора в банке, ваши действия у банкомата.

Исторически такие системы возникли в первую очередь, поскольку реализовывали потребности в учете, скорости обслуживания, сборе данных и пр. Однако вскоре пришло понимание, что сбор данных – не самоцель, и накопленные данные могут быть полезны для извлечения информации.

Это привело к появлению другого типа систем (и соответственно класса задач) – **систем поддержки принятия решений DSS (Decision**

Support System), ориентированных на анализ данных, на выполнение более сложных запросов, моделирование процессов предметной области, прогнозирование, нахождение зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками покупателей), для проведения анализа «что, если...».

DSS-системы оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми из электронных таблиц или из других источников данных, и характеризуются следующими признаками:

- использование больших объемов данных;
- добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложений);
- данные, добавленные в систему, обычно никогда не удаляются;
- перед загрузкой данные проходят различные процедуры «очистки», связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления, данные могут быть некорректны, ошибочны;
- небольшое число пользователей;
- очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса (интерактивность);
- скорость выполнения запросов важна, но не критична.

Перечисленные характеристики требуют особой организации данных, отличных от тех, что используются в OLTP-системах (нормализованные реляционные таблицы).

Аналитические системы, ориентированные на аналитика, можно разделить:

- на статические DSS, известные в литературе как информационные системы руководителя (Executive Information Systems – EIS);
- динамические DSS.

EIS-системы содержат в себе предопределенные множества запросов (примером могут служить программные продукты финансового характера компании 1С).

Вторая группа (динамические DSS), напротив, ориентированы на обработку нерегламентированных запросов аналитиков к данным. Постепенно в этом направлении оформился ряд концепций хранения и анализа корпоративных данных:

- концепция хранилища данных (Data Warehouse);
- оперативная аналитическая обработка OLAP (OnLine Analytical Processing);
- интеллектуальный анализ данных (Data Mining).

Концепция хранилища данных определяет процесс сбора, отсеивания, предварительной обработки и накопления данных с целью:

- долговременного хранения данных;
- представления данных в форме, удобной для проведения анализа и создания аналитических отчетов.

Концепция OLAP – концепция комплексной оперативной обработки данных, использующая методы многомерного анализа данных в целях поддержки процессов принятия решений.

Концепция интеллектуального анализа данных определяет задачи поиска функциональных и логических закономерностей в накопленной информации, построение моделей и правил, которые объясняют найденные аномалии и/или прогнозируют развитие некоторых процессов.

4. Типовая организация СУБД.

Типовая организация современной СУБД



5. Функции СУБД. Понятие мягкого и жесткого сбоя. Журнализация данных. Содержимое журнала транзакций. Процедура восстановления данных. Протокол WAL. Понятие конвейерного и разнесенного параллелизма.

Основные функции СУБД

1. Управление данными во внешней памяти
2. Управление буферами ОП
3. Поддержка языков баз данных
4. Поддержка логической целостности БД

- **Мягкий сбой системы (аварийный отказ программного обеспечения).** Мягкий сбой характеризуется утратой оперативной памяти системы. При этом поражаются все выполняющиеся в момент сбоя транзакции, теряется содержимое всех буферов базы данных. Данные, хранящиеся на диске, остаются неповрежденными. Мягкий сбой может

произойти, например, в результате аварийного отключения электрического питания или в результате неустранимого сбоя процессора.

- **Жесткий сбой системы (аварийный отказ аппаратуры).** Жесткий сбой характеризуется повреждением внешних носителей памяти. Жесткий сбой может произойти, например, в результате поломки головок дисковых накопителей.

Журнализация изменений – функция СУБД, обеспечивающая восстановление БД в предыдущее согласованное состояние в случае логических или физических отказов

Журнал (log) – особая часть БД, недоступная пользователям и поддерживаемая с особой тщательностью, в которую поступают записи обо всех изменениях основной части БД

Содержимое журнала транзакций:

- порядковый номер, тип и время изменения;
- идентификатор транзакции;
- объект, подвергшийся изменению;
- предыдущее состояние объекта и новое состояние объекта.

→ протокол Write Ahead Log – WAL)

Алгоритм восстановления после мягкого сбоя

Метод отката (rollback): 1. Откат незавершенных транзакций (undo).
2. повторно воспроизводят (redo) тех операций завершенных транзакций, результаты которых не отображены во внешней памяти.

Алгоритм восстановления после жесткого сбоя

Метод наката (rollforward): 1. Загрузка архивной копии с помощью утилиты восстановления.
2. Перезапуск всех транзакций, которые закончились к моменту сбоя.

Write-Ahead Logging - Упреждающее журналирование является ключевым методом обеспечения требований ACID. WAL позволяет обеспечитьброс на диск записей из журнала транзакций, относящихся к изменениям данных, раньше того, как будут сброшены на диск сами эти изменённые страницы данных.

Вертикальный (конвейерный) параллелизм предполагает организацию параллельного выполнения различных операций плана запроса на базе механизма конвейеризации. В соответствие с данным механизмом между смежными операциями в дереве запроса организуется поток данных в виде *конвейера*, по которому элементы данных (*гранулы*) передаются от *поставщика* к *потребителю*. Традиционный подход к организации конвейерного параллелизма заключается в использовании абстракции итератора для реализации операций в дереве запроса. Подобный подход получил название *синхронного конвейера*.

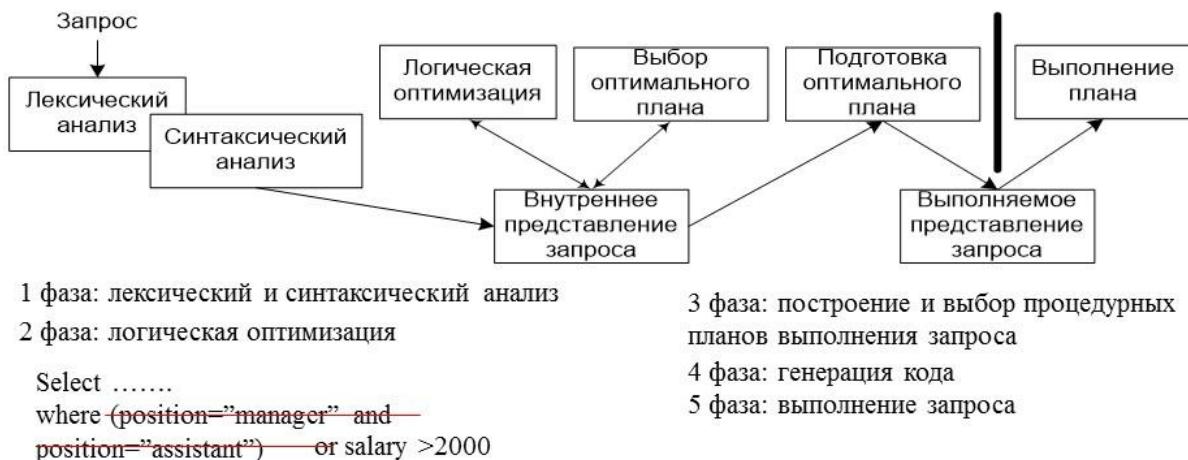
Следует отметить, что степень конвейерного параллелизма в любом случае ограничена количеством операций, вовлекаемых в конвейер. При этом для реляционных систем баз данных длина конвейера редко превышает 10 операций. Поэтому для достижения более высокой степени распараллеливания наряду с конвейерным параллелизмом необходимо использовать *внутриоперационный параллелизм*.

Горизонтальный (кустовой) параллелизм предполагает параллельное выполнение независимых поддеревьев дерева, представляющего план запроса. Основная проблема, связанная с кустовым параллелизмом, заключается в том, что очень трудно обеспечить, чтобы два подплана одного плана начали генерировать выходные данные в правильное время и в правильном темпе. При этом правильное время далеко не всегда означает одинаковое время, например для входных потоков операции хеш-соединения, а правильный темп далеко не всегда означает одинаковый темп, например для случая, когда входные потоки соединения слиянием имеют различные размеры. В силу указанных причин кустовой параллелизм редко используется

на практике. В научных публикациях кустовой параллелизм исследовался главным образом в контексте оптимизации запросов с мультисоединениями

6. Этапы обработки запроса в серверах баз данных.

Обработка SQL-запросов в серверах баз данных



7. Уровни доступа к базам данных.

Уровни доступа к базам данных

1. Без использования специальных средств
2. С использованием интерфейса CLI (Call-Level Interface) ----- ODBC (Open Database Connectivity)
Call_open_cursor_for_read (curs,,Ncol,buf1,buf2,pole3,pole4,,1);
3. С использованием встроенного SQL (Embedded SQL)
4. На базе специальных языков 4 поколения
5. Посредством хранимых процедур и триггеров
6. Доступ из среды Internet-Intranet
7. Доступ без кодирования в диалоговом режиме на основе средств разработки

8. Встроенный SQL (правила описания и использования главных переменных в ESQL/C, средства обработки ошибок в ESQL/C, средства обработки NULL-значений в ESQL/C, курсоры в ESQL/C, средства динамического SQL в ESQL/C).

главные переменные

(переменные, собственные переменные). В объявлении главных переменных в ESQL/C им предшествует знак \$ или описание производится внутри блока:

<Exec SQL begin declare section ... Exec SQL end declare section>

Тип и класс хранения главных переменных определяются аналогично C (automatic, static, external). Как и в языке C, возможны массивы (одно-, двумерные), структуры и указатели главных переменных. Инициализация переменных также аналогична C.

обработка ошибок

Сервер базы данных всегда возвращает код ошибки и некоторую другую информацию о выполнении операции в структуре данных, называемой областью связи SQL (**SQL Communication Area – sqlca**) (приложение 1).

Поле **sqlerrm** структуры **sqlca** содержит строку **sqlerrmc** с текстовым описанием ошибки и длину строки **sqlerrml** в символах (но не более чем **SQLERRMC_LEN**).

Поле **sqlstate** содержит символьный код возврата **SQLSTATE** (современная схема кодирования). Пять символов поля **sqlstate** могут содержать цифры и прописные буквы, обозначающие коды ошибок или предупреждений. Первые два символа обозначают класс состояния, последующие три символа уточняют само состояние (приложение 2).

Поле **sqlcode** содержит числовой код возврата **SQLCODE** (предыдущая схема кодирования):

- | | |
|--|---|
| <code>sqlca.sqlcode = 0</code> | – успешное выполнение (константа ECPG_NO_ERROR); |
| <code>0 < sqlca.sqlcode < 100</code> | – успешное выполнение с дополнительной информацией (в зависимости от описания); |

11

- | | |
|-----------------------------------|--|
| <code>sqlca.sqlcode = 100</code> | – успешное выполнение и наступление события «больше не найдено строк» (константа ECPG_NOT_FOUND); |
| <code>sqlca.sqlcode < 0</code> | – выполнение с ошибкой (детализированное значение кода ошибки может содержаться в других полях структуры sqlca). <hr/> |

обработка NULL

Поскольку в языке С нет возможности убедиться, имеет ли элемент таблицы какое-либо значение, ESQL/C делает это для своих главных переменных, называемых переменными-индикаторами. Переменная-индикатор – это дополнительная переменная, ассоциированная с главной переменной, в которую могут поступать NULL-значения. Когда сервер БД помещает данные в главную переменную, он также устанавливает специальное значение в переменную-индикатор, которое показывает, не являются ли эти данные NULL-значением.

Переменная-индикатор описывается как обычная главная переменная целого типа, а при использовании отделяется от главной переменной, в которую передаются значения, знаками «:», «\$» или словом **indicator**.

Примеры равнозначных использований переменных-индикаторов:

```
$int var1:varind1;  
$int var2$varind2;  
$int var3 INDICATOR varind3;
```

При выборе оператором Select NULL-значения переменная-индикатор (если она используется) получает значение **-1**. В случае нормального возврата переменной индикатор равен **0**. Если индикатор не используется, то результат зависит от режима генерации программы:

- если программа компилировалась с флагом **-icheck**, ESQL/C генерирует ошибку и устанавливает **sqlca.sqlcode** в отрицательное значение при возврате NULL-значения;
- если программа компилировалась без указанного флага, при отсутствии индикатора ошибка не генерируется.

Курсы

Обработка многострочного запроса осуществляется в два этапа:

- программа запускает запрос, не возвращая никаких данных;
- программа запрашивает строки данных по одной на каждое требование.

Указанные операции выполняются с помощью специального объекта данных, называемого **курсором**, который является структурой данных, отслеживающей внутреннее состояние запроса. Рассмотрим последовательность программных операций при работе с курсором.

1. Программа объявляет курсор и ассоциированный с ним оператор Select. Операция приводит к выделению памяти для хранения курсора (оператор Declare).

2. Программа открывает курсор. Это приводит к началу выполнения ассоциированного оператора Select, а также распознаванию наличия ошибок (оператор Open).

3. Программа считывает строку данных в главные переменные и обрабатывает их (оператор Fetch).

4. Программа закрывает курсор после прочтения последней строки (оператор Close).

Множество строк, возвращаемое предложением Select, носит название **активного множества**.

Курсор может находиться в одном из двух состояний: открытом и закрытым. Когда курсор открыт, он связан с активным множеством и может либо указывать на текущую строку, либо находиться между строк, перед первой строкой, после последней строки. Если курсор находится в закрытом состоянии, то он не связан с активным множеством, хотя и остается связанным с предложением Select.

Пример использования последовательного курсора:

```
$int o_num, i_num, s_num;  
$Declare the_item cursor for select order_num, item_num, stock_num  
           into $o_num, $i_num, $s_num from items;  
$open the_item;  
while (sqlca.sqlcode == 0)  
{   $fetch the_item  
    if (sqlca.sqlcode == 0)  
        printf("%d %d %d", o_num, i_num, s_num); }
```

Пример использования скроллирующего курсора (выбор каждой второй строки из некоторого набора):

```
$struct customer  
{      char cust_name[20],  
      char cust_state[2],  
      ....} cust_list[100];  
  
$char wanted_state[2];  
int row_count;  
$declare cust scroll cursor for  
    select * from customer where state=$wanted_state;  
printf("Enter 2-letter state code:");  
scanf("%s", &wanted_state);  
$open cust;  row_count=0;  
while (sqlca.sqlcode == 0)  
{  row_count=row_count+2;  
  $fetch relative+2 cust into $cust_list[row_count]; }  
$close cust;
```

Объявление массива структур

динамический SQL

Последовательность выполнения приведенных выше операторов такова.

1. Символьная строка, содержащая оператор Select, помещается в программную переменную select_2. Она содержит два占олнителя, отмеченные знаком «?».

2. Оператор Prepare преобразует символьную строку в структуру данных, связанную с именем q_orders.

3. Объявляется курсор cu_orders и связывается с именем подготовленного оператора.

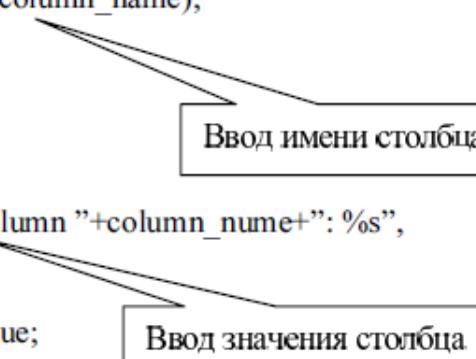
4. При открытии курсора начинается выполнение подготовленного оператора. Спецификация using в операторе Open представляет имена двух главных переменных, содержимое которых заменяет знаки вопроса в выполняемом операторе.

5. Спецификатор into оператора Fetch специфицирует главные переменные, которые должны принимать значения столбцов строки, выбранной по курсору.

Замечание. Указатели позиции «?» нельзя использовать вместо идентификаторов SQL, таких как имя БД, таблицы или столбца: эти идентификаторы должны указываться в тексте оператора при его подготовке. Если эти имена неизвестны при написании оператора, они могут быть получены через пользовательский ввод.

Последнее замечание поясняется следующим примером:

```
$char column_value[40], del_str[100]= “Delete from customer where ”;  
char column_name[30];  
scan (“Enter column name %s”, column_name);  
stcat ( column_name, del_str);  
stcat (“= ?”, del_str);  
$Prepare del from $del_str;  
scan( “Enter a search value in column ”+column_name+: %s”,  
      column_value);  
$Execute del using $column_value;
```



9. Доступ к базам данных на основе стандарта ODBC (архитектура ODBC, последовательность действий при разработке ODBC-программы).

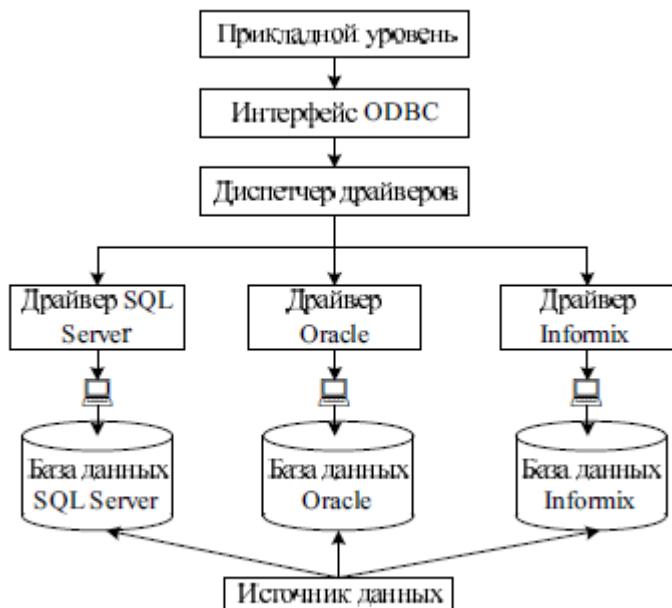


Рис. 2.2. Архитектура ODBC

Основной алгоритм создания ODBC-программ для наглядности можно представить в виде следующей блок-схемы (рис. 2.3).



Рис. 2.3. Последовательность действий при создании ODBC-программы

10. Доступ к базам данных посредством CGI-скриптов (методы передачи данных, спецификация CGI-скрипта). Доступ к базам данных с помощью языка PHP.

1. GET. Служит для извлечения информации по заданной в URL-ссылке. При использовании метода **GET** информация, введенная пользователем, посылается на сервер с помощью переменной окружения **QUERY_STRING**. Таким образом, длина сообщений при использовании этого метода ограничена. Метод **GET** используется по умолчанию при описании формы. Данные добавляются в конец URL-адреса и отделяются знаком «?».

`http://127.0.0.1/cgi-bin/varegi.cgi?RealName=Jack&age=25`

2. HEAD. Практически идентичен **GET**, но сервер не возвращает тело объекта. В ответе содержится только заголовок. Метод может использоваться для проверки гиперссылок.

3. POST. Передает данные для обработки CGI-программе, указанной в URL. С помощью метода **POST** информация, введенная пользователем, посылается непосредственно в сценарий с помощью выходного потока сервера **STDOUT** и входного потока **STDIN** вашего сценария. Преимущества использования этого метода – неограниченность длины передаваемого сообщения и безопасность передачи по сравнению с **GET**.

Существуют и другие реже используемые методы.

Собственно спецификация CGI описывает четыре набора механизмов обмена данными:

- переменные окружения
- формат командной строки
- формат стандартного ввода
- формат стандартного вывода

Переменные окружения. При запуске внешней программы сервер создает специфические переменные окружения, через которые передает приложению как служебную информацию, так и данные. Все переменные можно разделить на общие переменные окружения, которые генерируются при любой форме запроса, и запрос-ориентированные.

Опции командной строки. Командная строка используется только при запросах типа **ISINDEX**. При HTML FORMS или любых других запросах неопределенного типа командная строка не используется. Если сервер определил, что к скрипту обращаются через **ISINDEX** документ, то поисковый критерий выделяется из URL и преобразуется в параметры командной строки. При этом знаком разделения параметром является символ "+". Тип запроса определяется по наличию или отсутствию символа "=" в запросе. Если этот символ есть, то запрос не является запросом **ISINDEX**, если символа нет, то запрос принадлежит к типу **ISINDEX**. Параметры, выделенные из запроса, помещаются в `argv[1...]`. При этом после из выделения происходит преобразование всех шестнадцатеричных символов в их ASCII коды. Если число

параметров превышает ограничения, установленные в командном языке, например в shell, то формирования командной строки не происходит и данные передаются только через QUERY_STRING. Вообще говоря, следует заранее подумать об объеме данных, передаваемом скрипту, и выбрать соответствующий метод доступа. Размер переменных окружения тоже не ограничен, и если необходимо передавать много данных, то лучше сразу выбрать метод POST, т.е. передачу данных через стандартный ввод.

Формат стандартного ввода. Стандартный ввод используется при передачи данных в скрипт по методу POST. Объем передаваемых данных задается переменной окружения CONTENT_LENGTH, а тип данных в переменной CONTENT_TYPE. Если из HTML формы надо передать запрос типа: a=b&b=c, то CONTENT_LENGTH=7, CONTENT_TYPE=application/x-www-form-urlencoded, а первым символом в стандартном вводе будет символ "a". Следует всегда помнить, что конец файла сервером в скрипт не передается, а поэтому завершать чтение следует по числу прочитанных символов. Позже мы разберем примеры скриптов и обсудим особенности их реализации в разных операционных системах.

Формат стандартного вывода. Стандартный вывод используется скриптом для возврата данных серверу. При этом вывод состоит из заголовка и собственно данных. Результат работы скрипта может передаваться клиенту без каких-либо преобразований со стороны сервера, если скрипт обеспечивает построение полного HTTP заголовка, в противном случае сервер заголовок модифицирует в соответствии со спецификацией HTTP. Заголовок сообщения должен отделяться от тела сообщения пустой строкой. Обычно в скриптах указывают только три поля HTTP заголовка: Content-type, Location, Status.

Общая схема написания PHP-программы, выполняющей взаимодействие с базой данных, мало отличается от структуры CGI-скрипта, написанного любыми другими средствами, и включает:

- подключение к серверу баз данных и регистрацию;
- выбор базы данных, которая будет использоваться;
- посылку SQL-запроса на сервер и получение данных;
- отсоединение от сервера баз данных.

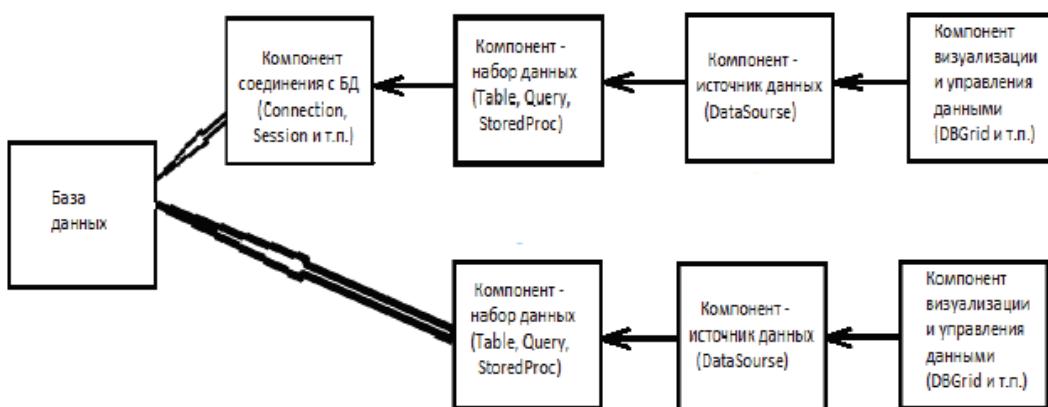
При этом остаются актуальными все замечания, сделанные в разделе 3 относительно установки переменных окружения и обес-

печения мер безопасности при работе с базой данных. Следующий пример демонстрирует использование средств PHP для работы с базой данных.

11. Доступ к базам данных с использованием технологии ActiveXDataObjects (ADO)

5.6. РАБОТА С БАЗАМИ ДАННЫХ В C++Builder

В общем случае схема работы с базой данных следующая:



Основной компонент при работе с базами данных – **TDataSet** – набор данных и его потомки – **TTable** – таблица БД, **TQuery** – запрос, **TStoredProc** – серверная процедура.

Назначение этих компонентов – получение наборов строк из базы данных, а для **TQuery** и **TStoredProc** – также выполнение модификации данных. Чтобы работать с базой данных, необходимо прежде всего с ней соединиться. Компонент набора данных может непосредственно соединиться с базой данных, но если в приложении много компонентов наборов данных, целесообразно делать это через специальный компонент соединения (обычно в их названии есть либо **Connection**, либо **Session**). Кроме того, компонент соединения позволяет управлять транзакциями на уровне приложения и получать информацию о произошедших ошибках через свойство **Errors**. К компоненту соединения остальные компоненты привязываются с помощью свойства **Connection**. После того как компонент **TDataSet** соединился с базой данных и получил набор строк, нужно сделать эти данные видимыми для пользователя. Этой цели служат компоненты визуализации и управления данными (**Data Controls**), такие как **TDBGrid** – табличное отображение курсора, **TDBLookupListBox** и **TDBLookupComboBox** – отображение списка значений определенного поля и т. п. Компонент **TDataSource** действует как посредник между компонентами **TDataSet** (**TTable**, **TQuery**, **TStoredProc**) и компонентами **Data Controls**.

12. Общие понятия теории отношений. Понятие отношения. Свойства бинарных отношений (рефлексивность, иррефлексивность, симметричность, ассиметричность, антисимметричность, транзитивность). Классы отношений (эквивалентность, толерантность, квазипорядок, строгий порядок). Модель данных. Классические модели данных.

1. Каждая система состоит из множества элементов

2. Элементы множеств находятся между собой в определенных отношениях

Бинарное отношение $R(A,B)$: (a,b) , где $a \in A$, $b \in B$

Если $(a,b) \in R \rightarrow aRb$

Пример 1. $A=\{1,2,3\}$, $B=\{2,3,4,5,6\}$ $R=\{(1,2), (1,3), (1,4), (1,5), (1,6), (2,2), (2,4), (2,6), (3,3), (3,6)\}$

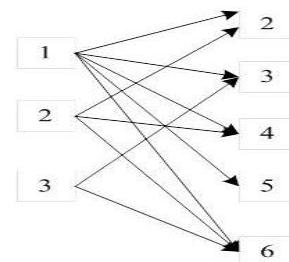
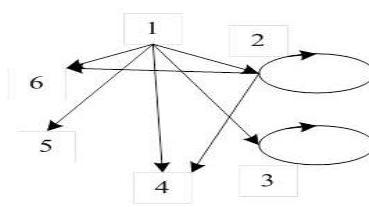
Пример 2. {Ольга, Павел, Иван} Отношение **а - брат б:** $R=\{(\Pi,O), (I,O), (\Pi,I), (I,\Pi)\}$

Бинарным отношением на множествах A и B называется любое подмножество R декартового произведения $A \times B$ ($A \times B$ – все пары вида (a, b)). Говорят, что элементы $a \in A$, $b \in B$ находятся в отношении R , если $(x, y) \in R$. Обозначают aRb или $R(a, b)$.

Дополнение к бинарному отношению R : $\underline{R}=(A^*B) \setminus R$. В примере 1 $\underline{R}=\{(2,3), (2,5), (3,2), (3,4), (3,5)\}$

Табличный и графические способы представления бинарных отношений:

R	2	3	4	5	6
1	1	1	1	1	1
2	1	0	1	0	1
3	0	1	0	0	1



Свойства однородных отношений, определенных на множестве A:

1. **Рефлексивность**, если $xRx \quad \forall x \in A$
2. **Иррефлексивность**, если xRx не имеет места ни для одного $x \in A$
3. **Симметричность**, если xRy влечет yRx , $\forall x, y \in A$
4. **Ассиметричность**, если $\forall x, y \in A$ имеет место xRy , либо yRx
5. **Антисимметричность**, если из xRy и yRx $\rightarrow x=y \quad \forall x, y \in A$
6. **Транзитивность**, если xRy и $yRz \rightarrow xRz, \forall x, y, z \in A$

Класс отношений	Рефлексивность	Антирефлексивность	Симметричность	Антисимметричность	Транзитивность
Эквивалентность	+		+		+
Толерантность	+		+		
Квазипорядок	+				+
Строгий порядок		+		+	+
Частичный порядок	+			+	+

Моделью данных называется формализованное описание структур единиц информации и операций над ними в информационной системе. Наиболее распространены реляционная, иерархическая и сетевая модели.

13. Иерархическая модель базы данных и ее свойства. Сущность. Связь. Связи 1:1, 1:M, M:N. Основные понятия иерархической модели данных. Характеристика операторов манипулирования данными иерархической модели. Недостатки иерархической модели.

Иерархическая модель данных (исторически первая структура баз данных) – представление БД в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

В этой модели данные представляются как дерево связанных записей. С ним в подчиненной связи типа 1 : N находятся дочерние записи. Связи между записями выражаются в виде отношений предок / потомок, а у каждой записи есть ровно одна родительская запись. Это помогает поддерживать ссылочную целостность: когда запись удаляется из дерева, все ее потомки должны быть также удалены.

В графической диаграмме схемы базы данных вершины используются для интерпретации сущностей, а дуги – для интерпретации связей. При этом возможна ситуация, когда сущность-предок не имеет потомков или имеет их несколько, тогда как у сущности-потомка обязательно должен быть только один предок. Объекты, имеющие общего предка, называются *близнецами*.

Для описания структуры (или, правильнее, схемы) иерархической базы данных на некотором языке программирования используется тип данных *дерево*. Тип данных *дерево* схож с типами *структура* языков программирования PL/I или С и *запись* языка Паскаль.

Тип *дерево* является составным. Он может включать в себя подтипы, каждый из которых, в свою очередь является типом *дерево*. Каждый из типов *дерево* состоит из одного *корневого* типа и упорядоченного (возможно, пустого) набора подчиненных типов. Каждый из элементарных типов, включенных в тип *дерево*, является простым или составным типом *запись*. Простая запись состоит из одного типа, например, числового, а составная запись объединяет некоторую совокупность типов (целое, строку символов и т.д.).

В иерархических СУБД может использоваться терминология, отличная от приведенной выше. Так в упомянутой системе IMS понятию *запись* соответствует термин *сегмент*, а под *записью базы данных* понимается вся совокупность записей, относящихся к одному экземпляру типа *дерево*.

Сущность – любой различимый объект, информацию о котором необходимо хранить в БД

Основные операции манипулирования иерархически организованными данными:

- найти указанное дерево базы данных (например, дерево со значением 40 в поле «№_больницы»);
- перейти от одного дерева к другому;
- перейти от одной записи к другой внутри дерева в порядке иерархии (например, от палаты к первому пациенту);
- перейти от одной записи к другой на одном уровне внутри дерева (например, к следующему пациенту);
- вставить новую запись в указанную позицию;
- удалить текущую запись.

Способы описания схем данных, базирующиеся на иерархической модели, и соответствующие языки манипулирования данными зависят от конкретной реализации.

Достоинства иерархической модели

1. *Простота модели.* Принцип построения баз данных в иерархической модели легок для понимания. Иерархия базы данных напоминает структуру компании или генеалогическое дерево.

2. *Использование отношений предок/потомок.* Иерархическая модель позволяет легко представлять отношения предок/потомок, например, «А является частью В» или «А принадлежит В».

3. *Быстродействие.* В иерархической модели отношения предок/потомок реализуются в виде физических указателей из одной записи на другую, поэтому перемещение по базе данных происходит достаточно быстро. Поскольку структура данных в иерархической модели отличается простотой, СУБД может размещать записи предков и потомков на диске рядом друг с другом, что позволяло свести к минимуму количество операций чтения-записи.

Недостатки иерархической модели

1. Операции манипулирования данными в иерархических системах ориентированы прежде всего на поиск информации сверху-вниз, т.е. по данному экземпляру сегмента-отца можно найти все экземпляры сегментов-сыновей. Обратный поиск затруднен, а часто и невозможен. Например, попытка реализовать запрос типа «В скольких сборниках статей опубликовал свои статьи господин Петров?» может оказаться весьма трудной задачей.

2. Дублирование данных на логическом уровне.
3. Для представления связи M:N необходимо дублирование деревьев (рис. 1.11).

4. В иерархической модели автоматически поддерживается целостность ссылок между предками и потомками по правилу: никакой потомок не может существовать без своего родителя. Целостность по ссылкам между записями, не входящими в одну иерархию, не поддерживается. Поэтому невозможно хранение в базе данных порожденного узла без соответствующего исходного. Аналогично, удаление исходного узла влечет удаление всех порожденных узлов (деревьев), связанных с ним.

Связь **1:1** существует, когда один экземпляр одной сущности связан с единственным экземпляром другой сущности.

Связь **1:M** возникает, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан только с одним экземпляром первой сущности.

Связь **M:N** существует, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан с одним или более экземпляром первой сущности.

14. Сетевая модель базы данных и ее свойства. Основные понятия сетевой модели данных: элемент данных, агрегат данных, запись, набор. Свойства типов записей и связей сетевой модели. Характеристика операторов манипулирования данными сетевой модели. Недостатки сетевой модели.

Сетевой подход к организации данных является расширением иерархического. Цель разработчиков сетевой модели – создание модели, позволяющей описывать связи М:N, чтобы одна запись могла участвовать в нескольких отношениях предок/потомок (рис. 1.14).

Элемент данных – наименьшая поименованная единица данных (аналог поля в файловых системах). Значение элемента данных может быть числовым (разных типов), символьным, логическим, может быть неопределенным, имя используется для идентификации (табельный номер, шифр детали, год рождения и т.д.).

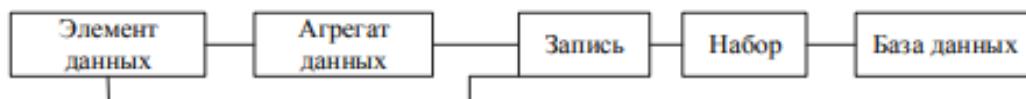


Рис. 1.15. Структурные элементы сетевой модели данных CODASYL

Агрегат данных – поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единую целую. Имя агрегата используется для его идентификации в схеме структуры более высокого уровня. Агрегат может быть простым, если состоит только из элементов данных, и составным, если включает в себя другие агрегаты (рис. 1.16).

Дата		
Число	Месяц	Год
Предприятие		
Название	Адрес	
	Почтовый индекс	Город
		Улица Номер дома

Рис. 1.16. Примеры агрегатов данных

Различают агрегаты типа *вектор* и типа *повторяющаяся группа*. Если в агрегате повторяющаяся компонента является простым элементом данных, его называют вектором. Пример вектора – агрегат «Зарплатная плата», в которой экземпляр элемента данных может повторяться до 12 раз (по числу месяцев в году). Агрегат, повторяющаяся компонента которого представлена совокупностью данных, называется повторяющейся группой. В повторяющуюся группу могут входить отдельные элементы данных, векторы, агрегаты или повторяющиеся группы. На рис. 1.17 представлен агрегат «Заказ на покупку», имеющий в своем составе повторяющуюся группу «Партия товара».

Запись – поименованная совокупность элементов данных и/или агрегатов, которая не входит в состав никакого другого агрегата (рис. 1.18). Запись может иметь сложную иерархическую структуру, допускает многократное применение агрегации. Имя записи использу-

Связи между записями выполняют так называемые наборы. **Набор** – поименованная двухуровневая иерархическая структура, связывающая запись-владельца и записи-членов. Каждый набор должен содержать только один экземпляр записи-владельца и любое количество экземпляров записей-членов (рис. 1.19).

Операции манипулирования данными:

- найти запись в наборе однотипных записей (палату с указанным номером);
- перейти от предка к первому потомку по некоторой связи (к первому врачу некоторой больницы);
- перейти к следующему потомку в некоторой связи (от врача Сидорова к Иванову);
- перейти от потомка к предку по некоторой связи (найти больницу врача Сидорова);
- создать новую запись, уничтожить запись, модифицировать запись;
- включить в связь, исключить из связи;
- переставить в другую связь и т.д.

15. Системы, основанные на инвертированных списках.

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в реляционных СУБД пользователи не имеют непосредственного доступа к инвертированным спискам (индексам).

База данных на инвертированных списках похожа на реляционную базу данных, т. е. также состоит из таблиц отношений, однако ей присущи важные отличия:

- допускается сложная структура атрибутов (атрибуты не обязательно атомарны);
- строки таблиц (записи) упорядочены в некоторой последовательности, каждой строке присваивается уникальный номер; физическая упорядоченность строк всех таблиц может определяться и для всей базы данных (так делается, например, в СУБД Datacom/DB);
- пользователям видны и хранимые таблицы, и пути доступа к ним;
- пользователь может управлять логическим порядком строк в каждой таблице с помощью специального инструмента – индексов; эти индексы автоматически поддерживаются системой и явно видны пользователям.

Некоторые атрибуты могут быть объявлены поисковыми (ключевыми), для каждого из них создается индекс, который содержит упорядоченные значения ключей и указатели на соответствующие записи основной таблицы (инвертированный список). Если таблицу требуется упорядочить по нескольким ключам, то создается несколько индексов. Если, например, в основной таблице с данными, представленной на рис. 1.27, ключевыми атрибутами являются атрибуты «Фамилия» и «Группа», то инвертированные списки выглядят так, как показано на рис. 1.28.

Системой поддерживаются два класса операций над данными:

- поиск адреса записи по некоторому пути доступа и некоторому условию;

- обновление, удаление или выборка записи с заданным адресом.

Типичный набор операторов:

- LOCATE FIRST – найти первую запись таблицы Т в физическом порядке; возвращается адрес записи;

- LOCATE FIRST WITH SEARCH KEY EQUAL – найти первую запись таблицы Т с заданным значением ключа поиска К; возвращается адрес записи;

- LOCATE NEXT – найти первую запись, следующую за записью с заданным адресом в заданном пути доступа; возвращается адрес записи;

- LOCATE NEXT WITH SEARCH KEY EQUAL – найти следующую запись таблицы Т в порядке пути поиска с заданным значением К; при этом должно быть установлено соответствие между используемым способом сканирования и ключом К; возвращается адрес записи;

- LOCATE FIRST WITH SEARCH KEY GREATER – найти первую запись таблицы Т в порядке ключа поиска К со значением ключевого поля, большим заданного значения К; возвращается адрес записи;

- RETRIEVE – выбрать запись с указанным адресом;

- UPDATE – обновить запись с указанным адресом;

- DELETE – удалить запись с указанным адресом;

- STORE – включить запись в указанную таблицу; операция генерирует адрес записи.

К достоинствам рассмотренного метода построения базы данных следует отнести:

- более быстрый поиск (особенно поиск уникальной записи по нескольким условиям);

- возможность хранения элементов данных со сложной структурой.

Недостаток модели – отсутствие строгого математического аппарата, отсутствие средств для описания ограничений целостности базы данных и, как следствие – большая трудоемкость программирования запросов к базе данных. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу. Кроме этого, такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей.

Однако СУБД Adabas хороша в задачах, в которых надо из очень большого объема данных выбрать по сложному запросу относительно небольшое количество данных (например, найти автомобиль, иномарка, то ли серого, то ли голубого цвета, седан, номер московский, цифровая его часть оканчивается на 78, водитель среднего возраста, в очках).

16. Реляционная модель данных. Отношение реляционной модели данных. Основные понятия реляционной модели. Определение домена, семантическая нагрузка понятия домена. Фундаментальные свойства отношений. Базовые свойства реляционной модели данных. Обоснование требования отсутствия кортежей-дубликатов, отсутствия упорядоченности кортежей, упорядоченности атрибутов. Требования структурной, манипуляционной, целостной частей реляционной модели данных. Возможные подходы для выполнения требования целостности по ссылкам. Процедура каскадного удаления. Null-значения. Правила Кодда

Реляционная база данных – это конечное множество отношений, записываемое в виде:

$$R_1(A_{11}, A_{12} \dots A_{1n}) \quad \text{где } A_{ij} \text{ пара } \{\text{имя атрибута, имя домена}\}$$

Отношение R на множествах D1, D2, ..., Dn – это некоторое подмножество кортежей арности n декартового произведения доменов D1 × D2 × ... × Dn.

Основные понятия реляционной модели

Домен – допустимое множество значений атрибута. Семантическая нагрузка подразумевает, что данные считаются сравнимыми тогда и только тогда, когда они принадлежат одному домену.

Атрибут – поименованная характеристика некоторой сущности.

Имеется совокупность множеств D1, D2, ..., Dn, не обязательно различных, называемых доменами n-арного отношения. Совокупность значений , где $d_i \in D_i$, $i = 1, n$ называется **кортежем**.

Схема отношения (заголовок отношения) – именованное конечное множество упорядоченных пар вида $\langle A, T \rangle$, где A является именем атрибута, а T обозначает имя некоторого базового типа или ранее определенного домена.

Число атрибутов отношения носит название **степени или «арности»** схемы отношения

Фундаментальные свойства отношений

1. Отсутствие кортежей-дубликатов.

Свойство, что тело любого отношения никогда не содержит **кортежей-дубликатов**, следует из определения тела отношения как множества кортежей. В классической теории множеств по определению любое множество состоит из различных элементов.

2. Отсутствие упорядоченности кортежей.

Хранить упорядоченные списки кортежей в условиях интенсивно обновляемой базы данных гораздо сложнее технически, а поддержка упорядоченности влечет за собой существенные накладные расходы. Отсутствие требования к поддержанию порядка на множестве кортежей отношения придает СУБД дополнительную гибкость при хранении баз данных во внешней памяти и при выполнении запросов к базе данных.

3. Отсутствие упорядоченности атрибутов.

Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}.

4. Атомарность значений атрибутов.

Базовым требованием для классических реляционных баз данных является требование нормализованных отношений или отношений, представленных в первой нормальной форме. А отношения, представленные в первой нормальной форме, предполагают атомарность (неделимость) атрибутов.

Реляционный термин	Описание
База данных	Набор таблиц и других объектов, необходимых для абстрактного представления части реального мира
Схема базы данных	Набор заголовков таблиц, взаимосвязанных друг с другом
Отношение	Совокупность объектов реального мира, которые характеризуются общими свойствами и характеристиками (поля таблицы)
Заголовок отношения	Названия полей (столбцов) таблицы
Тело отношения	Совокупность значений для всех объектов реального мира, которая представима в виде записей таблицы (строки таблицы)
Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
Атрибут отношения	Наименование столбца таблицы (поле таблицы)
Кортеж отношения	Однозначное представление объекта реального мира, созданное с использованием значений полей таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в записи
Первичный ключ	Один или несколько атрибутов, который уникальным образом определяет значение кортежа (значение строки таблицы)
Внешний ключ	Атрибут таблицы, значения которого соответствуют значениям первичного ключа в другой связанной таблице.
Степень (арность) отношения	Количество столбцов таблицы
Мощность отношения	Количество строк таблицы (количество кортежей)
Тип данных	Тип значений элементов таблицы
Базовое отношение	Отношение, которое содержит один или несколько столбцов, характеризующих свойства объекта, а также первичный ключ
Производное отношение	Не является базовым отношением, т. е. не характеризует свойства объекта и используется для обеспечения связей между другими таблицами
Связь	Устанавливает взаимосвязь между первичным ключом одной таблицы и внешним ключом другой таблицы

Требования структурной, манипуляционной, целостной частей реляционной модели данных.

Структурная часть: все данные должны быть представлены в виде отношений.

Манипуляционная часть: язык запросов к реляционной СУБД должен обладать не меньшей мощностью и выразительностью, чем язык реляционной алгебры или реляционного исчисления.

Целостная часть:

1. Целостность сущностей: Любое отношение должно обладать первичным ключом.
2. Целостность по ссылкам: отношение с определенным внешним ключом должно ссылаться на отношение, где такой же атрибут является первичным ключом, либо быть неопределенным.

Для соблюдения **целостности по сущностям** достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа.

Возможные подходы для выполнения требования целостности по ссылкам. Процедура каскадного удаления.

Для соблюдения целостности по ссылкам:

- при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа;
- при удалении кортежа из отношения для обеспечения целостности по ссылкам на практике существуют три подхода:
 - запрещается производить удаление кортежа, на который имеются ссылки; сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа;
 - при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным;
 - третий подход (каскадное удаление) состоит в том, что при удалении кортежа из основного отношения с некоторым значением внешнего ключа автоматически удаляются кортежи из связанного с ним отношения с таким же значением первичного ключа.

Null-значения.

Неопределенные значения (Null-значения) не принадлежат ни к какому типу данных и могут присутствовать среди значений любого атрибута, определенного на любом типе данных, если только это явно не запрещено при определении атрибута. Поскольку требование целостностей сущности означает, что первичный ключ должен полностью идентифицировать каждую сущность, в первичном ключе не допускаются неопределенные значения.

Если a – это значение некоторого типа данных или NULL, op – любая двуместная «арифметическая» операция этого типа данных (например, $+$), а lop – операция сравнения значений этого типа (например, $=$), то по определению:

```
a op NULL = NULL  
NULL op a = NULL  
a lop NULL = unknown  
NULL lop a = unknown
```

Здесь *unknown* – это третье значение логического, или булевского, типа, обладающее следующими свойствами:

```
NOT unknown = unknown  
true AND unknown = unknown  
true OR unknown = true  
false AND unknown = false  
false OR unknown = unknown
```

Правила Кодда

Э. Коду принадлежат 12 правил, которым должна удовлетворять реляционная база данных.

1. *Правило информации.* Вся информация на логическом уровне представляется только значениями в таблицах без использования указателей и индексов.

2. *Правило гарантированного доступа.* Каждый элемент таблицы должен быть доступен через комбинацию из имени таблицы, имени поля и ключа.

3. *Системная поддержка неопределенных значений.* Для представления отсутствующих данных с любым типом используются Null-значения.

4. *Динамический каталог, создаваемый на основе реляционной модели.* Метаданные (словари) формируются тем же языком, что и данные. Пользователи, обладающие соответствующими правами, могут

работать с метаданными с помощью того же реляционного языка, который они применяют для работы с основными данными.

5. *Правило исчерпывающего подъязыка данных.* В базе данных возможно использование нескольких языков взаимодействия с базой данных (например, язык вопросов-ответов), однако один, чаще всего SQL, должен быть главным и поддерживать все следующие элементы:

- определение данных,
- определение представлений,
- обработку данных (интерактивную и программную),
- задание условий целостности данных,
- идентификацию прав доступа,
- задание границ транзакций (начало, завершение и отмена).

6. *Правило обновления представления (view).* Все обновляемые представления должны обновлять и система.

7. *Ввод, обновление и удаление данных.* Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. *Физическая независимость данных.* Возможное изменение расположения базы данных, способов хранения данных не должно оказывать влияния на прикладные программы и пользователя.

9. *Логическая независимость данных.* При добавлении или удалении элементов (таблиц, полей) в структуре базы данных другие части базы данных остаются неизменными.

10. *Независимость условий целостности.* Должна существовать возможность определять условия целостности в языке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. *Независимость распределения.* В распределенных базах данных расположение данных независимо. Для пользователя такая база данных должна выступать как локальная база данных.

12. *Правило соблюдения правил.* Нельзя обходить ограничения, введенные с помощью языка SQL.

17. Язык реляционной алгебры и его свойства. Операции реляционной алгебры. Запись операций реляционной алгебры на языке SQL.

Язык реляционной алгебры и его свойства

Базируется на классической теории множеств.

Свойства (РА и РИ):

- Замкнуты относительно понятия отношения (т.е. любое выражение или формула интерпретируется как отношение)
- Механизмы РА и РИ эквивалентны (т.е. для любого допустимого выражения РА можно построить производящую такой же результат формулу РИ)

Операции реляционной алгебры. Запись операций реляционной алгебры на языке SQL

1. Объединение (UNION)

Производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов.

2. Разность (MINUS) (в SQL = EXCEPT)

Производится отношение, включающее множество кортежей, принадлежащих R, но не принадлежащих S.

3. Пересечение (INTERSECT)

Производится отношение, включающее множество кортежей, принадлежащих как R, так и S.

4. Декартово произведение (TIMES) (cross join)

Производится отношение, включающее все возможные пары кортежей.

5. Проекция (PROJECT) (в SQL = distinct)

Удаление дублируемых кортежей

6. Ограничение (WHERE)

7. Соединение (JOIN)

- а) эквисоединение (EQUIJOIN) (частный случай, когда соединение идет равенством)
- б) естественное соединение (NATURAL JOIN) (спроектированный результат эквисоединения)
- с) внешнее соединение (OUTER JOIN)

8. Деление (DIVIDE BY) (в SQL = group by)

18. Реляционное исчисление на кортежах. Основные понятия. Запись операций реляционного исчисления средствами языка реляционной алгебры. Реляционное исчисление на доменах.

Реляционное исчисление на кортежах

4.2.1 Реляционное исчисление с переменными-кортежами

$\{t \mid \Psi(t)\}$ – множество кортежей

$\Psi(t)$ – well-formed formula, wff

1. $R(t)$, где R - имя отношения.

2. $S[i] \theta U[j]$, где S, U - переменные-кортежи; θ - арифметический оператор;
 i, j - номера или имена компонентов в соответствующих кортежах.

3. $S[i] \theta \text{const.}$

Вхождение переменной x в предикате Ψ **связано**, если в Ψ оно находится в подформуле, начинающейся квантором \exists или \forall , за которым непосредственно следует x , иначе x входит в предикат **свободно**

$$\forall(x) (R(x,y) \vee (\exists y) (U(x,y,z) \wedge Q(x,y)))$$

~~$\{t \mid \neg R(t)\}$~~

Безопасное – такое выражение $Y(t)$, которое обеспечивает, что множество элементов t , удовлетворяют $\Psi(t)$ является непустым множеством $D(Y)$.

Теорема 1. Если E – выражение реляционной алгебры, то существует эквивалентное выражение в реляционном исчислении с переменными кортежами.

Операция объединения $R \cup S \rightarrow \{t \mid R(t) \vee S(t)\}$

Операция разности $R - S \rightarrow \{t \mid R(t) \wedge \neg S(t)\}$

Операция декартового произведения $T = R \times S \rightarrow \{t \mid \exists(u,w) (R(u) \wedge S(w) \wedge t = u \otimes w)\}$

Операция проекции $\Pi_{i_1, i_2, \dots, i_k}(R) \rightarrow \{t^{(k)} \mid \exists(u) (R(u) \wedge ((t[1]=u(i1)) \wedge \dots \wedge (t[k]=u(ik))))\}$

Операция экви соединения $R \text{ EQUIJOIN } S (i=j) \rightarrow$

$$\{t^{(k1+k2)} \mid \exists(u^{(k1)}, w^{(k2)}) (R(u) \wedge S(w) \wedge t=u \otimes w \wedge (u[i]=w[i]))\}$$

Получить имена и номера сотрудников, являющихся начальниками отделов с количеством сотрудников более 50

Отношение Сотрудники (Таб_номер, Сотр_имя, Сотр_зарпл, Отдел_ном) – $R(u)$

Отношение Отделы (Отдел_ном, Отдел_нач, Отдел_Кол) – $S(w)$

$$\{r^{(2)} \mid \exists(u, w) (R(u) \wedge S(w) \wedge (u[4]=w[1]) \wedge (w[3]>50) \wedge (r[1]=u[1]) \wedge (r[2]=w[2]))\}$$

4.2 Реляционное исчисление с переменными-кортежами

1. $S[i] \rightarrow S.[\text{имя_атрибута}]$.

2. Кванторы \forall и $\exists \rightarrow FORALL$ и $EXISTS$.

3. Общая форма выражения реляционного исчисления:

[список переменных-кортежей] WHERE [формула реляционного исчисления]

4. Перечень переменных-кортежей вводится фразой

[список переменных-кортежей] RANGE IS [имя отношения]

Сотрудник RANGE IS Сотрудники

Отдел RANGE IS Отделы

Сотрудник.Сотр_имя, Сотрудник.Таб_номер

WHERE (Сотрудник.Сотр_имя

=Отдел.Отдел_нач

AND Отдел.Отдел_Кол > 50)

Задача: получить имена и таб. номера сотрудников, являющихся начальниками отделов с количеством сотрудников больше 50

СЛУ1, СЛУ2 RANGE IS СЛУЖАЩИЕ
НОМЕР_ПРОЕКТА RANGE IS НОМЕРА_ПРОЕКТОВ
СЛУ1.СЛУ_НОМЕР, СЛУ1.СЛУ_ИМЯ, СЛУ1.СЛУ_ЗАРП
WHERE FORALL НОМЕР_ПРОЕКТА

EXISTS СЛУ2
(СЛУ1.СЛУ_НОМЕР = СЛУ2.СЛУ_НОМЕР AND
СЛУ1.ПРО_НОМ = НОМЕРА_ПРОЕКТОВ.ПРО_НОМ)

Отношение НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22400.00
2935	Петров	29600.00
2936	Сидоров	18000.00
2937	Федоров	20000.00
2938	Иванова	22000.00
2934	Иванов	22400.00
2935	Петров	29600.00
2939	Сидоренко	18000.00
2940	Федоренко	20000.00
2941	Иваненко	22000.00

СЛУЖАЩИЕ DIVIDE BY НОМЕРА_ПРОЕКТОВ

Это какая- то дичь, есть еще ссылка, но там немного другое

http://citforum.ru/database/advanced_intro/17.shtml#6.2

Реляционное исчисление на доменах

4.2.2 Реляционное исчисление с переменными-доменами

$\{x_1, x_2, \dots, x_k, | \Psi(x_1, x_2, \dots, x_k)\}$

1. $R(x_1, x_2, \dots, x_k)$, где R - k -арное отношение, x_i - константа или переменная на некотором домене.

2. $x \Theta y$, где x, y - переменные или константы на домене.

Домен	Переменная домена x_1, x_2, \dots, x_k
n_post	SX, SY – переменные на одном домене
n_det	PX, PY
$name$	$NAMEX, NAMEY$
$cvet$	$CVETX, CVETY$
$reiting$	$REITINGX, REITINGY$
$town$	$TOWNX, TOWNY$

$\{x_1, x_2 | R_1(x_1, x_2) \wedge (\forall y)(\neg R_2(x_1, y) \wedge \neg R_2(x_2, y))\}$

Теорема 1. Для любого безопасного выражения реляционного исчисления с переменными-кортежами существует эквивалентное безопасное выражение реляционного исчисления с переменными-доменами.

Теорема 2. Для любого безопасного выражения реляционного исчисления с переменными на доменах существует эквивалентное выражение реляционной алгебры.

Задача Найти имена менеджеров, зарплата которых превышает 25000 единиц.

$\{fname, lname | \exists position, salary$

$(Staff(fname, lname, position, salary) \wedge position = 'Manager' \wedge salary > 25000)\}$

Примеры записи формул реляционного исчисления на доменах:

SX – множество всех номеров поставщиков;

$SX WHERE S(n_post:SX)$ – множество всех номеров поставщиков в отношении S ;

$SX WHERE S(n_post:SX, town:'Лондон')$ – множество номеров поставщиков из Лондона;

$SX WHERE EXISTS REITINGX (S(n_post:SX, reiting:REITINGX, CITY:'Париж') AND REITINGX > 20)$
– номера поставщиков из Парижа с рейтингом, большим 20

$NAMEX WHERE EXISTS SX EXISTS PX (S(n_post:SX, name:NAMEX) AND SP(n_post:SX, n_det:PX)$
 $AND P(n_det:PX, cvet:'Красный'))$ – имена поставщиков хотя бы одной красной детали

http://citforum.ru/database/advanced_intro/18.shtml#6.3

19. Язык SQL. Стандартизация языка SQL. Структура операторов SQL: Select, Insert, Update, Delete. Подзапросы. Соединения. Виды соединений. Основные правила использования конструкций языка SQL. Представления. Оптимизация запросов SQL. Достоинства и недостатки языка SQL.

Стандартизация SQL

Международный стандарт 1989 г.

Международный стандарт 1992 г. (SQL2)

Стандарт SQL:1999 (SQL3)

1. *SQL/Framework*
2. *SQL/Foundation*
3. *SQL/CLI*
4. *SQL/PSM*
5. *SQL/Bindings*

В конце 2003 г. был принят и опубликован новый вариант **международного стандарта SQL:2003**

Структура операторов SQL: Select, Insert, Update, Delete.

Запрос на языке SQL формируется с использованием оператора Select.

Оператор Select используется

- для выборки данных из базы данных;
- для получения новых строк в составе оператора Insert;
- для обновления информации в составе оператора Update.

Общая форма оператора вставки.

Insert into таблица [(поле [,поле]...)]

values (константа [,константа]...) или подзапрос

Общая форма оператора обновления

Update таблица

set поле=выражение [,поле=выражение]...[where предикат]

Общая форма оператора удаления:

delete from таблица [where предикат]

Подзапросы.

Оператор select, вложенный в спецификатор where другого оператора select (или одного из операторов insert, delete, update), называется подзапросом. В состав каждого подзапроса должны входить спецификаторы select и from. Кроме того, каждый подзапрос должен быть заключен в круглые скобки, чтобы указать серверу баз данных на то, что эту операцию следует выполнить первой.

Подзапросы бывают **коррелированными и некоррелированными**. Подзапрос является коррелированным, если его значение зависит от значения, производимого внешним оператором select, который содержит этот подзапрос. Любой другой вид запроса называется некоррелированным.

Важное свойство коррелированного подзапроса состоит в следующем: так как он зависит от значения результата внешнего оператора select, то должен выполняться повторно по одному разу для каждого значения, производимого внешним оператором select. Некоррелированный подзапрос выполняется только один раз.

Подзапрос включается в спецификатор where оператора select с помощью следующих ключевых слов: ALL, ANY, IN, EXISTS

Соединения. Виды соединений.

Соединение

Операция соединения представляет собой соединение кортежей соединяемых таблиц по указанному условию соединения. Строки, которые не удовлетворяют условиям соединения, отбрасываются.

Операция экви соединения является частным случаем операции соединение по условию равенства атрибутов. Кроме этого существует практически важное расширение операции экви соединения – естественное (внешнее) соединение.

INNER JOIN

Запрос с оператором INNER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON.



То же самое делает и просто JOIN. Таким образом, слово INNER - не обязательное.

LEFT OUTER JOIN (левое внешнее соединение)

Запрос с оператором LEFT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой по порядку (левой) таблицы, даже если они не соответствуют условию. У записей левой таблицы, которые не соответствуют условию, значение столбца из правой таблицы будет NULL (неопределенным).



RIGHT OUTER JOIN (правое внешнее соединение)

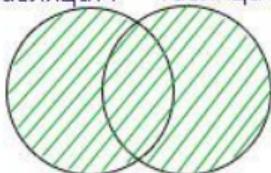
Запрос с оператором RIGHT OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из второй по порядку (правой) таблицы, даже если они не соответствуют условию. У записей правой таблицы, которые не соответствуют условию, значение столбца из левой таблицы будет NULL (неопределенным).



FULL OUTER JOIN (полное внешнее соединение)

Запрос с оператором FULL OUTER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON, и дополняются записями из первой (левой) и второй (правой) таблиц, даже если они не соответствуют условию. У записей, которые не соответствуют условию, значение столбцов из другой таблицы будет NULL (неопределённым).

Таблица А Таблица В



Основные правила использования конструкций языка SQL.

4.3.1 Основные правила использования SQL

SQL-запрос → план выполнения запроса → ✓скорость выполнения запроса;
✓влияние выполняемого запроса на выполнение параллельно выполняемых запросов и работоспособность системы в целом

1. Грамотное проектирование базы данных.

Скорость выполнения запроса зависит от:

- размеров таблиц;
- структуры таблиц;
- степени нормализации таблиц;
- типы данных ключей

2. Знание источников данных и бизнес-приложения.

3. Для фильтрации записей используйте WHERE, а не HAVING.

```
SELECT DEPTID, SUM(SALARY)
  FROM EMP
 GROUP BY DEPTID
 HAVING DEPTID = 100;
```



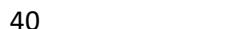
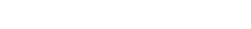
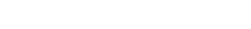
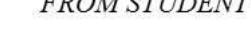
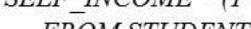
```
SELECT DEPTID, SUM(SALARY)
  FROM EMP
 WHERE DEPTID = 100
 GROUP BY DEPTID
```

4. Минимизируйте число просмотров таблиц

```
SELECT NAME, PARENT_INCOME
  FROM STUDENT
 WHERE STATUS = 1
 UNION
SELECT NAME, SELF_INCOME
  FROM STUDENT
 WHERE STATUS = 0
```



```
SELECT NAME, PARENT_INCOME * STATUS +
      SELF_INCOME * (1 - STATUS)
  FROM STUDENT
```



4.3.1 Основные правила использования SQL

5. Используйте специальные столбцы.

```
SELECT ROWID, SALARY  
    INTO TEMP_ROWID, TEMP_SALARY  
        FROM EMPLOYEE;  
UPDATE EMPLOYEE  
    SET SALARY = TEMP_SALARY * 1.5  
    WHERE ROWID = TEMP_ROWID
```

6. Где возможно, избегайте использования соединений

Select lname, fname from Persons, Companies where Persons.company=Companies.Company_id and Company.name="Sony"	↔	Select lname, fname from Persons Where Persons.company in (Select company_id from Companies where Company.name="Sony")
--	---	---

7. Если это неизбежно, соединяйте таблицы в правильном порядке.

- отфильтровать как можно большее число строк на ранних фазах выполнения запроса с соединениями;
- главная таблица (просматриваемая во внешнем цикле соединения на основе вложенных циклов) содержит наименьшее число строк.

8. Зачастую выгоднее разбить сложный запрос на несколько простых с использованием, возможно, временных таблиц.

9. Страйтесь избегать коррелированных запросов .

Порядок предпочтения:

- некоррелированные запросы;
- соединения;
- коррелированные запросы.

Представления.

Понятие представляемой таблицы, или представления, которую можно использовать в операторе выборки наряду с базовыми таблицами.

оператор создания представления в общем случае определяется следующими синтаксическими правилами:

```
create_view ::= CREATE [ RECURSIVE ] VIEW table_name  
    [ column_name_comma_list ]  
    AS query_expression  
    [ WITH [ CASCDED | LOCAL ] CHECK OPTION ]
```

Простая форма:

```
create_view ::= CREATE VIEW table_name  
    [ column_name_comma_list ]  
    AS query_expression
```

Имя таблицы, задаваемое в определении представления, существует в том же пространстве имен, что и имена базовых таблиц, и, следовательно, должно отличаться от всех имен таблиц (базовых и представляемых), созданных тем же пользователем. Если имя представления встречается в разделе FROM какого-либо оператора выборки, то вычисляется выражение запроса, указанное в разделе AS, и оператор выборки работает с результирующей таблицей этого выражения запроса.133) Явное указание имен столбцов представляемой таблицы требуется в том случае, когда эти имена не выводятся из соответствующего выражения запроса.

Как и для всех других вариантов оператора CREATE, для CREATE VIEW имеется обратный оператор DROP VIEW table_name, выполнение которого приводит к отмене определения представления (реально это выражается в удалении данных о представлении из таблиц-каталогов базы данных). После выполнения операции пользоваться представлением с данным именем становится невозможно.

Оптимизация запросов SQL.

1. Индексирование
2. Изменение порядка соединения
3. Вертикальная и горизонтальная фрагментация таблицы

Достоинства и недостатки языка SQL.

4.3.3 Достоинства и недостатки SQL

Достоинства:

- повсеместная распространенность;
- быстрое обучение в простых случаях;
- связывание с различными языками программирования;
- поддержка ODBC и JDBC;
- эффективность современных средств обработки запросов.

Недостатки:

- несоответствие реляционной модели данных;
- недостаточно продуманный механизм неопределенных значений;
- сложность формулировок и громоздкость.

20. Жизненный цикл базы данных. Основные этапы жизненного цикла.

Понятие бизнес-процесса.

Жизненный цикл базы данных

Как и любой программный продукт, база данных обладает собственным жизненным циклом (ЖЦ БД). Главной составляющей в жизненном цикле БД является создание единой базы данных и программ, необходимых для ее работы. Жизненный цикл системы базы данных определяет и жизненный цикл всей информационной системы организации, поскольку база данных является фундаментальным компонентом информационной системы.

ЖЦ БД включает в себя следующие основные этапы:

- планирование разработки базы данных;
- определение требований к системе;
- сбор и анализ требований пользователей;
- проектирование базы данных:
- концептуальное проектирование базы данных;
- логическое проектирование базы данных;
- физическое проектирование базы данных;
- разработка приложений;
- реализация;
- загрузка данных;
- тестирование;
- эксплуатация и сопровождение.

Планирование разработки базы данных

Содержание данного этапа - разработка стратегического плана, в процессе которой осуществляется предварительное планирование конкретной системы управления базами данных. Планирование разработки базы данных состоит в определении трех основных компонентов: объема работ,

ресурсов и стоимости проекта. Планирование разработки БД должно быть связано с общей стратегией построения информационной системы организации. Важной частью разработки стратегического плана является проверка осуществимости проекта, состоящая из проверки технологической осуществимости, проверки операционной осуществимости, проверки экономической целесообразности осуществления проекта.

Определение требований к системе

На данном этапе необходимо определить диапазон действия приложения БД, состав его пользователей и области применения. Определение требований включает выбор целей БД, выяснение информационных потребностей различных отделов и руководителей фирмы и требований к оборудованию и программному обеспечению. При этом также требуется рассмотреть вопрос, следует ли создавать распределенную базу данных или же централизованную, и какие в рассматриваемой ситуации понадобятся коммуникационные средства.

Сбор и анализ требований пользователей

Этот этап является предварительным этапом концептуального проектирования базы данных. Проектирование базы данных основано на информации о той части организации, которая будет обслуживаться базой данных. Информационные потребности выясняются с помощью анкет, опросов менеджеров и работников фирмы, с помощью наблюдений за деятельностью предприятия, а также отчетов и форм, которыми фирма пользуется в текущий момент.

Проектирование базы данных

Полный цикл разработки БД включает концептуальное, логическое и физическое ее проектирование. Основными целями проектирования базы данных являются:

- представление данных и связей между ними, необходимых для всех основных областей применения данного приложения и любых существующих групп его пользователей;
- создание модели данных, способной поддерживать выполнение любых требуемых транзакций обработки данных;
- разработка предварительного варианта проекта, структура которого позволяет удовлетворить требования, предъявляемые к производительности системы.

В создании БД как модели предметной области выделяют:

- объектную (предметную) систему, предъявляющую фрагмент реального мира;
- информационную систему, описывающую некоторую объектную систему;
- даталогическую систему, представляющую информационную систему с помощью данных.

Оптимальная модель данных должна удовлетворять таким критериям, как структурная достоверность, простота, выразительность, отсутствие избыточности, расширяемость, целостность, способность к совместному использованию.

Разработка приложений

Параллельно с проектированием системы БД выполняется разработка приложений. Главные составляющие данного процесса - это проектирование транзакций и пользовательского интерфейса.

Реализация

На данном этапе осуществляется физическая реализация базы данных и разработанных приложений, позволяющих пользователю формулировать требуемые запросы к БД и манипулировать данными в БД. На этом этапе реализуются также используемые приложением средства защиты базы данных и поддержки ее целостности. Реализация этого, а также и более ранних этапов проектирования БД может осуществляться с помощью инструментов автоматизированного проектирования и создания программ, которые принято называть CASE-инструментами. Использование CASE-инструментов способствует повышению производительности труда разработчиков и обеспечению эффективности самой разрабатываемой системы.

Загрузка данных

На этом этапе созданные в соответствии со схемой базы данных пустые файлы, предназначенные для хранения информации, должны быть заполнены данными. Наполнение базы данных может протекать по-разному, в зависимости от того, создается ли база данных вновь или новая база данных предназначена для замены старой.

В первом случае для ввода информации используются созданные в процессе проектирования БД удобные специальные формы, которые позволяют администратору базы данных занести в базу заранее подготовленные данные.

Если же новая база данных предназначена для замены старой, то возникает проблема переноса данных в новую систему, причем очень часто с преобразованием формата их представления. Данная задача получила название - *конвертирование данных*. В настоящее время любая система управления базами данных имеет утилиту загрузки уже существующих файлов в новую базу данных.

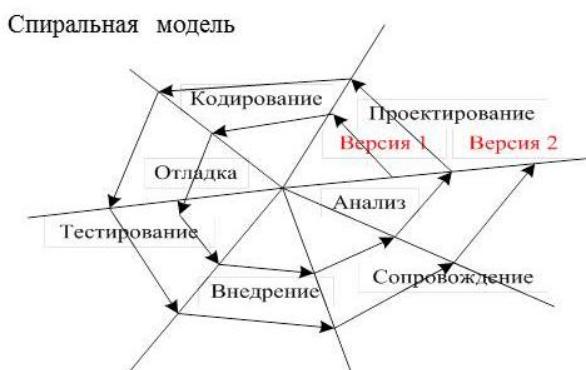
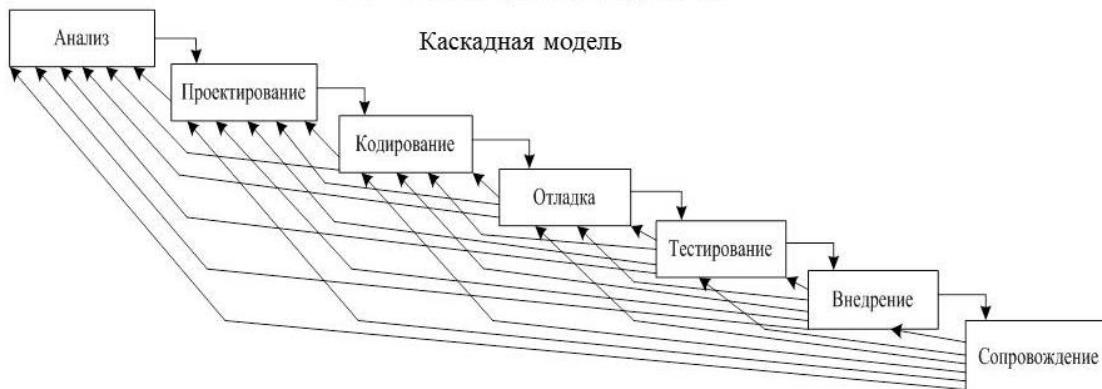
Тестирование

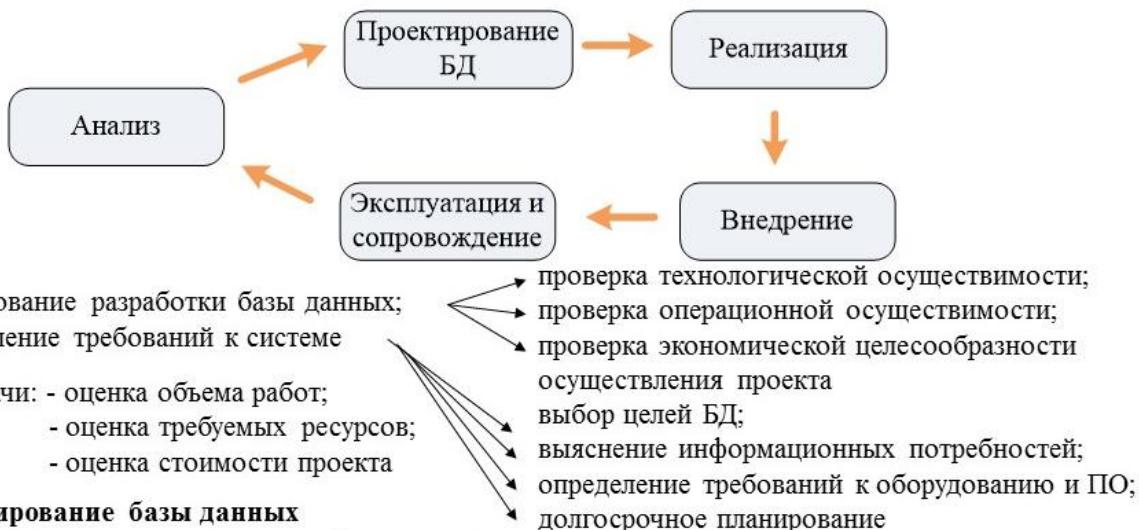
Тщательное тестирование должен проходить любой программный продукт тем более такой, как прикладные программы информационной системы. Стратегия тестирования должна предполагать использование реальных данных и должна быть построена таким образом, чтобы весь процесс выполнялся строго последовательно и методически правильно. Помимо обнаружения имеющихся в прикладных программах и, возможно, в структурах базы данных ошибок, сбор статистических данных на стадии тестирования позволяет установить показатели надежности и качества созданного программного обеспечения.

Эксплуатация и сопровождение

Данный этап является последним, но, безусловно, и самым продолжительным в жизненном цикле правильно спроектированной базы данных. Основные действия, связанные с этим заключительным этапом, сводятся к наблюдению за созданной системой, поддержке ее нормального функционирования, а также к созданию дополнительных программных компонент или модернизации самой базы данных.

Жизненный цикл базы данных





Проектирование БД – содержимое базы данных;

- эффективный для пользователей способ организации данных;
- инструментальные средства управления данными.

Требования к проекту базы данных

- корректность схемы базы данных;
- обеспечение целостности;
- защита данных;
 - восстанавливаемость данных;
 - согласованность методов модификации данных;
 - безопасность данных;
- эффективность функционирования;
- простота и удобство в эксплуатации;
- гибкость.

Бизнес-процесс – связанный между собой совокупность бизнес-функций, в ходе которых потребляются определенные ресурсы и создается продукт (товар, услуга), представляющая ценность для потребителя

1. Определение сферы применения БД
2. Определение производственных функций
3. Определение функций управления
4. Составление списка правил поведения
5. Составление списка возможных будущих изменений
6. Выявление и устранение противоречивости данных
7. Описание целей создания БД
8. Формулировка перечня задач



21. Назначение методологии IDEF0. Виды связей в IDEF0. Основные понятия и конструкции диаграмм работ. Назначение методологии диаграмм потоков данных. Основные понятия и конструкции диаграмм потоков данных. Методология объектного проектирования на языке UML. Диаграммы прецедентов. Основные понятия и конструкции. Диаграммы деятельности. Основные понятия и конструкции. CASE-средства. Классификация CASE-средств.

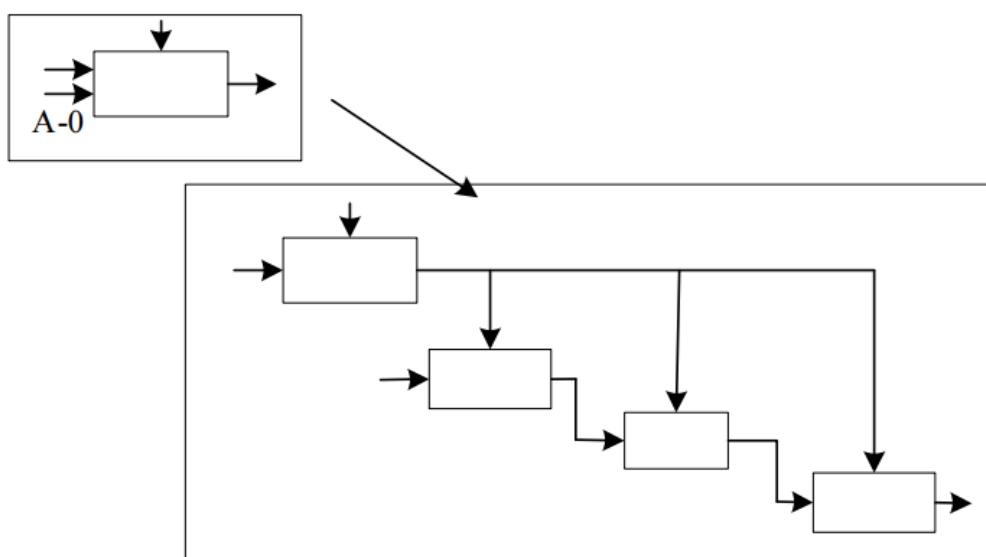
Виды моделей описания бизнес процессов

1. Методология функционального моделирования работ SADT (Structured Analysis and Design Technique)
2. Диаграммы потоков данных DFD (Data Flow Diagrams);
3. Методология объектного проектирования на языке UML (UML-диаграммы):
 - диаграмма прецедентов (Use case diagram);
 - диаграмма последовательностей (Sequence diagram);
 - диаграмма деятельности (Activity diagram);
 - диаграмма кооперации (Collaboration diagram);
 - диаграмма классов (Class diagram);
 - диаграмма компонентов (Component diagram);
 - диаграмма состояний (Statechart diagram);
 - диаграмма развертывания (Deployment diagram);

1. Методология SADT (IDEF0 (Icam DEFinition))

В методологии функционального моделирования работ Structured Analysis and Design Technagie (SADT) система представляется как совокупность взаимодействующих *работ* (или *функций*). Связи между работами определяют технологический процесс, или структуру взаимосвязи внутри организации. Содержательно модель SADT – серия диаграмм, разбивающих сложный объект на составные части.

Работа (*activity*) – поименованный процесс, функция или задача (изображается в виде прямоугольника), которые происходят в течение определенного времени и имеют распознаваемый результат. Каждая из работ на диаграммах следующего уровня может быть декомпозирована на несколько блоков, соединенных интерфейсными дугами. Эти блоки называют *подфункциями* (*подмодулями*) исходной функции. Каждый из подмодулей может быть декомпозирован аналогичным образом



Взаимодействие работ с внешним миром и между собой описывается в виде стрелок, имеющих следующий смысл.

Вход (Input) – материал или информация, которые используются работой для получения результата (стрелка, входящая в левую грань).

Управление (Control) – правила, стратегии, стандарты, которыми руководствуется работа (стрелка, входящая в верхнюю грань). В отличие от входной информации управление не подлежит изменению.

Выход (Output) – материал или информация, которые производятся работой (стрелка, исходящая из правой грани). Каждая работа должна иметь хотя бы одну стрелку выхода, поскольку работа без результата не имеет смысла и не должна моделироваться.

Механизм (Mechanism) – ресурсы, которые выполняют работу (персонал, станки, устройства – стрелка, входящая в нижнюю грань).

Вызов (Call) – стрелка, указывающая на другую модель работы (стрелка, исходящая из нижней грани).

1. Методология SADT

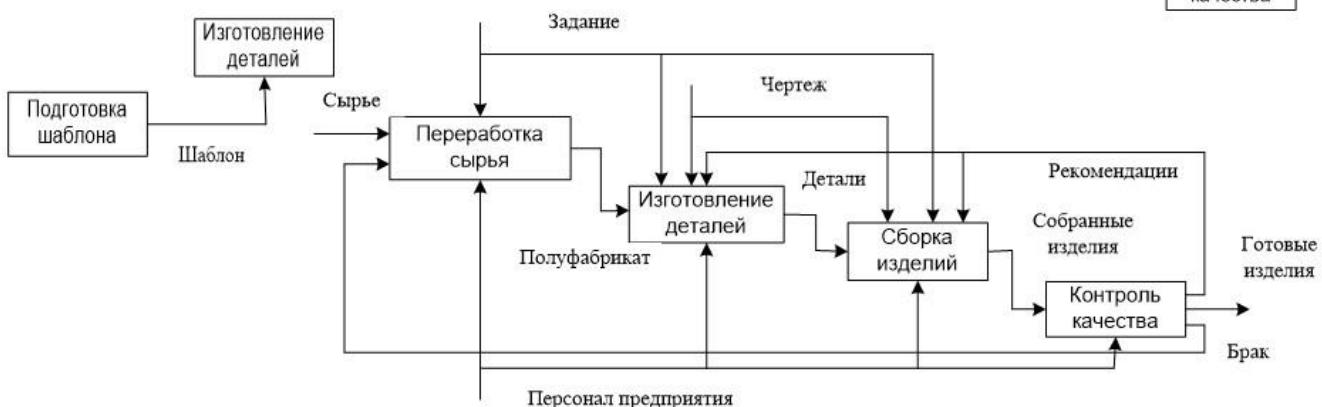
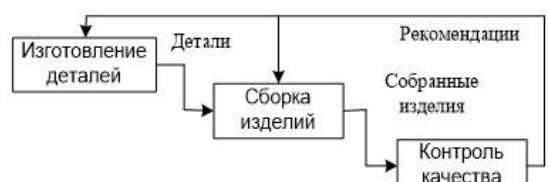
Связь по входу (input-output), когда выход вышестоящей работы направляется на вход следующей работы.

Связь по управлению (output-control), когда выход вышестоящей работы направляется на управление следующей работы.

Обратная связь по входу (output-input feedback), когда выход нижестоящей работы направляется на вход вышестоящей.

Обратная связь по управлению (output-control feedback), когда выход нижестоящей работы направляется на управление вышестоящей.

Связь выход-механизм (output-mechanism), когда выход одной работы направляется на механизм другой и показывает, что работа подготавливает ресурсы для проведения другой работы.

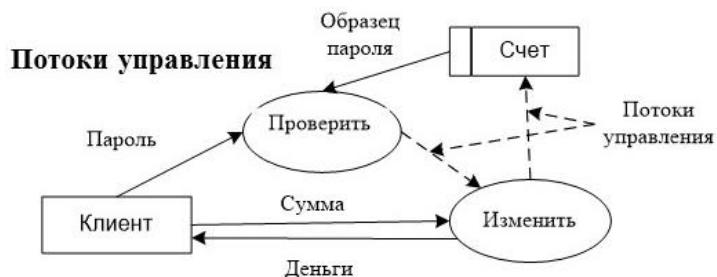
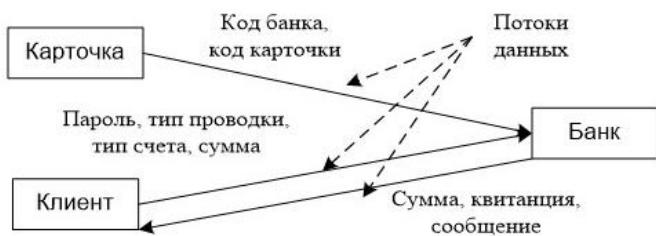


2. Методология DFD (Data Flow Diagrams)

Объекты диаграммы потоков данных:

- процессы;
- потоки данных;
- активные объекты;
- хранилища данных;
- потоки управления

Активные объекты



Процессы и потоки данных



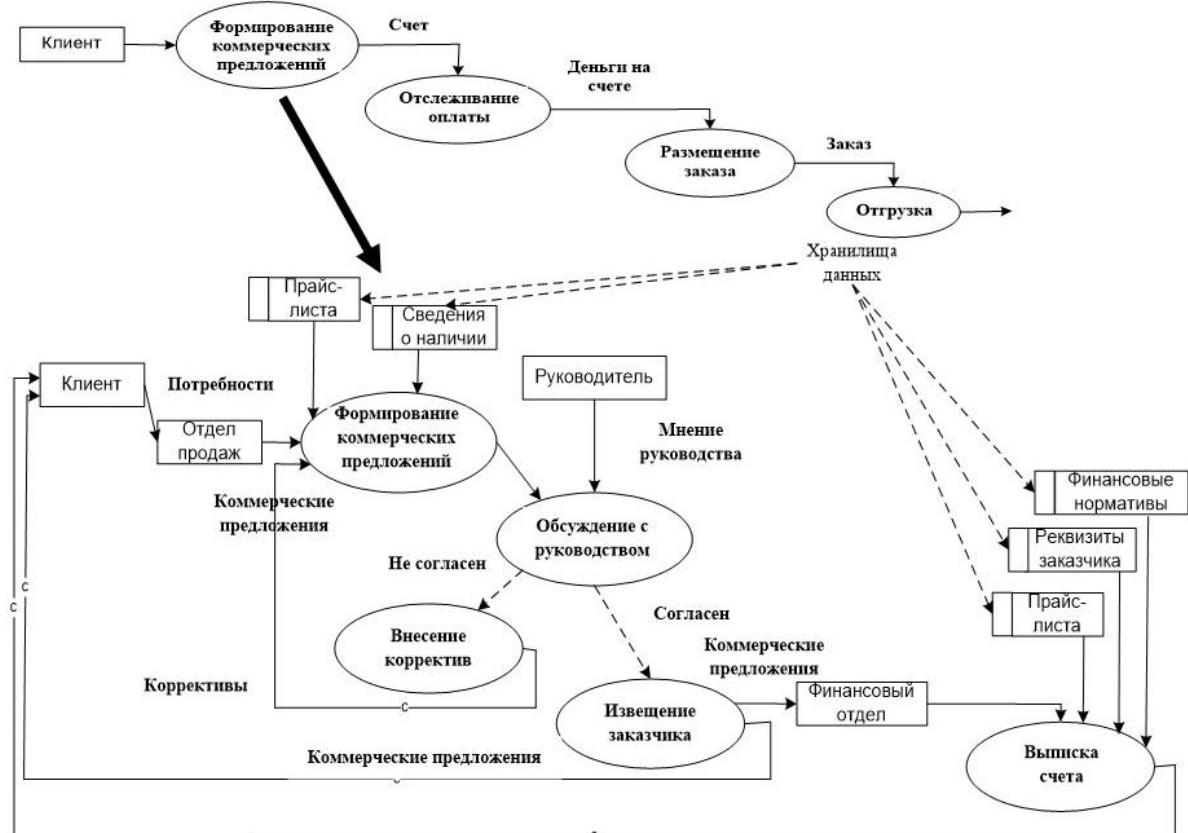
Хранилища данных



Назначение методологии потоков данных.

Диаграммы потоков данных (Data Flow Diagrams – DFD) используются для описания движения документов и обработки информации как дополнение к модели SADT. Стрелки в DFD-модели показывают, как объекты и данные движутся от одной работы к другой.

2. Методология DFD (Data Flow Diagrams)

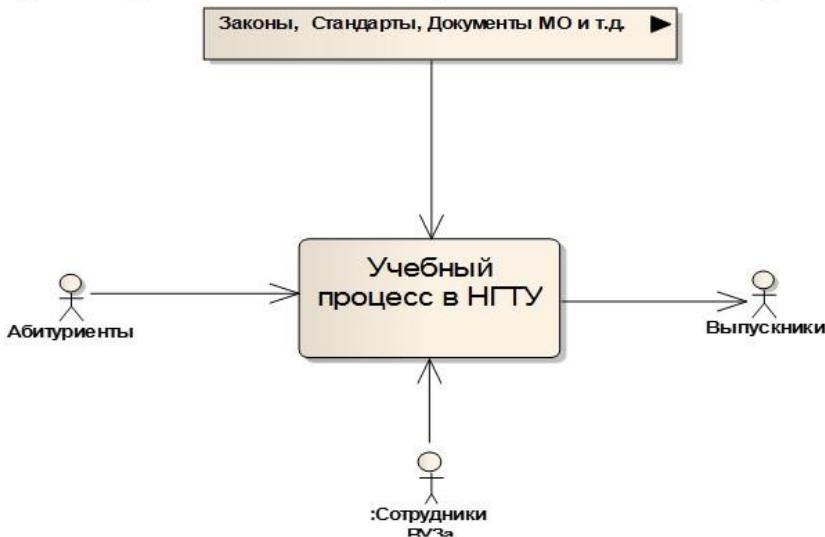


3. Методология объектного проектирования на языке UML (UML-диаграммы)

Модели методологии объектного проектирования на языке UML :

- диаграмма прецедентов (Use case diagram);
- диаграмма деятельности (Activity diagram);
- диаграмма классов (Class diagram);
- диаграмма состояний (Statechart diagram);
- диаграмма последовательностей (Sequence diagram);
- диаграмма кооперации (Collaboration diagram);
- диаграмма компонентов (Component diagram);
- диаграмма развертывания (Deployment diagram);

Пример проектирования бизнес-процесса “Учебный процесс”: уровень 1



1.3.1. Диаграммы прецедентов

Диаграмма прецедентов (*Use case diagram*) – исходная концептуальная модель системы в процессе ее проектирования и разработки. Разработка диаграммы прецедентов преследует цели:

- формулирования общих требований к функциональному поведению проектируемой системы;
- разработки исходной концептуальной модели системы для ее последующей детализации в форме логических и физических моделей;
- подготовка исходной документации для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграммы прецедентов применяются для моделирования вида системы с точки зрения внешнего наблюдателя. На диаграмме прецедентов графически показана совокупность *прецедентов* и *субъектов*, а также отношения между ними.

Субъект (actor) – любая сущность (человек, техническое устройство, программа и т. д.), взаимодействующая с системой извне, или множество логически связанных ролей, исполняемых при взаимодействии с прецедентами. Графически субъект на диаграммах обозначается фигурой человека, под которой записывается конкретное имя субъекта

Прецеденты (use case) – это описание последовательностей действий (включая их варианты), которые выполняются системой для того, чтобы субъект получил результат, имеющий для него определенное значение. Графически прецедент на диаграммах обозначается эллипсом, внутри которого содержится краткое название прецедента или имя в форме глагола с пояснительными словами.

Между субъектами и прецедентами – основными компонентами диаграммы прецедентов могут существовать различные отношения, которые описывают взаимодействие экземпляров одних субъектов и прецедентов с экземплярами других субъектов и прецедентов. В языке UML имеется несколько стандартных видов отношений между субъектами и прецедентами;

- **отношение ассоциации (association)**, обозначаемое сплошной линией, определяет наличие канала связи между экземплярами субъекта и прецедента (или между экземплярами двух субъектов);
- **отношение расширения (extend)**, обозначаемое пунктирной линией, направленной от того прецедента, который является расширением для прецедента исходного. Помечается ключевым словом *extend* (*расширять*), определяет взаимосвязь экземпляров отдельного прецедента с более общим прецедентом, свойства которого определяются на основе способа совместного объединения данных экземпляров;

- **отношение включения (include)**, обозначаемое пунктирной линией со стрелкой, направленной от базового прецедента к включаемому, и помечаемое ключевым словом *include* (*включать*). Указывает, что некоторое заданное поведение для одного прецедента включает в качестве составного компонента поведение другого прецедента;
- **отношение обобщения (generalization)**, обозначаемое сплошной линией с незакрашенной стрелкой которая указывает на родительский прецедент, сообщает что некоторый прецедент А может быть обобщен до прецедента В.

1.3.2. Диаграммы деятельности

Диаграмма деятельности (Activity diagram) – это по существу блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой. От традиционной блок-схемы диаграмма деятельности отличается более высоким уровнем абстракции и возможностью демонстрации управления параллельными потоками (наряду с последовательным управлением).

Разработка диаграммы деятельности необходима:

- для детализации особенностей алгоритмической и логической реализации выполняемых системой операций и прецедентов;
- выделения последовательных и параллельных потоков управления;
- подготовки детальной документации для взаимодействия разработчиков системы с ее заказчиками и проектировщиками.

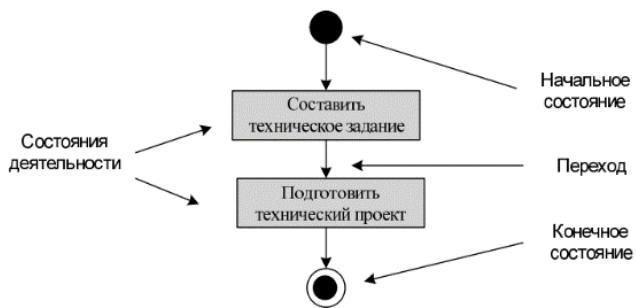
Диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия (или состояния деятельности), а дугами – переходы от одного состояния действия/деятельности к другому. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное – в ее нижней части. Ниже представлены основные элементы диаграммы деятельности.

Состояние деятельности (Activity, Process) – это продолжающийся во времени неатомарный шаг вычислений в автомате. Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности.

Состояние действия (Action state) – это вычисление атомарного действия, как правило – вызов операции. Состояния действия атомарны и не могут быть подвергнуты декомпозиции.

Состояния деятельности и состояния действия имеют одинаковое стандартное графическое обозначение – прямоугольник с закругленными краями. Внутри такого символа записывают произвольное выражение (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Начальное и конечное состояния на диаграммах деятельности изображаются в виде закрашенного круга и закрашенного круга внутри окружности соответственно



Переход (Transitions) – отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить некоторые действия и перейти во второе состояние. На языке UML переход представляется простой линией со стрелкой.

Ветвление описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Графически точка ветвления представляется ромбом.

Разделение (Concurrent fork) и слияние (Concurrent join) служат для организации параллельных потоков, которые часто необходимы для моделирования бизнес-процессов. Графически на языке UML для обозначения разделения и слияния используются обозначения, приведенные на рис. 1.10.



Рис. 1.10. Разделение и слияние
в диаграммах деятельности

Дорожка (Swimline) служит для описания связанных работ, каждая из которых выполняется параллельно (рис. 1.11).

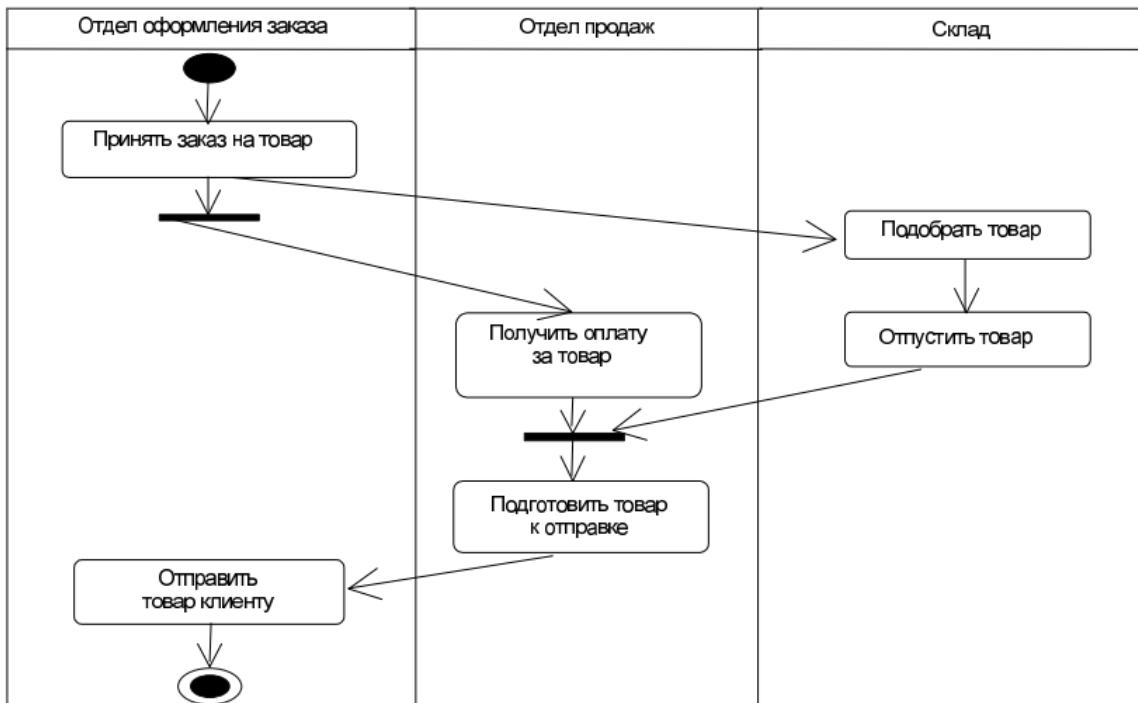


Рис. 1.11. Дорожки в диаграммах деятельности

CASE СРЕДСТВА

CASE средства используются при создании и разработке информационных систем управления предприятиями. Применительно к [моделирования бизнес процессов](#) они могут рассматриваться как инструментарий для совершенствования и непрерывного улучшения работы.

CASE средства (Computer - Aided Software Engineering) – это инструмент, который позволяет автоматизировать процесс разработки информационной системы и программного обеспечения. Разработка и создание информационных систем управления предприятием связаны с выделением бизнес-процессов, их анализом, определением взаимосвязи элементов процессов, оптимизации их инфраструктуры и т.д. Основной целью применения CASE средств является сокращение времени и затрат на разработку информационных систем, и повышение их качества.

Многие современные CASE средства предоставляют возможности для моделирования практически всех предметных областей деятельности организаций. В составе этих средств существуют инструменты для описания моделей бизнес-процессов за счет различных диаграмм, схем, графов и таблиц.

КЛАССИФИКАЦИЯ CASE СРЕДСТВ

Из всего многообразия CASE средств, существующих на сегодняшний день, можно выделить три основные группы. Эти группы связаны с этапами разработки информационных систем и их жизненным циклом. Классификация CASE средств осуществляется в зависимости от того, какие из этапов разработки они поддерживают.

Выделяют следующие группы CASE средств:

- **CASE средства верхнего уровня.** Эти CASE средства ориентированы на начальные этапы построения информационной системы. Они связаны с анализом и планированием. CASE средства верхнего уровня обеспечивают стратегическое планирование, расстановку целей, задач и приоритетов, а также графическое представление необходимой информации. Все CASE средства верхнего уровня содержат графические инструменты построения диаграмм, таких как диаграммы сущность-связь (ER диаграммы), диаграммы потока данных (DFD), структурные схемы, деревья решений и пр.
- **CASE средства нижнего уровня.** Эти CASE средства больше сфокусированы на последних этапах разработки информационной системы – проектирование, разработка программного кода, тестирование и внедрение. CASE средства нижнего уровня зависят от данных, которые предоставляют средства верхнего уровня. Они используются разработчиками приложений и помогают создать информационную систему, однако не являются полноценными инструментами разработки программного обеспечения.
- **Интегрированные CASE средства (I – CASE).** Эти CASE средства охватывают полный жизненный цикл разработки информационной системы. Они позволяют обмениваться данными между инструментами верхнего и нижнего уровня и являются своего рода «мостом» между CASE средствами верхнего и нижнего уровней.

Для моделирования и оптимизации бизнес процессов применяются CASE средства верхнего уровня и интегрированные CASE средства. Они позволяют повысить качество моделей бизнес процессов за счет автоматического контроля, дают возможность оценить ожидаемый результат, ускоряют процесс проектирования, обеспечивают возможности по изменению и обновлению моделей.

22. Этапы проектирования баз данных. Задачи инфологического, логического, физического проектирования.

2.1. Этапы проектирования базы данных

При проектировании базы данных решаются три основные проблемы.

1. Как адекватно отразить предметную область и информационные потребности пользователей в концептуальной модели? Эту проблему называют *проблемой инфологического проектирования баз данных*. Цель инфологического этапа проектирования состоит в получении семантических (смысловых) моделей, отражающих информационное содержание проблемы.

2. Каким образом представить объекты предметной области в абстрактных объектах моделей данных так, чтобы это отображение не противоречило семантике предметной области и было максимально удобным? Эта проблема известна как *проблема логического проектирования баз данных*. Цель логического этапа проектирования – организация данных, выделенных на предыдущем этапе, в форму, принятую в выбранной СУБД.

3. Как обеспечить эффективность выполнения запросов к базе данных? Каким образом для конкретной СУБД расположить данные во внешней памяти? Создание каких дополнительных структур (например, индексов) необходимо потребовать и т. д.? Эту проблему называют *проблемой физического проектирования баз данных*. Цель физического этапа – выбор рациональной структуры хранения данных и методов доступа к ним.

2.3. Логическое проектирование реляционных баз данных

Задача логического проектирования реляционной базы данных состоит в обоснованном принятии решений о том,

- из каких отношений должна состоять база данных и
- какие атрибуты должны быть у этих отношений.

Концептуальное (инфологическое) проектирование

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Термины «семантическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами. Кроме того, в этом контексте равноправно могут использоваться слова «модель базы данных» и «модель предметной области» (например, «концептуальная модель базы данных» и «концептуальная модель предметной области»), поскольку такая модель является как образом реальности, так и образом проектируемой базы данных для этой реальности.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

Чаще всего концептуальная модель базы данных включает в себя:

- описание информационных объектов или понятий предметной области и связей между ними.
- описание ограничений целостности, т.е. требований к допустимым значениям данных и к связям между ними.

Логическое (даталогическое) проектирование

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. Этот этап может быть в значительной степени автоматизирован.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Физическое проектирование

Физическое проектирование — создание схемы базы данных для конкретной СУБД. Специфика конкретной СУБД может включать в себя ограничения на именование объектов базы данных, ограничения на поддерживаемые типы данных и т.п. Кроме того, специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и т.д.

23. Инфологическое проектирование. Сущности, атрибуты, связи. Способы представления сущности. Классификация атрибутов.

Инфологическое проектирование – получение смысловых моделей, отражающих информационное содержание проблемы.

Сущность – реальный объект, информацию о котором необходимо хранить в БД.

Тип сущности – это набор, а экземпляр сущности – вещь в наборе.

Виды сущностей:

1. **Стержневая** – независимая сущность.

Пример: врач, анализ и пациент - стержневые

2. **Ассоциативная** – это сущность, формализующая связь М:М между двумя или более сущностями, или связь вида 1:1 между экземплярами.

Пример: Назначенный анализ

3. **Характеристическая** – это сущность, формализующая связь вида М:1 или 1:1 и служащая для уточнения стержневой сущности.

Пример: Вид_издания – характеристическая, т.е. уточняет стержневую сущность Книга

4. **Обозначающая** – это сущность, формализующая связь вида М:1 или 1:1, но не зависящая от обозначаемой сущности.

Пример: обозначающая сущность Собака, обозначаемая – Владелец собаки



Атрибут – поименованная характеристика сущности.

Атрибуты:

1. **Описательные** – представляют факты, внутренне присущие каждому экземпляру сущности.

Например: Кошка.Вес

2. **Указывающие** – используются для присвоения имени или обозначения экземплярам сущности

Например: Город.Название

3. **Вспомогательные** – используются для связи экземпляра одной сущности с экземпляром другого

Например: Кошка. Имя хозяина

Связь – поименованная, графически изображаемая ассоциация, устанавливаемая между сущностями.

Связь **один-к-одному (1:1)** существует, когда один экземпляр одной сущности связан с единственным экземпляром другой сущности.

Связь **один-ко-многим (1:M)** возникает, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан только с одним экземпляром первой сущности.

Связь **многие-ко-многим (M:N)** существует, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан с одним или более экземпляром первой сущности.

Правила атрибутов. Понятие безусловной, условной, биусловной, рекурсивной связи.
Фундаментальные виды связей. Формализация связи. Формализация связей 1:1, 1:M, M:N.
Стержневая, ассоциативная, характеристическая, обозначающая сущности. Композиция связей. Понятие подтипа и супертипа. Взаимноисключающие связи. Получение реляционной схемы из ER-диаграммы.

Правила атрибутов:

1. Один экземпляр сущности имеет одно единственное значение для каждого атрибута в любой момент времени.
2. Атрибут не должен содержать никакой внутренней структуры.
3. Когда сущность имеет составной идентификатор, то каждый атрибут, не являющийся частью идентификатора, представляет характеристику всей сущности, а не ее части.
4. Каждый атрибут, не являющийся частью идентификатора, представляет характеристику экземпляра, указанного идентификатором, а не характеристику некоторого другого атрибута – не идентификатора.

Связь – поименованная, графически изображаемая ассоциация, устанавливаемая между сущностями.

В **условной связи** могут существовать экземпляры сущностей, которые не принимают участия в связи.

К **безусловным связям** относятся связи вида: один к одному (1:1), один ко многим (1:M), многие ко многим (M:N).

Биусловная – связь, условная с обеих сторон.

Рекурсивная – связь, которая связывается сама с собой.

Среди бинарных связей можно выделить три фундаментальные безусловные связи, требующие участия каждого экземпляра сущности.

Связь **один-к-одному (1:1)** существует, когда один экземпляр одной сущности связан с единственным экземпляром другой сущности.

Связь **один-ко-многим (1:M)** возникает, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан только с одним экземпляром первой сущности.

Связь **многие-ко-многим (M:N)** существует, когда один экземпляр одной сущности связан с одним или более экземпляром другой сущности и каждый экземпляр второй сущности связан с одним или более экземпляром первой сущности.

Формализация связи

Все связи требуют описания. Описание должно включать:

- идентификатор связи;
- формулировку имен связи с точки зрения каждой участвующей сущности;
- вид связи (множественность и условность);
- формулировку того, как связь была formalизована.

Цель формулировки связи состоит в том, чтобы позволить установить связь экземпляра одной сущности с экземпляром другой. Это выполняется размещением вспомогательных атрибутов в соответствующих сущностях на модели. Когда это выполнено, говорят, что *связь formalизована*.

Формализация связей 1:1, 1:M, M:N

Для формализации связи *один к одному* вспомогательные атрибуты могут быть добавлены к любой сущности (но не к обеим). Для формализации связи *один ко многим* вспомогательные атрибуты должны быть добавлены к сущности на стороне *многого*, поскольку размещение такого вспомогательного атрибута на стороне *один* будет некорректным. Для формализации связи *многие ко многим* создают отдельную ассоциативную сущность, которая содержит ссылки на идентификаторы каждого из участвующих экземпляров (рис. 2.2).

..

Виды сущностей:

1. **Стержневая** – независимая сущность.

Пример: врач, анализ и пациент - стержневые

2. **Ассоциативная** – это сущность, формализующая связь M:M между двумя или более сущностями, или связь вида 1:1 между экземплярами.

Пример: Назначенный анализ

3. **Характеристическая** – это сущность, формализующая связь вида M:1 или 1:1 и служащая для уточнения стержневой сущности.

Пример: Вид_издания – характеристическая, т.е. уточняет стержневую сущность Книга

4. **Обозначающая** – это сущность, формализующая связь вида M:1 или 1:1, но не зависящая от обозначаемой сущности.

Пример: обозначающая сущность Собака, обозначаемая – Владелец собаки



Связь, образованная композицией, не может быть формализована во вспомогательных атрибутах.

Подтип – часть супертипа, созданная с целью уменьшения общего количества атрибутов каждой сущности.

Супертип – содержит общую для всех типов сущностей информацию.

Две или более связей одной и той же сущности могут оказаться взаимоисключающими (либо-либо).

Пример диаграммы из двух сущностей со взаимно исключающими связями показан на рис. 2.4. Автомобиль может находиться в рабочем состоянии, и тогда у него имеется только один водитель, или же автомобиль может находиться на ремонте на одном из нескольких возмож-

ных авторемонтных мастерских (каждое предприятие может производить ремонт нескольких автомобилей).

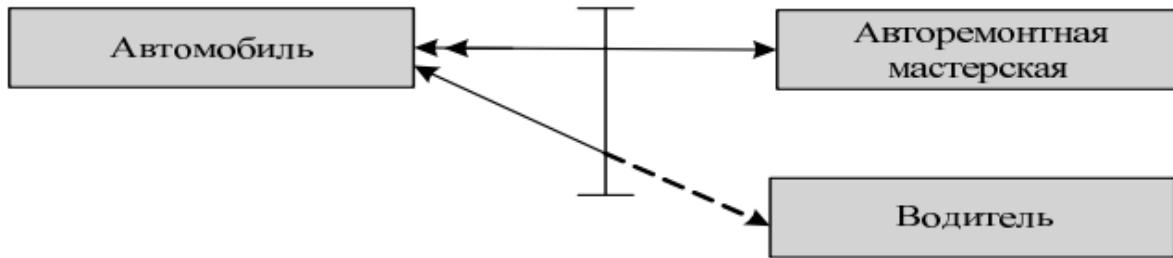


Рис. 2.4. Взаимоисключающие связи

24. **Логическое проектирование реляционных баз данных. Аномалии операций с базой данных. Общие свойства нормальных форм. Виды нормальных форм. Условия нахождения отношения в первой нормальной форме. Негативные последствия нахождение отношения лишь в первой нормальной форме. Зависимость адекватности базы данных предметной области, легкости разработки и сопровождения базы данных, скорости выполнения основных операций от степени нормализации отношений базы данных.**

Логическое проектирование – организация данных, в форму, принятую в выбранной СУБД.

Аномалии операций с базой данных

Аномалия – противоречие между моделью предметной области и моделью данных, поддерживаемой средствами конкретной СУБД.

Аномалия вставки – при добавлении данных, часть которых у нас отсутствует, мы вынуждены или не выполнять добавление или подставлять пустые или фиктивные данные.

Аномалия обновления – при обновлении данных мы вынуждены обновлять много строк и рискуем часть строк «забыть обновить».

Аномалия удаления – при удалении части данных мы теряем другую часть, которую не надо было удалять.

новления. Под аномалиями обновления понимаются трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE):

– **добавление кортежей.** Невозможно занести в отношение СТУДЕНТЫ студента, который в данное время еще не участвует ни в одном проекте (атрибут *Номер_проекта* является частью первичного ключа и не может содержать неопределенных значений);

– **удаление кортежей.** Невозможно сохранить в отношении СТУДЕНТЫ данные о студенте, завершившем участие в своем последнем проекте потому, что значение атрибута *Номер_проекта* становится неопределенным. Однако вероятны ситуации, когда между выполнением проектов у студента могут возникать перерывы;

– **модификация кортежей.** Чтобы изменить стипендию студента, необходимо модифицировать все кортежи с соответствующим значением атрибута *Id_номер*;

– **избыточность данных.** В отношении прослеживается избыточность хранения значений атрибутов *Тип_стипендии* и *Стипендия* в каждом кортеже, описывающем задание студента в некотором проекте.

Общие свойства нормальных форм:

1. Каждая следующая НФ в некотором смысле лучше предыдущей.
2. При переходе к следующей НФ свойства предыдущих сохраняются.

Отношение находится в 1НФ тогда, и только тогда, когда в любом допустимом значении этой переменной каждый кортеж отношения содержит только одно значение для каждого из атрибутов.

Свойства 1НФ:

1. В отношении нет одинаковых кортежей.
2. Кортежи не упорядочены.
3. Атрибуты не упорядочены.
4. Все значения атрибутов атомарны.

Негативные последствия нахождение отношения лишь в первой нормальной форме – возникновение аномалий (вставки, удаления, обновления).

25. Функциональные зависимости и их свойства. Замыкание множества FD. Аксиомы Амстронга. Функционально полная и частичная зависимости неключевого атрибута от составного ключа. Минимальное множество функциональных зависимостей. Декомпозиция без потерь и функциональные зависимости. Теорема Хеза. Условия нахождения отношений во второй нормальной форме.

Функциональная зависимость – когда X однозначно соответствует Y.

Детерминант – определитель. Т.е. X – детерминант Y.

Функционально полная зависимость – если атрибут Y не зависит функционально от любого точного подмножества X.

FD A->B называется **тривиальной**, если множество A включает множество B.

FD A->C называется **транзитивной**, если существует такой атрибут B, что имеются функциональные зависимости A->B и B->C, но обратная зависимость C->A отсутствует.

Замыканием множества FD S является множество FD S⁺, включающее все FD, логически выводимые из FD множества S.

Аксиомы Армстронга. А, В и С - атрибуты (в общем случае, составные) отношения г. АВ обозначает А UNION В. Тогда:

1. если $B \subseteq A$, то $A \rightarrow B$ (рефлексивность);
2. если $A \rightarrow B$, то $AC \rightarrow BC$ (пополнение);
3. если $A \rightarrow B$ и $B \rightarrow C$, то $A \rightarrow C$ (транзитивность).

Не ключевой атрибут **функционально полно** зависит от составного ключа, если он функционально зависит от ключа, но не находится в функциональной зависимости ни от какой части ключа.

Частичная зависимость - это зависимость не ключевого атрибута от части составного ключа (например, должность зависит от фамилии).

Определение 7. *Множество FD S называется минимальным* \Leftrightarrow , когда:

1. правая часть любой FD из S является множеством из одного атрибута (простым атрибутом);
2. *дeterminант* каждой FD из S обладает свойством **минимальности**; это означает, что удаление любого атрибута из *дeterminанта* приводит к изменению замыкания S^+ , т. е. порождению множества FD, не эквивалентного S;
3. удаление любой FD из S приводит к изменению S^+ , т. е. порождению множества FD, не эквивалентного S.

Определение 4. Под декомпозицией без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений. Условие, при котором декомпозиция является декомпозицией без потерь, определяется теоремой Хеза.

Теорема Хеза. Пусть задано отношение $R\{A, B, C\}$ (атрибуты А, В и С в общем случае являются составными атрибутами) и выполняется функциональная зависимость $A \rightarrow B$. Тогда справедливо

$$R = (R \text{ PROJECT } \{A, B\}) \text{ NATURAL JOIN } (R \text{ PROJECT } \{A, C\}).$$

Определение 6. Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа. Для приведения отношения в 2НФ необходимо:

- 1) построить его проекцию, исключив атрибуты, которые не находятся в полной функциональной зависимости от составного ключа;
- 2) дополнительно построить одну или несколько проекций на часть составного ключа и атрибуты, функционально зависящие от этой части.

Любое отношение, находящееся в 1 НФ, но не находящаяся в 2НФ, может быть приведено к набору отношений, находящихся в 2НФ. В результате декомпозиции получается набор проекций исходного отношения, естественное соединение которых восстанавливает исходное отношение.

26. Транзитивная зависимость. Условия нахождения отношений в третьей нормальной форме. Теорема Риссонена. Перекрывающиеся возможные ключи и нормальная форма Бойса-Кодда. Условия нахождения отношений в усиленной третьей нормальной форме.

FD $A \rightarrow C$ называется **транзитивной**, если существует такой атрибут В, что имеются функциональные зависимости $A \rightarrow B$ и $B \rightarrow C$, но обратная зависимость $C \rightarrow A$ отсутствует.

Определение 7. Отношение находится в ЗНФ только в том случае, если оно находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Любое отношение, находящееся в 2НФ, но не находящееся в ЗНФ, может быть приведено к набору отношений, находящихся в ЗНФ, естественное соединение которых воспроизводит исходное отношение.

Теорема

Теорема Риссанена от первичного ключа.

Проекции r_1 и r_2 отношения r являются **независимыми** \leftrightarrow , когда:

- 1) каждая FD в отношении r логически следует из FD в r_1 и r_2 ;
- 2) общие атрибуты r_1 и r_2 образуют возможный ключ хотя бы для одного из этих отношений.

В случае, когда у отношения имеется несколько возможных ключей и при этом некоторые из этих возможных ключей «перекрываются» (т. е. содержат общие атрибуты), могут иметь место аномалии обновления.

Определение 8. Отношение находится в нормальной форме Бойса–Кодда (НФБК) в том и только в том случае, когда детерминанты всех ее функциональных зависимостей являются потенциальными ключами.

27. Многозначные зависимости. Лемма и теорема Фейджина. Тривиальные многозначные зависимости. Условия нахождения отношений в четвертой нормальной форме.

Многозначная зависимость (обозначается MVD) B от A существует тогда и только тогда, когда множество значений атрибута B , соответствующее паре значений атрибутов A и C , зависит от значения A и не зависит от значения C .

Лемма Фейджина

Многозначные зависимости обладают свойством *двойственности* (лемма Фейджина): в отношении $R\{A, B, C\}$ выполняется многозначная зависимость $A \rightarrow\rightarrow B$ в том и только в том случае, когда выполняется многозначная зависимость $A \rightarrow\rightarrow C$. Свойство двойственности обозначается как $A \rightarrow\rightarrow B | C$.

Теорема Фейджина. Пусть имеется отношение R с атрибутами A, B, C (в общем случае – составными). Отношение R декомпозируется без потерь на проекции $\{A, B\}$ и $\{A, C\}$ тогда и только тогда, когда для него выполняется MVD $A \rightarrow\rightarrow B | C$ (когда есть две многозначные зависимости).

Теорема Фейджина служит основой для декомпозиции отношений, удаляющих аномальные многозначные зависимости, с приведением отношений в четвертую нормальную форму.

Многозначная зависимость $A \twoheadrightarrow B$ называется **тривиальной**, если выполняется хотя бы одно из условий:

- Множество A является **надмножеством** B :

$$B \subseteq A$$

- Объединение A и B образует весь заголовок отношения.

$$A \cup B = R$$

Определение 10. Отношение R находится в четвертой нормальной форме (4НФ) в том и только в том случае, если оно находится в ЗНФ (или НФБК) и при наличии единственной многозначной зависимости $A \twoheadrightarrow B$, все остальные атрибуты отношения R функционально зависят от детерминанта этой многозначной зависимости.

28. Зависимости проекции-соединения. Циклические ограничения (3D-ограничения). Условия нахождения отношений в пятой нормальной форме проекции-соединения.

Определение 11. В отношении R с произвольными атрибутами A, B, Z (в общем случае – составными) удовлетворяется **зависимость проекции/соединения** (*project-join dependency* – PJD) * (A, B, \dots, Z) тогда и только тогда, когда отношение R можно получить путем естественного соединения проекций этого отношения на атрибуты A, B, \dots, Z .

Определение 12. Отношение находится в пятой нормальной форме, или нормальной форме проекции-соединения (*project-join normal form* – 5НФ, или PJ/NF) в том и только в том случае, если в каждой нетривиальной PJD все проекции содержат возможный ключ.

В лекции по курсу есть пример приведения отношения к 5НФ через декомпозицию его на три отношения. Обнаружение всех зависимостей соединения – нетривиальная задача, для решения которой нет общих методов.

29. Ограничения целостности. Общие требования относительно ограничений целостности. Классификация ограничений целостности. Возможные подходы относительно удаления целевой сущности, на которую ссылается внешний ключ. Возможные подходы относительно обновления первичного ключа целевой сущности, на которую ссылается внешний ключ. Средства поддержания целостности информации в базе данных. Средства и способы задания ограничений целостности в языке SQL.

Ограничение целостности представляет собой некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

Примеры ограничений:

- 1) возраст сотрудника не может быть меньше 18 и больше 65 лет;
- 2) каждый студент имеет уникальный идентификационный номер;
- 3) студент обязан числиться только в одной группе;

В информационных системах, работающих с базами данных, можно выделить три уровня:

- выполнения функций сохранения и выборки данных;
- выполнения правил обработки данных;
- управления интерфейсом пользователя.

Первый уровень – это набор таблиц, а также декларативные и триггерные ограничения целостности, которые в совокупности обеспечивают хранение и непротиворечивость данных.

Третий уровень – это специально разработанное приложение, обеспечивающее корректный и комфортный для пользователя диалог с информационной системой. Такое приложение, как правило, не имеет прямого доступа к базе данных, а обращается ко второму уровню.

Второй уровень – промежуточный – представляет собой набор процедур и функций, которые инкапсулируют элементарные операции с базой данных (SQL-запросы) и обеспечивают соблюдение бизнес-правил.



[Возможные подходы относительно удаления целевой сущности, на которую ссылается внешний ключ](#)

2. Что должно случиться при попытке УДАЛЕНИЯ целевой сущности, на которую ссылается внешний ключ? Например, при удалении поставщика, который осуществил по крайней мере одну поставку. Существует три возможности:

КАСКАДИРУЕТСЯ Операция удаления "каскадируется" с тем, чтобы удалить также поставки этого поставщика.

ОГРАНИЧИВАЕТСЯ Удаляются лишь те поставщики, которые еще не осуществляли поставок. Иначе операция удаления отвергается.

УСТАНАВЛИВАЕТСЯ Для всех поставок удаляемого поставщика NULL-значение внешний ключ устанавливается в неопределенное значение, а затем этот поставщик удаляется. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

[Возможные подходы относительно обновления первичного ключа целевой сущности, на которую ссылается внешний ключ](#)

3. Что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа целевой сущности, на которую ссылается некоторый внешний ключ? Например, может быть предпринята попытка обновить номер такого поставщика, для которого имеется по крайней мере одна соответствующая поставка. Этот случай для определенности снова рассмотрим подробнее. Имеются те же три возможности, как и при удалении:

КАСКАДИРУЕТСЯ Операция обновления "каскадируется" с тем, чтобы обновить также и внешний ключ в поставках этого поставщика.

ОГРАНИЧИВАЕТСЯ Обновляются первичные ключи лишь тех поставщиков, которые еще не осуществляли поставок. Иначе операция обновления отвергается.

УСТАНАВЛИВАЕТСЯ Для всех поставок такого поставщика NULL-значение внешний ключ устанавливается в неопределенное значение, а затем обновляется первичный ключ поставщика. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

[Средства поддержания целостности информации в базе данных](#)

1. Задание недопустимости неопределенных значений столбца
2. Задание неопределенных значений столбца в качестве значений по умолчанию
3. Ограничение на значения атрибута
4. Ограничения на значения различных атрибутов таблицы
5. Задание первичного ключа

[Средства и способы задания ограничений целостности в языке SQL](#)

6. Задание первичного ключа
7. Каскадное удаление

30. Физическое проектирование. Задачи физического проектирования.

Способы управления данными.

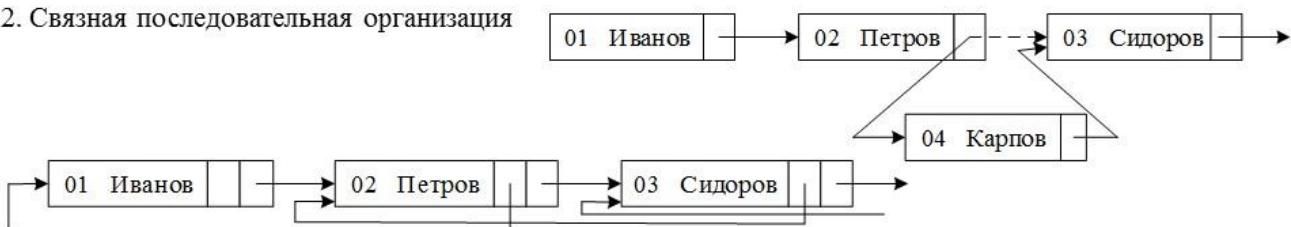
3. Физическое проектирование реляционных баз данных

Способы последовательной организации:

1. Физически последовательная организация



2. Связная последовательная организация



Методы отображения для файлов с прямым доступом:

- индексирование;
- хеширование

Механизм поиска – алгоритм, определяющий способ размещения и поиска хранимых записей



31. Индексы. Создание индекса. Правила определение набора требуемых индексов. Задача рефакторинга.

Индекс – объект БД, созданный с целью повышения производительности поиска данных. Они создаются для столбцов таблиц и представлений.

Правила индексирования:

- Число строк > 200.
- Столбцы, которые часто используются в WHERE, ORDER BY, GROUP BY, DISTINCT.
- Столбцы, используемые для соединения таблиц или являющиеся внешними ключами.
- Индексирование столбцов с высокой селективностью.
- Не следует индексировать столбцы, которые используются только в функциях.
- Не следует индексировать часто изменяемые столбцы.

Задача рефакторинга

Классификация индексов

- **По значению ключей**
 - первичный индекс;
 - вторичный индекс
- **По количеству полей в индексе**
 - простой индекс;
 - составной индекс
- **По уровням**
 - одноуровневый индекс;
 - многоуровневый индекс
- **По способу организации**
 - плотный индекс;
 - разреженный индекс;
 - индекс со структурой В-дерева;
 - индекс на основе инвертированных файлов;
 - индекс на битовых шкалах

Задача рефакторинга

Рефакторинг баз данных — это простое изменение в схеме базы данных, которое способствует улучшению её проекта при сохранении функциональной и информационной семантики.

Рефакторинг структуры

Изменения в структуре таблиц или [представлений](#).

Рефакторинг качества данных

Изменения, направленные на улучшение качества хранимой в базе данных информации.

Рефакторинг ссылочной целостности

Изменения, направленные на поддержание [ссылочной целостности](#) в базе данных.

Рефакторинг архитектуры

Изменения, направленные на улучшение взаимодействия внешних программ с базой данных.

Рефакторинг методов

Методы [рефакторинга кода](#), применимые к [триггерам](#) и [хранимым процедурам](#).

32. Организация индексов. Классификация индексов. Доступ с плотным индексом. Доступ с неплотным индексом. Решение проблемы переполнения при использовании разреженного индексного файла. Доступ посредством инвертированных файлов. Доступ посредством В-деревьев. В- и В+-деревья. Битовые индексы. Хеш-индексирование. Метод открытой адресации. Метод области переполнения. Метод многократного хеширования. Достоинства и недостатки различных методов индексирования.

Классификация индексов

- **По значению ключей**
 - первичный индекс;
 - вторичный индекс
- **По количеству полей в индексе**
 - простой индекс;
 - составной индекс
- **По уровням**
 - одноуровневый индекс;
 - многоуровневый индекс
- **По способу организации**
 - плотный индекс;
 - разреженный индекс;
 - индекс со структурой В-дерева;
 - индекс на основе инвертированных файлов;
 - индекс на битовых шкалах

Рассмотрим **файлы с плотным индексом**. В этих файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а структура индексной записи в них имеет следующий вид:

Значение ключа Номер записи

Так как индексные файлы строятся для первичных ключей, однозначно определяющих запись, то в них не может быть двух записей, имеющих одинаковые значения первичного ключа. В индексных файлах с плотным индексом для каждой записи в основной области существует одна запись из индексной области. Все записи в индексной области упорядочены по значению ключа, поэтому можно применить более эффективные способы поиска в упорядоченном пространстве.

Длина доступа к произвольной записи оценивается не в абсолютных значениях, а в количестве обращений к *устройству внешней памяти*, которым обычно является диск. Именно обращение к диску является наиболее *длительной операцией* по сравнению со всеми обработками в оперативной памяти.

Неплотный индекс строится именно для упорядоченных файлов. Для этих файлов используется принцип внутреннего упорядочения для уменьшения количества хранимых индексов. Структура записи индекса для таких файлов имеет следующий вид:

Значение ключа первой записи блока Номер блока с этой записью

В индексной области мы теперь ищем нужный блок по заданному значению первичного ключа. Так как все записи упорядочены, то значение первой записи блока позволяет нам быстро определить, в каком блоке находится искомая запись. Все остальные действия происходят в основной области.

Решение проблемы переполнения при использовании разреженного индексного файла

При вставке нового индекса может возникнуть переполнение блока индексного файла, когда резерв для новых индексов в блоке исчерпан. В этом случае необходима *перестройка индексного файла*, заключающаяся в вытеснении части индексов в последующий блок или расщеплении переполненного блока. *Расщепление* выполняется путем создания нового блока индексного файла и вставки его вслед за переполненным блоком. Часть индексов переносится из переполненного блока в новый блок индекса.

3. Инвертированный файл – файл, для которого поддерживается **плотный вторичный индекс** по значениям записей некоторого поля

Инвертированные файлы – двухуровневая структура:

1 уровень: упорядоченные значения вторичных индексов → 2 уровень: цепочка связанных блоков с номерами записей основного файла с этим значением вторичного ключа → запись с данными

В-дерево – сбалансированное, сильно ветвистое дерево. Узлы Б-дерева содержат не один элемент, а группу элементов с суммарным объемом в один блок файла, который обычно по размеру соответствует одному сектору диска.

Как и в других деревьях, данные и ключи хранятся в узлах дерева. Обычно данные представлены в виде файлового указателя на запись в файле с данными. При перемещении в В - дереве ключа вместе с ключом перемещается и указатель.

Свойства В-дерева:

- Корень является листом, либо имеет по крайней мере двух сыновей.
- Каждый узел, за исключением корня и листьев, имеет от $(m/2)$ до m сыновей.
- Все пути от корня до любого листа имеют одинаковую длину.
- Ключи в вершинах отсортированы по возрастанию.
- **ссылки на записи данных размещаются не только в листьях, но и во внутренних узлах дерева.**

B+ дерево

- +
- **Нелистовые вершины содержат только ключи и ссылки на дочерние страницы;**
 - листовые вершины содержат все множество ключей отношения, сами указатели на записи, плюс указатель на следующий по порядку лист;
 - ключи в листовых вершинах отсортированы по возрастанию.

Битовый индекс (bitmap index) — метод битовых индексов заключается в создании отдельных битовых карт (последовательность 0 и 1) для каждого возможного значения столбца, где каждому биту соответствует строка с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита, содержит индексируемое значение для данного столбца или свойства.

3.3 Хеширование (хеш-индексирование)

Хеширование – это технология быстрого доступа к нужной записи файла на основе значения некоторого поля записи, которое называется **хеш-ключом**

Метод открытой адресации состоит в том, чтобы, пользуясь каким-либо алгоритмом, обеспечивающим перебор элементов таблицы, просматривать их в поисках свободного места для новой записи.

Преимуществом хранения хеш-кода является постоянная и достаточно малая величина (например, 2 байта) его длины, которая не зависит от длины исходящего значения ключевого поля, что существенно влияет на снижение времени поисковых операций.

Недостаток хеширования состоит в необходимости выполнения операции свертки, которая занимает определенное время, а также предупреждение возникновения коллизий (результатом свертки разных значений может быть одинаковый хеш-код).

Главной причиной повышения скорости выполнения разных операций в индексированных таблицах является выполнение основной части работы с небольшими индексными файлами. Повышение производительности работы достигается с наибольшим эффектом при использовании больших таблиц.

3. Физическое проектирование реляционных баз данных

Плотный и разряженный индексы используются в простейших СУБД для небольших файлов

Индексы В-деревьев

Хороши для данных с высокой кардинальностью
Хороши для баз данных OLTP
Занимают много места
Легко обновляются

Индексы В-деревьев

Универсальны
Слабо зависят от распределения значений ключа
Большие расходы на хранение
Участвуют в журнале изменений

Битовые индексы

Хороши для данных с низкой кардинальностью
Хороши для приложений хранилищ данных OLAP
Используют, относительно мало места
Трудно обновляются

Хеш-индексы

Используются только при доступе к отдельной записи при поиске по равенству
Требуют равномерного распределения значений ключа
Малые расходы на хранение
Не участвуют в протоколе Write Ahead Log

33. Методы организации хранения данных. Способы хранения отношений.

Табличные пространства. Фрагментация. Кластеризация таблиц.

Параметры проектирования физического уровня.

Способы хранения отношений:

- по кортежное
- по столбцам

Табличное пространство – логическая единица хранения данных.

Особенности хранения кортежей таблиц:

- ✓ каждый кортеж обладает уникальным идентификатором (tid);
- ✓ каждый кортеж хранится целиком в одной странице;
- ✓ в одной странице данных хранятся кортежи только одной таблицы;
- ✓ логическая реорганизация таблицы при добавлении новых полей;
- ✓ поддержка хранения неопределенных значений;
- ✓ решение задачи распределения памяти в страницах данных в связи с проблемами синхронизации и журнализации

Фрагментация базы данных – это шаблон архитектуры базы данных, связанный с горизонтальным секционированием (это разделение строк одной таблицы на несколько различных таблиц, называемых разделами). Каждый раздел имеет одинаковую схему и столбцы, но разные строки. Соответственно, данные, хранящиеся в каждом из них, уникальны и не зависят от данных, хранящихся в других разделах.

Идея кластеризации заключается в наиболее близком физическом размещении на диске логически связанных между собой данных (например, по какому-нибудь полю, по ключу). Записи физически лежат друг за другом.

Применяется 2 подхода: внутрифайловая и межфайловая кластеризация.

Внутрифайловая – это когда записи располагаются друг за другом в рамках одной таблицы.

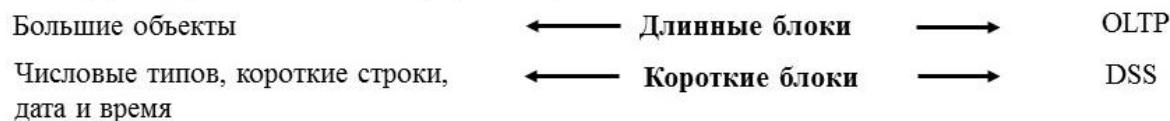
Межфайловая – друг за другом располагаются записи из связанных таблиц (читывается запись поочерёдно, то из одной, то из другой).

Экстент – непрерывная последовательность страниц (единица выделения\освобождения памяти)

III. Параметры и проблемы проектирования физического уровня

- ✓ размеры табличных пространств, экстентов, страниц для хранения таблиц, индексов...;
- ✓ размер словаря данных, включая код всех хранимых процедур, функций, триггеров;
- ✓ управляющие файлы, файлы журнала;
- ✓ интенсивность потока запросов, модифицирующие данные и индексы;
- ✓ файлы временных табличных пространств;
- ✓ интенсивность потока запросов, инициирующих создание временных таблиц;
- ✓ потоки транзакций read-write, read only, объем данных;
- ✓ характеристики параллельной работы;
- ✓ количество приложений, одновременно работающих с БД;
- ✓ количество соединений с БД для каждого приложения;
- ✓ входные и выходные данные, генерируемые пользовательскими приложениями.

a) Выбор размера блока данных (страницы)



б) Выбор размера экстента

- в) Фрагментация → Выделение свободного пространства
 → Расход свободного пространства и общее увеличение
 времени обработки данных

34. Понятие транзакции. Свойства классических транзакций. Модели транзакций. Транзакции с контрольными точками. Многозвенные транзакции. Вложенные транзакции. Многоуровневые транзакции.

Транзакция – неделимая с точки зрения воздействия на базу данных группа операторов манипулирования данными, выполняющаяся как единое целое и переводящая БД из одного целостного состояния в другое.

Свойства транзакций (свойства ACID):

Атомарность – никакая транзакция не будет зафиксирована в системе частично. Либо все, либо ничего.

Согласованность – каждая успешная транзакция по определению фиксирует только допустимые результаты.

Изолированность – параллельные транзакции не могут оказывать влияние.

Долговечность – независимо от проблем на нижних уровнях, изменения должны сохраняться после возвращения системы в работу.

Типы транзакций:

1. Плоские (классические)
2. Цепочечные
3. Вложенные

Контрольная точка выполняет в базе данных следующее:

Записывает в файл журнала запись, отмечающую начало контрольной точки.

Сохраняет данные, записанные для контрольной точки в цепи записей журнала контрольной точки. Одним из элементов данных, регистрируемых в записях контрольной точки, является номер LSN первой записи журнала, при отсутствии которой успешный откат в масштабе всей базы данных невозможен. Такой номер LSN называется минимальным номером LSN восстановления (MinLSN).

Модель **многозвенных транзакций** включает оператор CHAIN WORK - неделимую комбинацию операторов COMMIT WORK и BEGIN WORK, - которая неравноцenna последовательному выполнению операторов COMMIT WORK и BEGIN WORK по отдельности. При выполнении этих операторов по отдельности контекст пропадает; некоторая другая транзакция может "вклиниваться" и изменить значения в базе данных, которые нужны для выполнения следующего "звена" многозвенной транзакции, прежде чем это звено начнет выполняться.

Таким образом, многозвенные транзакции концептуально эквивалентны транзакциям с контрольными точками с той разницей, что откат может производиться только до последней зафиксированной точки, а не до любой предыдущей контрольной точки.

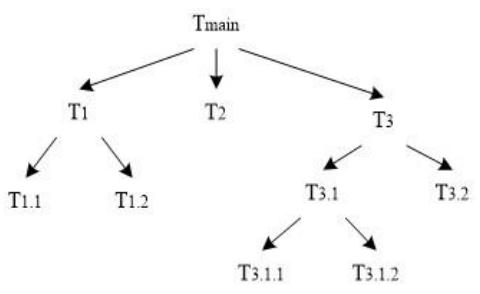
Иногда бывает так, что при обработке запроса необходимо открыть ещё одну транзакцию в рамках текущей транзакции. Это называется вложенной транзакцией.

Вложенные транзакции выглядят достаточно подозрительными в том смысле, что они могут нарушать принципы ACID:

1. Вложенная субтранзакция может нарушать принцип Durability, так как уже зафиксированные изменения могут откатиться в случае отката внешней транзакции.
2. Вложенная независимая транзакция может нарушать принципы Atomicity и Consistency при возникновении ошибок во внешней или вложенной транзакции.

5. Обработка транзакций в распределенных системах

3. Вложенные транзакции



Правила управления поведением вложенных транзакций:

Правило фиксации. Выполнение оператора Commit Work в некоторой субтранзакции делает ее результаты видимыми только для родительской транзакции.

Правило отката. Если субтранзакция некоторого уровня вложенности, включая корневую транзакцию, подвергается откату, то же самое должно быть сделано для всех ее подтранзакций, независимо от того, производилась ли для любой из них фиксация на локальном уровне.

Правило видимости. Все изменения, осуществляемые в рамках субтранзакции, становятся видимыми ее родительской транзакции после того, как для нее будет произведена фиксация.

5. Обработка транзакций в распределенных системах

4. Многоуровневые транзакции – вложенные транзакции, в которых субтранзакция T_1 предварительно фиксирует результаты своей работы

5.2 Методы управления транзакциями в распределенных базах данных

Протоколы механизма блокировок:

- ✓ централизованный протокол двухфазной блокировки;
- ✓ блокировка первичной копии;
- ✓ распределенный протокол двухфазной блокировки.



Альтернативные алгоритмы:

- алгоритмы, основанные на временных метках,
- а также протоколы оптимистического управления одновременным доступом.

35. Транзакции и параллелизм. Три проблемы, связанные с параллелизмом. Эффекты параллелизма. Управление транзакциями. СерIALIZАЦИЯ транзакций. Понятие смеси и графика транзакций. Виды конфликтов между транзакциями. Изолированность пользователей. Уровни изолированности.

Параллелизм – механизм, с помощью которого СУБД предотвращает взаимное влияние операций, одновременно выполняемых над одними и теми же данными равными пользователями.

2.1. Проблемы, связанные с параллелизмом

- - проблема утраченного обновления;
- - проблема зависимости от незафиксированного обновления;
- - проблема анализа на непротиворечивость.

Эффекты параллелизма

Эффект фиктивных элементов - здесь за один шаг выполняется достаточно много операций - чтение одновременно нескольких строк, удовлетворяющих некоторому условию.

Транзакция А дважды выполняет выборку строк с одним и тем же условием. Между выборками вклинивается транзакция В, которая добавляет новую строку, удовлетворяющую условию отбора.

Транзакция А ничего не знает о существовании транзакции В, и, т.к. сама она не меняет ничего в базе данных, то ожидает, что после повторного отбора будут отобраны те же самые строки.

Результат. Транзакция А в двух одинаковых выборках строк получила разные результаты.

Эффект несовместимого анализа - присутствуют две транзакции - одна длинная, другая короткая.

Длинная транзакция выполняет некоторый анализ по всей таблице, например, подсчитывает общую сумму денег на счетах клиентов банка для главного бухгалтера. Пусть на всех счетах находятся одинаковые суммы, например, по \$100. Короткая транзакция в этот момент выполняет перевод \$50 с одного счета на другой так, что общая сумма по всем счетам не меняется.

Результат. Хотя транзакция В все сделала правильно - деньги переведены без потери, но в результате транзакция А подсчитала неверную общую сумму.

Т.к. транзакции по переводу денег идут обычно непрерывно, то в данной ситуации следует ожидать, что главный бухгалтер *никогда* не узнает, сколько же денег в банке.

3. Управление транзакциями в СУБД

Определение 1. Набор из нескольких транзакций, элементарные операции которых чередуются друг с другом, называется **смесью транзакций**

$$T = (T_1, T_2, \dots, T_n)$$

$$S = (S_1, S_2, \dots, S_m)$$

$$\rightarrow T = (T_1, Q_1, T_2, S_1, T_3, S_2, \dots)$$

$$Q = (Q_1, Q_2, \dots, Q_k)$$

Определение 2. Последовательность, в которой выполняются элементарные операции заданного набора транзакций, называется **графиком запуска набора транзакций**

Дисциплина сериализация транзакций:

1. В процессе выполнения транзакции пользователь видит только согласованные состояния базы данных. Пользователь никогда не может получить доступ к незафиксированным изменениям в данных, достигнутым в результате действий другого пользователя.
2. Если две транзакции, А и В, выполняются параллельно, то СУБД полагает, что результат должен быть таким же, как если бы:
 - транзакция А выполнялась первой, а за ней была бы выполнена транзакция В;
 - транзакция В выполнялась первой, а за ней была бы выполнена транзакция А.

Определение 3. Сериализация транзакций – это механизм их выполнения по некоторому **сериальному плану**.

Определение 4. План (способ) выполнения набора транзакций называется **сериальным**, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций

Определение 5. Транзакции называются **конкурирующими**, если они пересекаются по времени и обращаются к одним и тем же данным.

Виды конфликтов между транзакциями:

- **W-W** (попытка транзакции В изменить объект, измененный не закончившейся транзакцией А);
- **R-W** (попытка транзакции В изменять объект, прочитанный не закончившейся транзакцией А);
- **W-R** (попытка транзакции В прочитать объект, измененный не закончившейся транзакцией А)

Уровень изоляции транзакций - степень обеспечиваемой внутренними механизмами СУБД защиты от всех или некоторых видов вышеперечисленных несогласованности данных, возникающих при параллельном выполнении транзакций.

Грязное чтение. При *грязном чтении* другая транзакция может читать записи, измененные внутри данной транзакции. Поскольку данные, изменяемые внутри транзакции, могут быть откатаны к своему исходному состоянию, чтение такого промежуточного состояния из другой транзакции трактуется как “грязное”, то есть данные не были зафиксированы. Этого можно избежать блокировкой изменяемых записей.

Невоспроизводимое чтение. *Невоспроизводимое чтение* случается, когда данные читаются внутри транзакции, и пока эта транзакция выполняется, другая транзакция изменяет те же самые записи. Если запись читается повторно внутри транзакции, то вы получаете отличающийся результат — невоспроизводимый. Это можно предотвратить посредством блокировки чтения записей.

Фантомное чтение. *Фантомное чтение* случается при чтении диапазона данных, например, с использованием конструкции WHERE. Другая транзакция может добавить новую запись, которая попадает в диапазон подлежащих чтению в транзакции. Новая запись с тем же выражением WHERE возвратит другое количество строк. Фантомное чтение может стать серьезной проблемой при выполнении оператора UPDATE для диапазона записей.

36. Методы управления транзакциями. Метод синхронизационных захватов. Метод гранулированных синхронизационных захватов. Решение на основе аппарата синхронизационных захватов проблем, связанных с параллелизмом. Распознавание тупиковых ситуаций. Разрушение тупиков. Метод временных меток. Метод выделения версий данных.

3.1. Синхронизационные захваты

S (Shared) - совместный режим
X (eXclusive) - монопольный режим

1. Захват транзакцией А объекта R в монопольном режиме → переход транзакции В в состояние ожидания до снятия блокировки транзакцией А.
2. Захват транзакцией А объекта R в совместном режиме → то
 - запрос из транзакции В на захват объекта R в монопольном режиме → переход транзакции В в состояние ожидания до снятия блокировки транзакцией А;
 - запрос из транзакции В на захват объекта R в совместном режиме → запрос удовлетворен.

Тип захвата, установленный транзакцией А			
	X	S	-
X	n	n	y
S	n	y	y
-	y	y	y

Запрос на захват, выданный транзакцией В

Решение проблем, связанных с параллелизмом, средствами синхронизационных захватов

I. Проблема утраченного обновления

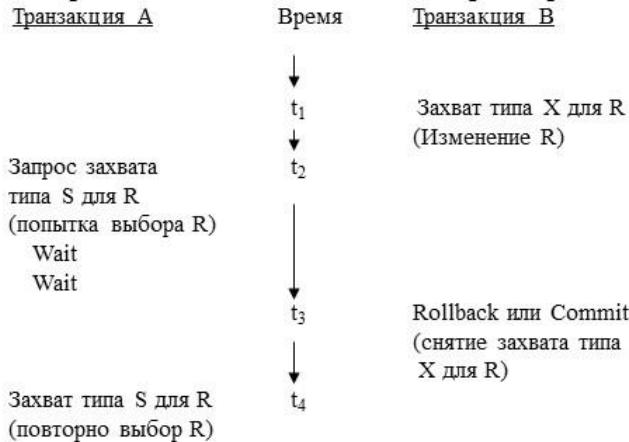


2PL - двухфазный протокол синхронизационных захватов:

- первая фаза транзакции - накопление захватов;
- вторая фаза (фиксация или откат) - освобождение захватов.

Решение проблем, связанных с параллелизмом, средствами синхронизационных захватов

II. Проблема зависимости от незафиксированных обновлений

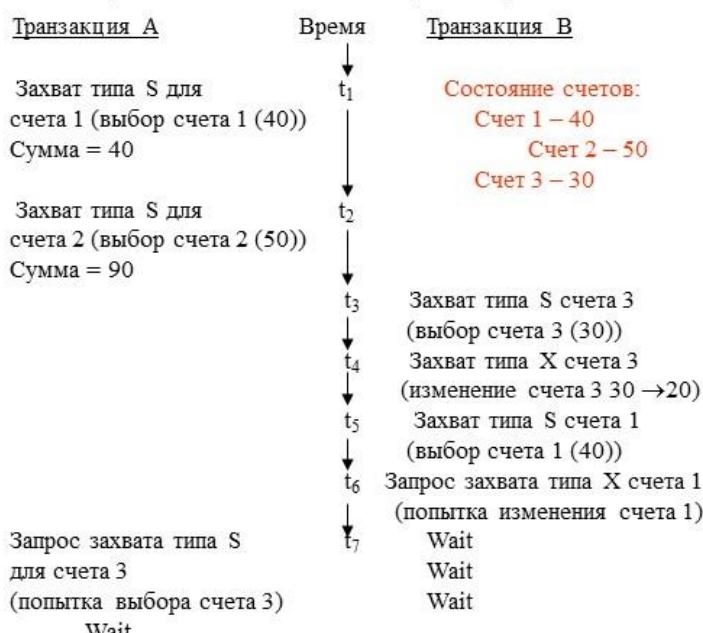


III.2. Проблема появления фиктивных элементов (фантомов)



Решение проблем, связанных с параллелизмом, средствами синхронизационных захватов

III.3. Проблема анализа на противоречивость



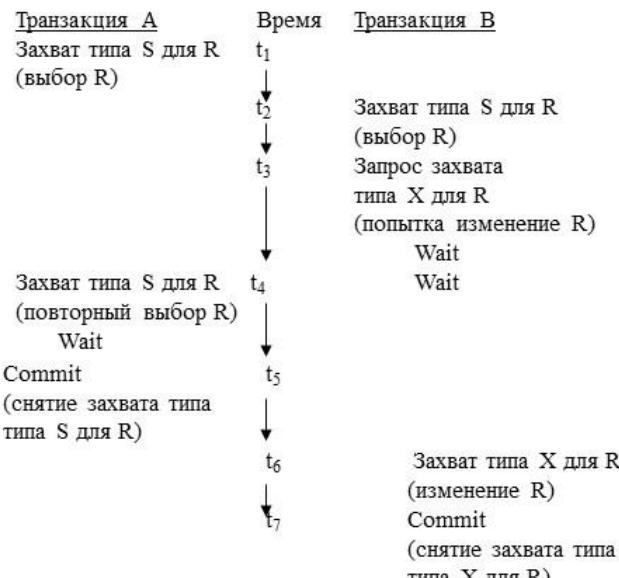
3.2. Гранулированные синхронизационные захваты

IS (Intended for Shared lock) - намерение захватить некоторый входящий в О объект в совместном режиме

IX (Intended for eXclusive lock) - намерение захватить некоторый входящий в О объект в монопольном режиме

SIX (Shared, Intended for eXclusive lock) -совместный захват всего этого объекта с намерением впоследствии захватывать какие-либо входящие в него объекты в монопольном режиме

III.1. Проблема повторяемого считывания



Основой для обнаружения тупиков является построение и постоянное поддержание ориентированного графа ожидания транзакции. Стандартный метод нахождения тупиков – метод редукции графа.

3.3. Тупики, их распознавание и разрушение

Граф ожидания транзакций – это ориентированный двудольный граф с двумя типами вершин – *вершины, соответствующие транзакциям*, и *вершины, соответствующие объектам захвата*.

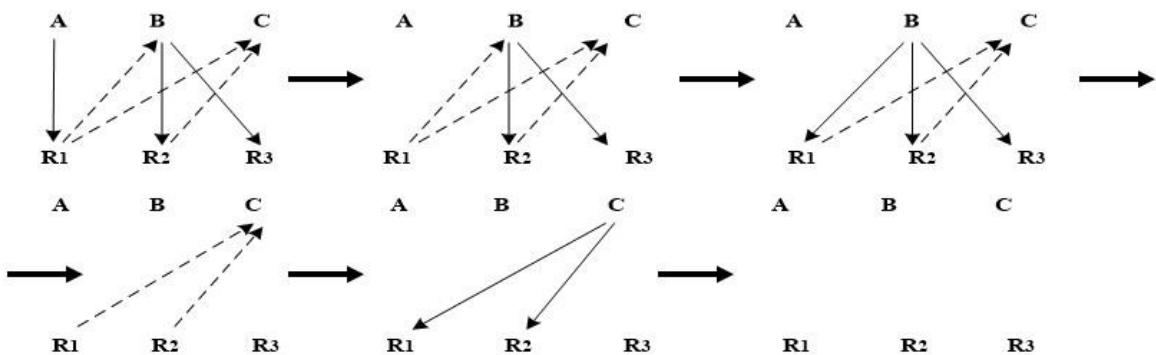
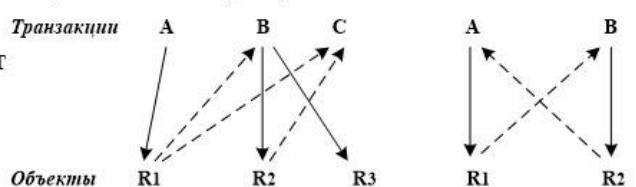
Дуга, ведущая из вершины-транзакции к вершине-объекту, говорит, что для этой транзакции существует удовлетворенный захват объекта.

Дуга, ведущая из вершины-объекта к вершине-транзакции, говорит, что транзакция ожидает удовлетворения захвата объекта

Алгоритм редукции графа:

1. Из графа ожидания удаляются все дуги, исходящие из вершин-транзакций, в которые не входят дуги из вершин-объектов (моделирование ситуации успешного завершения транзакций, не ожидающих удовлетворения захватов).
2. Для тех вершин-объектов, для которых не осталось входящих дуг, но существуют исходящие, ориентация одной из исходящих дуг изменяется на противоположную (моделирование удовлетворение захватов).

Если в графе в конце процедуры остались дуги, то в исходном графе имел место цикл.



3.4. Метод временных меток

t - временная метка, соответствующая времени начала транзакции T .

Алгоритм метода временных меток перед выполнением транзакцией T_2 операции над объектом R :

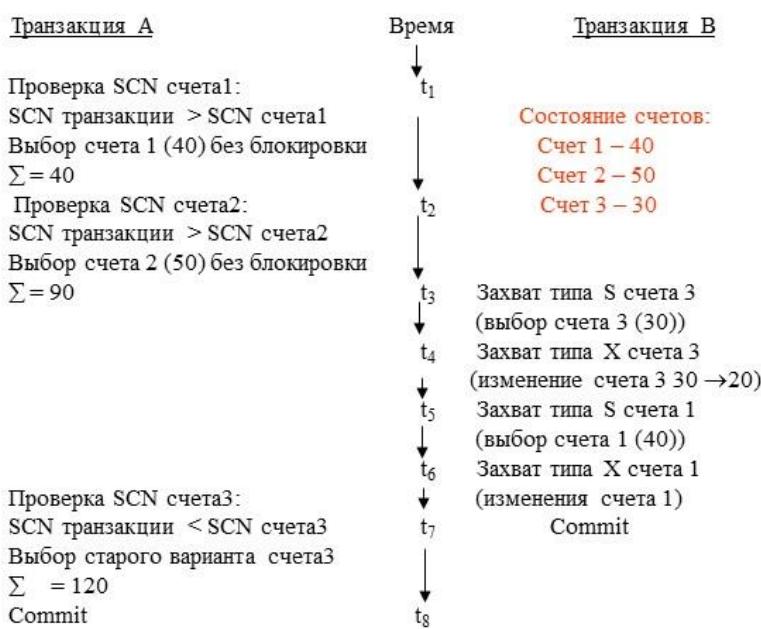
1. Проверяется завершенность транзакции T_1 . Если T_1 закончилась \rightarrow транзакция T_2 помечает объект R и выполняет свою операцию.
2. Если транзакция T_1 не завершилась \rightarrow проверяется конфликтность операций. Если операции неконфликтны \rightarrow при объекте R остается или проставляется временная метка с меньшим значением, и транзакция T_2 выполняет свою операцию.
3. Если операции T_1 и T_2 конфликтуют, \rightarrow
 - если $t(T_1) > t(T_2)$ \rightarrow производится откат T_1 и T_2 продолжает работу.
 - если же $t(T_1) < t(T_2)$ \rightarrow то T_2 получает новую временную метку и начинается заново.

3.5. Метод выделения версий данных

Алгоритм метода выделения версий данных:

1. Для каждой транзакции запоминается текущий системный номер (SCN - System Current Number). Чем позже начата транзакция, тем больше ее SCN.
2. При записи страницы данных SCN транзакции, производящей эту запись, становится SCN страницы данных.
3. Транзакции, только читающие данные не блокируют ничего в базе данных.
4. Если транзакция A читает страницу данных, то SCN транзакции A сравнивается с SCN читаемой страницы данных.
 - если SCN страницы данных $<$ SCN транзакции A , то транзакция A читает эту страницу.
 - если SCN страницы данных $>$ SCN транзакции A \rightarrow транзакция A просматривает журнал транзакций назад в поиске первой записи об изменении нужной страницы данных с SCN меньшим, чем SCN транзакции A и использует старый вариант данных страницы.

3.3 Метод выделения версий данных



4. Изолированность пользователей при работе с СУБД

I уровень – уровень “грязного чтения” (**Dirty Read – Read Uncommitted** в стандарте SQL)

\rightarrow Запрет на изменение объекта R до завершения его изменения другой транзакцией

II уровень – уровень подтвержденного чтения (**Committed Read**)

\rightarrow Запрет на чтение объекта R до завершения его изменения другой транзакцией

III уровень – уровень повторяемого чтения (**Repeatable Read**)

\rightarrow Запрет на изменение объекта R до завершения его чтения другой транзакцией

IV уровень – “phantomnyy” уровень (**Serializable**)

Set transaction isolation level

```
{read uncommitted/  
committed read/  
repeatable read/  
serializable}  
{read write/read only}
```

Уровень изоляции	Неаккуратное считывание	Повторяемое считывание	Фантомы
Read Uncommitted	Да	Да	Да
Committed Read	Нет	Да	Да
Repeatable Read	Нет	Нет	Да
Serializable	Нет	Нет	Нет

37. Технология "клиент-сервер". Преимущества модели "клиент-сервер" в сравнении с традиционной моделью обработки данных. Логические компоненты приложений. Три модели архитектуры "клиент-сервер", их достоинства и недостатки. Трехзвенная архитектура модели "клиент-сервер". Сервер приложений. Традиционный подход к работе с сервером.

Клиент-сервер – модель взаимодействия компьютеров в сети, при которой программа-клиент выполняет первичную и конечную обработку данных, программа-сервер управляет хранением и выборкой данных.

Преимущества:

- снижение сетевого трафика при выполнении запросов.
- возможность хранения бизнес-правил на сервере, что позволяет избежать дублирования кода в различных приложениях, использующих общую базу данных.
- многочисленные средства управления пользовательскими привилегиями и правами доступа к различным объектам базы данных.

2. Три модели “Клиент-Сервер”

Группы функций стандартного приложения:

- функции ввода и отображения данных;
- чисто прикладные функции;
- функции хранения и управления данными.

Логические компоненты:

- компонент представления (presentation);
- прикладной компонент (business application);
- компонент доступа к информационным ресурсам (resource manager).

Три модели “Клиент-Сервер”

- модель доступа к удаленным данным (Remote Data Access – RDA);
- модель сервера базы данных (DataBase Server – DBS);
- модель сервера приложений (Application Server – AS).

RDA

Плюсы:

- унификация и широкий выбор средств разработки приложений.

Минусы:

RDA-модель имеет ряд ограничений.

- взаимодействие клиента и сервера посредством SQL-запросов существенно загружает сеть.
- удовлетворительное администрирование приложений в RDA-модели практически невозможно.

DBS

Плюсы:

- возможность централизованного администрирования бизнес-функций
- снижение трафика сети
- возможность разделения процедуры между несколькими приложениями
- экономия ресурсов компьютера за счет использования единожды созданного плана выполнения процедуры

Минусы:

- не является мобильной относительно СУБД.
- в большинстве СУБД отсутствуют возможности отладки и тестирования хранимых процедур
- не обеспечивает требуемой эффективности использования вычислительных ресурсов
- децентрализация приложений требует существенного разнообразия вариантов взаимодействия клиента и сервера

AS

Плюсы:

- повышение производительности за счет разгрузки сервера;
- удешевление эксплуатации ИС;
- возможность усложнения бизнес-логики без потери производительности;
- улучшение свойств переносимости и масштабируемости ИС
- клиентское ПО не нуждается в администрировании;
- высокая безопасность;
- высокая надежность;

Минусы:

- растет сложность серверной части и, как следствие, затраты на администрирование и обслуживание;
- более высокая сложность создания приложений;
- сложнее в разворачивании и администрировании;
- высокие требования к производительности серверов приложений и сервера базы данных, а, значит, и высокая стоимость серверного оборудования;
- высокие требования к скорости канала (сети) между сервером базы данных и серверами приложений.

Как правило, третьим звеном в **трехзвенной архитектуре** становится сервер приложений, т.е. компоненты распределяются следующим образом:

- Представление данных — на стороне клиента.
- Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО).
- Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.

Сервер приложений — это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (интерфейс прикладного программирования), определённый самой платформой.

38. Понятие активного сервера. Задачи активного сервера. Ограничения и утверждения.

Процедуры и функции. Перегружаемые функции. Сигнатура. Хранимые процедуры. Средства организации хранимых процедур (язык SPL). Особенности написания хранимых процедур и функций в различных СУБД (Informix, Oracle, PostgreSQL). Триггеры. DDL- и DML-триггеры. Средства написания триггеров. Особенности написания триггеров в различных СУБД (Informix, Oracle, PostgreSQL). Механизм событий. Средства механизма событий. Активные базы данных и модели транзакций.

Сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

3. Активный сервер

Задачи активного сервера:

- I. Необходимо, чтобы база данных в любой момент времени правильно отражала состояние предметной области и данные были непротиворечивы.
- II. База данных должна отражать некоторые правила предметной области, по которым она функционирует (business rules).
- III. Необходим постоянный контроль за состоянием базы данных, отслеживание всех состояний и адекватная реакция на них.
- IV. Необходимо, чтобы возникновение некоторой ситуации в базе данных оперативно влияло на ход выполнения прикладной программы.

3.2 Современные решения

Объекты базы данных: ограничения и утверждения; процедуры (функции) базы данных (хранимые процедуры); триггеры (правила); события в базе данных.

Ограничения и утверждения

Create assertion Max_query
Check ((Select sum(Cost) from Cost_in_table) ←
+ (Select sum(Cost) from Cost_out_table)) < 50000) Контролируется сумма заказов из двух
различных таблиц в общем случае с
разными атрибутами (< 50000)

Хранимые процедуры представляют собой набор команд SQL, которые могут компилироваться и храниться на сервере. Таким образом, вместо того, чтобы хранить часто используемый запрос, клиенты могут ссылаться на соответствующую хранимую процедуру.

Триггер представляет собой хранимую процедуру, которая активизируется при наступлении определенного события.

```

Create procedure Назначение(номер_сотрудника integer not null, имя_сотрудника char(20)) returning int;
Define result int;
Insert into таблица_резерва Select * from таблица_сотрудников
    where номер=номер_сотрудника or имя=имя_сотрудника;
Let result=(Select Count(*) from таблица_сотрудников
    where номер=номер_сотрудника or имя=имя_сотрудника);
If result # 0
    Call Print_certificate (номер_сотрудника, имя_сотрудника);
endif
Return result;
End procedure;

```

Основы языка SPL/SQL

SPL (Stored Procedure Language) – язык написания хранимых процедур

а) Описание и вызов процедуры:

Create procedure Имя (параметр тип, ..., параметр тип) [returning тип]

Execute procedure Имя (параметр, ..., параметр)

Call get_balance (customer_id, pay) [returning balance];

параметры тип возвращаемого значения

← вызов из программы

← вызов из процедуры

г) Оператор ветвления

*If условие1 then действие1
[elif условие2 then действие2]
[else действие3]
end if*

Пример If count > 50 then ...
elif count > 25 then ...
else ...
endif

Count > 50
25 < Count > 50
Count < 25

б) Создание локальных переменных внутри процедуры

Примеры Define count int;

Define global balance int;
Define global operator char(8) default USER;
Define global topic char(5) default "SALE";
Define mail_id LIKE customer.email_address;

в) Присвоение значений

Let переменная ={выражение}

Примеры Let count = (Select count(*) from costomer_table); д) выполнение циклов с условием
Let count = get_customers(); *While ... end while*

Основы языка SPL/SQL

е) выполнение циклов с указанным числом итераций

For ... end for

Диапазон задается

- фразой **in**

for input_value in ('Yes', 'y', 'No', 'n');

- перечислением **переменная = значение1, значение2, ..., значениеN**

- заданием начального, конечного значения и шага

переменная = значение1 to значение2 step значение3

Пример for count=1 to 100 step 5
 select balance from customer
 where customer_id = count;
end for

ж) переход к следующей итерации цикла

(**continue**), выход из цикла (**exit**)

з) возвращение значения процедуры (**Return**)

и) указание, какие действия нужно

препринять в случае ошибки (**On exception**)

```

Create procedure test (...) returning int, int, char(100);
Define code, isam integer;
Define errm char(100);
On exception in (-268, -530, -206, -690, -691)
    Set code, isam, errm;
    Return code, isam, errm;
End exception;
.....
Return 0, 0, "";
End Procedure

```

Пример Процедура устанавливает скидку в 10% на продукты с номерами от 1 до 100. Каждый продукт, за исключением 50, добавляется в таблицу sale_table. В случае накопления 200 продуктов в таблице sale_table, процедура завершается.

Create procedure sale()

```

Define count int;
For count in (1 to 100)
    if count = 50
        continue for;
    end if
    Update inventory_table
        set discount = 10 where product_id = count;
    Insert into sale_table
        Select product_id, name, price, discount
            where product_id = count;
    if select count(*) from inventory_table >= 200
        exit for;
    end if
end for
End procedure

```

Триггеры баз данных

Триггеры баз данных

→ DML – Data Manipulation Lang (Insert, Delete, Update)
→ DDL – Data Definition Lang (Create/Alter/Drop table, Connect)

I. Имя триггера *Create trigger Имя_триггера*

II. Условие включения триггера

Create trigger Имя_триггера **insert on** Имя_таблицы

Create trigger Имя_триггера **delete on** Имя_таблицы

Create trigger Имя_триггера **update on** Имя_таблицы

Create trigger Имя_триггера **update of** Имя_столбца **on** Имя_таблицы

{ *after . . . / before . . . / for each row . . .* }

III. Момент срабатывания триггера

Пример Delete from customer where balance=0

[*Before действие*] [*For each row действие*] [*After действие*]

IV.

Примеры

Create trigger test1

 Delete on **customer_table**

 Referencing old as original

 For each row . . .

Create trigger test2

 Update on **customer_table**

 Referencing old as original new as newversion

 For each row . . .

V. Дополнительное условие (**When**)

Пример

Create trigger test3

 Update of **balance** on **customer_table**

 Referencing new as Now

 For each row when (Now.balance < 1000)

Referencing new as имя

Referencing old as имя

Referencing old as имя1 new as имя2

Примеры

Create trigger test4

 Delete on **customer_table**

 Referencing old as Original

 For each row when (Original.balance != 0)

 (Insert into **notpaid**

 Value (Original.customer_id, Original.balance));

Create trigger test5

 Update of **balance** on **customer_table**

 Before (when (select count(balance) from customer_table) = 0)

 (Execute procedure track_balance());

 After (Execute procedure adjust_worth());

VI. Выполняемое действие (**Insert; Delete; Update; Execute procedure**)

Механизм событий позволяет прикладной программе и серверу БД уведомить другие программы о некотором событии и тем самым синхронизировать их выполнение.

Create dbevent имя события.

Вызов события raise dbevent имя события.

Программа регистрируется как получатель события register dbevent. get dbevent имя процедуры(Имя события=Dbeventname, ее параметры). В этой процедуре проверяется название события. Если оно известно, то что-то делается.

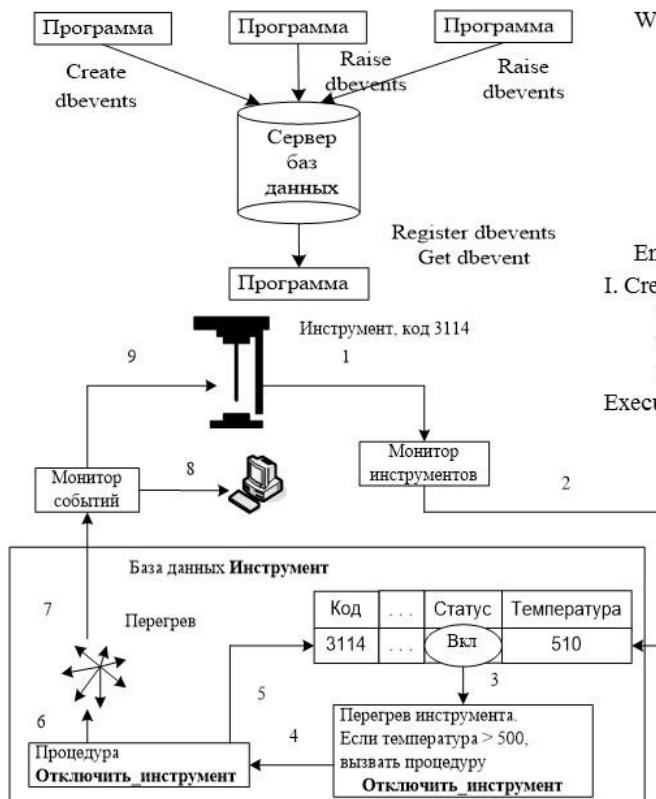
События в базе данных (DataBase events)

I. *Create dbevents* - создать событие

III. *Raisedbevent* - вызвать событие.

II. *Registerdbevent* - зарегистрировать событие

Getdbevent inquire_sql - получить (имя) событие.



```

While event_name # ""
  Getdbevent inquire_sql event_name=DBEVENTNAME
  If (event_name = "Event_1")
    Then Обработка события Event_1
  Else If (event_name = "Event_1")
    Обработка события Event_2
  Else
    .....
  End If
End While

```

I. Create trigger **Перегрев_инструмента**

Update of Температура on **Инструмент**
Referencing new as newt

For each row when (Newt.Температура >= 500)

Execute procedure **Отключить_инструмент** (Newt.Номер);

II. Create procedure **Отключить_инструмент**

(Номер_инструмента)

Update **Инструмент** set Статус = "Выкл"
where Номер = Номер_инструмента;

Raise Dbevent **Перегрев**;

End procedure

III. Create dbevents **Перегрев**

IV. Register Dbevent **Перегрев**;

...

Get Dbevent inquire_sql (Имя_события=Dbeventname, .

If (Имя_события = "Перегрев")

Then Послать сообщение; Отключить инструмент;

Endif

Процедуры и функции

Подпрограммы, определенные пользователем

Процедуры (Create procedure)

```
Create table product ( product_id int,
                      description char(20),
                      price money,
                      discount decimal(4,2))
```

```
Insert into product values (111, "Spanner",10, 2.5)
```

Результат: Product_id Description Cost
111 Spanner \$120

Сигнатура:

- тип подпрограммы (процедура или функция);
- имя подпрограммы;
- количество параметров;
- типы данных параметров;
- порядок следования параметров

```
Create function calc_cost(price money, quantity int)
  returning money;
  return (price*quantity);
End function/*функция вычисления стоимости*/
```

```
return (price*quantity- price*quantity*discount/100);
End function/*функция вычисления стоимости*/
```

```
Select product_id, description, calc_cost(price, 12) cost from product
```

Результат: Product_id Description Cost
111 Spanner \$117

Функции (Create function)

```
Create function calc_cost(price money, quantity int)
  returning money;
  return (price*quantity);
End function/*функция вычисления стоимости*/
```

```
Select product_id, description, calc_cost(price, 12) cost
  from product
```

→

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

↓

Модели транзакций и активные базы данных

1. Непосредственные (Immediate)

Tверх

Некоторые действия

Некоторые действия

Активное правило – приостановить Тверх

T₁

Обработка правил

.....

2. Отложенные (Deferred)

Tверх

Некоторые действия

Некоторые действия

Активное правило – зафиксировать

T₁

Обработка правил

.....

3. Разъединенные (Decoupled)

Тверх1

Некоторые действия

Некоторые действия

Активное правило

Некоторое действие

.....

Тверх2

Обработка правил

.....

4. Причинные (Causality)

Тверх1

Некоторые действия

Некоторые действия

Активное правило

Некорректное действие

Аварийное завершение

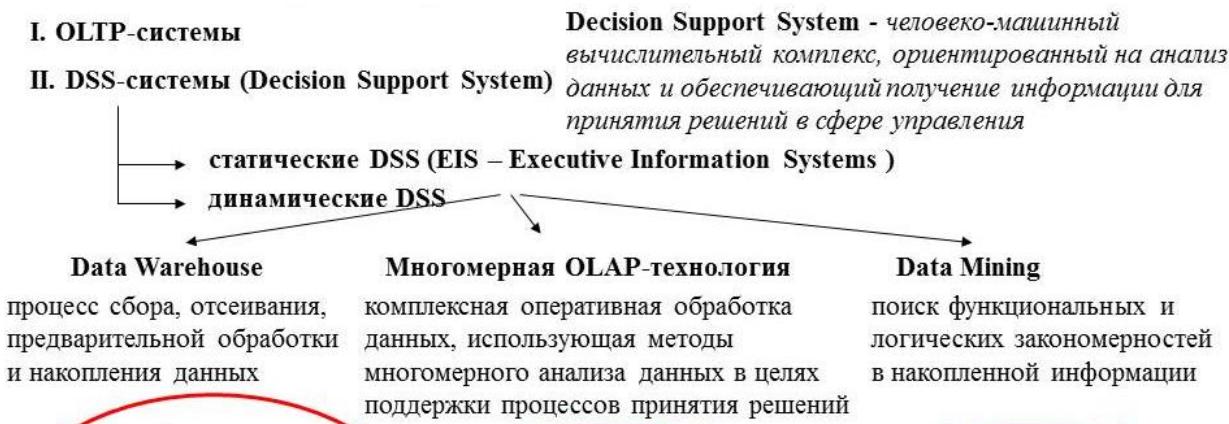
Тверх2

Обработка правил

.....

Аварийное завершение

39. Классификация информационных систем. Характеристики OLTP-систем, Характеристики DSS-систем. Концепции хранилища данных, OLAP-анализа, DataMining. Классификация аналитических систем (DSS-систем) по функциональности, по архитектуре.



OLTP-системы (OnLine Transaction Processing) – системы оперативной обработки транзакций. Основная функция подобных систем заключается в одновременном выполнении большого числа коротких транзакций от большого числа пользователей.

Системы OLTP характеризуются:

- поддержкой большого числа пользователей;
- малым временем отклика на запрос;
- относительно короткими запросами;
- участием в запросах небольшого числа таблиц.

DSS

DSS-системы оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми из электронных таблиц или из других источников данных, и характеризуются следующими признаками:

- использование больших объемов данных;
- добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложений);
- данные, добавленные в систему, обычно никогда не удаляются;
- перед загрузкой данные проходят различные процедуры «очистки», связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления, данные могут быть некорректны, ошибочны;
- небольшое число пользователей;
- очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса (интерактивность);
- скорость выполнения запросов важна, но не критична.

Аналитические системы, ориентированные на аналитика, можно разделить:

- на статические DSS, известные в литературе как информационные системы руководителя (Executive Information Systems – EIS);
- динамические DSS.

EIS-системы содержат в себе предопределенные множества запросов (примером могут служить программные продукты финансового характера компании 1С).

Вторая группа (динамические DSS), напротив, ориентированы на обработку нерегламентированных запросов аналитиков к данным. Постепенно в этом направлении оформился ряд концепций хранения и анализа корпоративных данных:

- концепция хранилища данных (Data Warehouse);
- оперативная аналитическая обработка OLAP (OnLine Analytical Processing);
- интеллектуальный анализ данных (Data Mining).

Концепция хранилища данных

Хранилище данных (Bill Inmon) – предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений

Базовые требования к хранилищу данных:

- ориентация на предметную область.
- интегрированность и внутренняя непротиворечивость.
- привязка ко времени.
- неизменяемость.
- поддержка высокой скорости получения данных из хранилища.
- возможность получения и сравнения так называемых срезов данных (*slice and dice*);
- полнота и достоверность хранимых данных;
- поддержка качественного процесса пополнения данных.

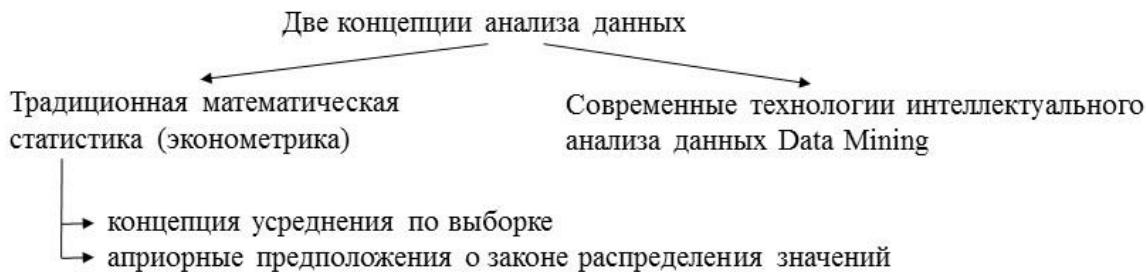
OLAP-технология – это технология оперативной аналитической обработки данных, использующая методы и средства сбора, хранения и анализа многомерных данных в целях поддержки процессов принятия решений.

Расширения языка SQL для OLAP

Особенности обработки данных в ходе OLAP-анализа:

- сортировка данных;
- хронологические последовательности;
- сравнение с итоговыми данными

Интеллектуальный анализ данных (Data Mining)



Интеллектуальный анализ данных (Data Mining) – это процесс поддержки принятия решений, основанный на поиске в данных скрытых знаний, которые ранее не были известны, нетривиальны, практически полезны, доступны для интерпретации человеком (Г.Пиатецкий-Шapiro).

Интеллектуальный анализ данных (Data Mining)

Технологии Data Mining:

- машинное обучение;
- визуализация

I. Визуализация

Перегонка данных – способ преобразования данных и представления в виде, способствующем лучшему восприятию их человеком

Интеллектуальный анализ данных (Data Mining)

II. Машинное обучение

Формы представления знаний	Задачи Data Mining	
Правила	Классификация	
Деревья решений	Регрессия	
Кластеры	Прогнозирование	
Математические функции (уравнения регрессии, временные ряды и пр.).	Прогнозирование временных рядов Кластеризация Поиск ассоциативных правил (последовательностей)	<p>Обучение с учителем</p> <p>Обучение без учителя</p>

40. Принципы построения хранилищ данных. Требования к хранилищу данных.

Архитектура хранилища данных. Основные его компоненты. Категории данных хранилища данных. Метаданные. ETL-процесс. Процедуры этапа преобразования данных в хранилище данных. Проблемы очистки данных. Подходы и методы решения проблем очистки данных. Процедуры ITL. Подходы к построению хранилищ данных.

Принципы хранилищ данных:

- **Проблемно-предметная ориентация.** Данные объединяются в категории и хранятся в соответствии с областями, которые они описывают, а не с приложениями, которые они используют.
- **Интегрированность.** Данные объединены так, чтобы они удовлетворяли всем требованиям предприятия в целом, а не единственной функции бизнеса.
- **Некорректируемость.** Данные в хранилище данных не создаются: то есть поступают из внешних источников, не корректируются и не удаляются.
- **Зависимость от времени.** Данные в хранилище точны и корректны только в том случае, когда они привязаны к некоторому промежутку или моменту времени.

Базовые требования к хранилищу данных:

- ориентация на предметную область.
- интегрированность и внутренняя непротиворечивость.
- привязка ко времени.
- неизменяемость.
- поддержка высокой скорости получения данных из хранилища.
- возможность получения и сравнения так называемых срезов данных (*slice and dice*);
- полнота и достоверность хранимых данных;
- поддержка качественного процесса пополнения данных.

Архитектура хранилища данных и его основные компоненты



Категории данных:

- Детальные
- Агрегированные
- Метаданные

Агрегированные данные:

- аддитивные
- полуаддитивные
- неаддитивные

Метаданные – сведения о данных, хранимых в хранилище данных

ETL-процесс -

Процесс управления хранилищами данных, который включает в себя:

- Извлечение данных из внешних источников
- Их трансформация и очистка, чтобы они соответствовали потребностям бизнес-модели
- Загрузка их в хранилище данных

Процедуры этапа преобразования данных в хранилище данных:

- Извлечение

- Преобразование:

- Обобщение данных
- Перевод значения
- Создание полей
- Очистка данных

- Загрузка

Проблема очистки данных

Проблемы очистки данных:

Уровень ячейки таблицы

1. Орфографические ошибки
2. Отсутствие данных
3. Фиктивные значения
4. Логически неверные значения
5. Закодированные значения
6. Составные значения

Уровень записи

Уровень таблицы базы данных

1. Нарушение уникальности.
2. Отсутствие стандартов:
 - 2.1. дублирующие записи: `emp1 = (name="John Smith", ...);
emp2 = (name="J. Smith",...);`
 - 2.2. противоречивые записи: `emp1 = (name="John Smith", bdate = 12.02.1970);
emp2 = (name="J. Smith", bdate = 12.12.1970);`

Уровень одиночной базы данных

Уровень множества баз данных

1. Различие структур баз данных
2. Одинаковые наименования разных атрибутов в разных базах данных;
3. Различное представление одинаковых данных в разных базах данных;
4. Различная классификация элементов в разных базах данных;
5. Различная временная градация данных в разных базах данных;
6. Различные значения ключевых значений, идентифицирующие один и тот же объект

△ лекция

Подходы и методы решения проблемы очистки данных

Подходы к решению проблем очистки данных

- | | |
|---------------------------------------|---|
| 1. Противоречивость информации | → 1. Удаление записей
2. Исправление противоречивых данных с учетом вероятности их появления |
| 2. Пропуски данных | → 1. Исключить объект из обработки
2. Рассчитать новое значение для пропущенных данных
а) Аппроксимация
б) Определение наиболее правдоподобного значения |
| 3. Дубликаты данных | → 1. Удаляется вся группа записей
2. Группа дубликатов заменяется на одну запись |
| 3. Аномальные значения | → 1. Удаление значения
2. Замена на ближайшее граничное значение |
| 5. Ошибки ввода данных | → 1. Проверка форматов
2. Исправление на основе различных тезаурусов
3. Использование логических зависимостей |

Подходы к построению хранилищ данных

- 1) С использованием программных продуктов
 - a. Кроссплатформенные инструментальные средства
 - b. Внутриплатформенные
- 2) Без использования специализированного ПО путём написания оригинальных программ

41. OLAP-технология. Многомерный анализ данных. Основные понятия OLAP-технологии. Тест FASMI. Операции OLAP-технологии. Классификация OLAP-средств по архитектуре. Модели хранилища данных. Многомерная модель хранилищ данных (MOLAP). Реляционная модель хранилищ данных (ROLAP). Схема "звезда". Схема "снежинка". Расширения языка SQL для OLAP-анализа данных. Классификация OLAP-средств и их примеры. Продукты класса Desktop OLAP.

OLAP-технология – технология комплексного динамического синтеза, анализа и консолидации больших объемов многомерных данных

FASMI – это основные принципы OLAP технологии.

Тест FASMI:

Fast (быстрый) – предоставление пользователю результатов анализа за приемлемое время;

Analysis (анализ) – возможность осуществления любого логического и статистического анализа; ;

Shared (разделяемой) – многопользовательский доступ к данным с поддержкой соответствующих механизмов блокировок и средств авторизованного доступа;

Multidimensional (многомерной) – многомерное концептуальное представление данных;

Information (информации) – возможность обращаться к любой нужной информации независимо от ее объема и места хранения.

OLAP-операции:

OLAP-операции:

- **сечение (рез – Slice)** - подмножество гиперкуба с фиксированным значением измерений;
- **вращение (Rotate)** - изменение порядка представления измерений;
- **консолидация (Drill up)** – обобщающие операции;
- **операция спуска (Drill down)** - отображение подробных сведений (операция обратная консолидации);
- **разбиение с поворотом (Slicing and dicing)** - представление данных с разных точек зрения.

OLAP = многомерное представление = Куб

1. Многомерное представление данных – средства конечного пользователя, обеспечивающие многомерную визуализацию и манипулирование данными;

2. Многомерная обработка – средство формулирования многомерных запросов;

3. Многомерное хранение – средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов.

		Февраль	Март	Апрель
		Январь	Сентябрь	Октябрь
Измерение	Измерение	США	Канада	Мексика
	Напитки	10 000	2000	1 000
Продукты питания		5000	500	250
Прочие товары		5000	500	250

Модели хранилищ данных:

- Многомерный (MOLAP)
- Реляционный (ROLAP)

MOLAP — классическая форма OLAP, так что её часто называют просто OLAP. Она использует суммирующую базу данных и создаёт требуемую многомерную схему данных с сохранением как базовых данных, так и агрегатов.

MOLAP (Multidimensional OLAP)

Достоинства	Недостатки
Высокая производительность	Не позволяют работать с большими базами данных
Большое число встроенных функций	Неэффективно используют внешнюю память

ROLAP работает напрямую с реляционной базой данных, факты и таблицы с измерениями хранятся в реляционных таблицах, и для хранения агрегатов создаются дополнительные реляционные таблицы.

Достоинства ROLAP:

- размер хранилища не является таким критичным параметром, как в случае MOLAP.
 - внесение изменений в структуру измерений не требует физической реорганизации базы данных;
- Главный недостаток ROLAP** – меньшая производительность.

Схема «Звезда»

Схема “звезда” (star schema): таблица фактов (fact table), таблицы измерений (dimension tables).

Схема «Снежинка»

Схема “снежинка” (snowflake schema) → детализация атрибутов в отдельных таблицах измерений
 → создание отдельных таблиц фактов для всех возможных сочетаний уровней обобщения различных измерений

Расширения языка sql для OLAP:

RISQL (Red Brick Intelligent SQL):

- диапазоны;
- перемещение итогов и средних;
- расчет текущих итогов и средних;
- сравнительные коэффициенты;
- промежуточные итоги;
- декодирование.

Классификация OLAP-средств:

Классификация OLAP-средств:

- клиентские OLAP-средства (*StatSoft, SPSS, Microsoft Excel 2000 и старше*)
- OLAP-сервера (*Oracle Express Server, IBM Informix MetaCube, IBM DB2 OLAP, Arbor Essbase, IBM Informix Red Brick, HighGate Project Sybase, Microsoft SQL Services 2000 Analysis Server*);
- DOLAP (Desktop OLAP): *SAP BusinessObjects PowerPlay* компании *Cognos*

42. Интеллектуальный анализ данных (Data Mining). Требования к обнаруживаемым знаниям. Технологии Data Mining. Машинное обучение. Модели Data Mining. Задача классификации и регрессии. Задача поиска ассоциативных правил. Задача кластеризации.

Интеллектуальный анализ данных (Data mining)

Интеллектуальный анализ данных (Data Mining) – это процесс поддержки принятия решений, основанный на поиске в данных скрытых знаний, которые ранее не были известны, нетривиальны, практически полезны, доступны для интерпретации человеком (Г.Пиатецкий-Шapiro).

Требования к обнаруживаемым знаниям:

Свойства обнаруживаемых знаний:

- Знания должны быть новые, ранее неизвестные.
- Знания должны быть нетривиальны.
- Знания должны быть практически полезны.
- Знания должны быть доступны для понимания человеку.

Технологии Data Mining:

- машинное обучение
- визуализация

Машинное обучение – методы построения алгоритмов, способных учиться.

Задачи и модели Data Mining:

- Задача классификации и регрессии

1. Задачи классификации и регрессии – отнесение объекта к одному из известных классов

- определение кредитоспособности потенциального клиента;
- фильтрация электронной почты;
- распознавание арабских цифр.

- Задача прогнозирования временных рядов

2. Задача прогнозирования временных рядов - оценка будущие значения прогнозируемых переменных.

- Задача поиска ассоциативных правил и последовательностей

- 3. Задачи поиска ассоциативных правил и последовательностей - выявление закономерностей между связанными событиями**
- анализ рыночных корзин в торговле;
 - анализ сочетания симптомов и болезней в медицине;
 - скрининговый анализ (создание товарных запасов в торговле);
- Задача кластеризации
- 4. Задача кластеризации** - разделении исследуемого множества объектов на группы объектов, называемых кластерами
- построение периодической таблицы химических элементов;
 - задача сегментации рынка;
- Описательные модели**
- $\{e_2, e_3, e_9, e_7, e_{11}, \dots\}$.

43. Распределенные базы данных. Свойства распределенных баз данных. Технологии распределенных баз данных (особенности обработки и оптимизации запросов, управление одновременным доступом, протоколы обеспечения надежности, технологии тиражирования данных). Протокол двухфазной фиксации данных. Схемы репликации данных.

Распределённые база данных – база данных, составные части которой размещаются в различных узлах компьютерной сети в соответствии с каким-либо критерием.

Обработка запроса - это процесс трансляции декларативного определения запроса в операции манипулирования данными низкого уровня.

Оптимизация запроса - это процедура выбора "наилучшей" стратегии для реализации запроса из множества альтернатив.

Двухфазовая блокировка - "Ни одна блокировка от имени какой-либо транзакции не должна устанавливаться, пока не будет снята ранее установленная блокировка"

Тиражирование данных - это асинхронный перенос изменений объектов исходной базы данных в БД, принадлежащим различным узлам распределенной системы.

Протокол двухфазной фиксации данных

Фаза 1 – подготовка к фиксации глобальной транзакции. Сервер БД направляет команду «записать» транзакцию всем локальным узлам. В случае неполучения отклика хотя бы от одного узла все локальные транзакции откатываются.

Фаза 2 – фиксация глобальной транзакции. Происходит фиксация всех локальных транзакций. В случае сбоя в течение данной фазы сервер БД гарантирует фиксацию глобальной транзакции вплоть до восстановления соединения.

44. Объектно-реляционные свойства СУБД. Сложные типы данных. Коллекции. Наследование при работе с базами данных. Определенные пользователем типы данных. Функции приведения.

Объектно-реляционная модель данных:

В объектно-реляционной модели данных используются такие объектно-ориентированные компоненты, как пользовательские типы данных, инкапсуляция, полиморфизм, наследование, переопределение методов и т.п.

Категория объектно-реляционных СУБД позволяет:

- поддерживать сложные типы данных;
- вводить новые типы данных;
- наследовать объекты: типы данных, таблицы и пр.

наследование - подразумевает возможность создавать из классов объектов новые классы объекты, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков).

Сложный тип данных:

Коллекция:

Сложные типы данных подразделяются на три категории:

- коллекции;
- строчные типы;
- типы данных, определенные пользователем.

В свою очередь коллекции делятся на множества (SET), мультимножества (MULTISET) и списки (LIST), а строчные типы делятся на именованные и неименованные.

Коллекция представляет собой сложный тип, составленный из отдельных элементов, каждый из которых принадлежит к одному и тому же типу данных, при этом элементы, в свою очередь, могут быть сложными типами, встроенными типами или типами, определенными пользователем.

Коллекция SET является множеством уникальных неупорядоченных элементов (без дублирования). Коллекция MULTISET не упорядочена, подобно SET, но допускает дублирующие значения:

В отличие от SET и MULTISET коллекция LIST является упорядоченным множеством элементов, где допустимы дублирующие значения.

В отличии от коллекций **строчные типы** являются группами элементов разных типов и формируют шаблон записи, который впоследствии можно использовать при определении таблиц.
Create row type NAME (name1 type1, name2 type2,...)

create table NAME_table of type NAME.

При добавлении такого строчного значения пишется row(p1,p2,...)::NAME.

Для обращения к отдельному полю NAME.name*.

Возможно вложение двух строчных типов.

Можно не вводить row, а писать сразу при создании таблицы.

Create distinct type NAME as известный тип.

Определенный пользователем тип данных основан на существующем типе, как встроенном, так и сложном.

Наследование:

Наследование дает возможность одному объекту приобретать свойства другого объекта, например, если одна таблица определяется как подтаблица другой, то она наследует поведение супертаблицы (ограничения, триггеры, индексы, функции и пр.). Аналогично строчный тип, определенный как подтип базового типа, наследует поля данных и поведение супертипа. Ниже приведен пример введения строчно-

Функции приведения:

Поля `Avg_summer` и `Avg_winter` и в одной, и в другой таблице – целые числа, но суть их различна. Как сравнивать эти типы? **Функция приведения (casting)** выполняет операции, необходимые для преобразования данных одного типа в другой. Приведения могут быть:

- неявные; они выполняются сервером с использованием множества встроенных функций приведения между всеми базовыми типами;
- явные; между новым типом и типом-источником.

В случае явных функций приведения необходимо использовать конструкцию «<источник>::<новый тип>» или функцию «`Cast (источник as новый тип)`».

```
insert into rsa_city values ("Capetown", 3200000, 21::celsius, 14::celsius);
```

45. NoSQL базы данных. Обеспечение согласованности данных. Теорема CAP. Модель ключ-значение. Основные операции в базах данных ключ-значение. Документная модель данных. Выбор уровня денормализации при использовании документных баз данных. Столбцовая модель данных. Основное назначение использования столбцовых баз данных.

Графовая модель данных. Типы графовых моделей данных. Достоинства и недостатки NoSQL баз данных.

NoSQL базы данных:

Термин NoSQL может трактоваться и как отрицание применения SQL как языка доступа к данным, так и как возможность использования альтернативных методов доступа (Not Only SQL).

Обеспечение согласованности данных:

Теорема CAP:

Обеспечение согласованности данных

Свойства ACID, как правило, обеспечиваются NoSQL СУБД только на уровне отдельного агрегата, так как транзакции, затрагивающие несколько агрегатов, в общем случае являются распределенными. Для распределенных баз данных (к которым относятся NoSQL базы данных) было сформулировано следующее утверждение, ставшее известным как **теорема CAP**.

Теорема CAP. В распределенной системе из трех характеристик согласованность (C), доступность (A), устойчивость к разделению сети (P) могут выполняться только две.

Модель ключ-значение:

6.2. Модель ключ-значение

Структура данных в этой модели представляет собой ассоциативный массив с двумя основными операциями.

- Операция получения значения по ключу. Если ключ не существует, то будет возвращено NULL-значение.
- Запись значения по ключу. Удаление ключа можно рассматривать как вариант этой операции, когда устанавливается NULL-значение. Важная возможность – задание времени существования ключа. По истечении этого времени значение ключ будет удален. Также к этой операции относится модификация значения по ключу.

Столбцовая модель данных, как и документная, является развитием модели ключ-значение. В ней объектом хранения данных является семейство столбцов (аналогично таблице в реляционных базах данных и коллекции в документных).

Столбцовые базы данных более эффективны в случаях, когда часто требуется вычислять какие-то совокупные величины — средние, суммы, и прочие по каким-то значениям.

Графовые модели данных можно разделить на следующие основные классы:

- модель с метками и свойствами;
- тройные кортежи;
- гиперграфы.

6.6. Общие замечания по NoSQL моделям данных

Можно выделить несколько основных причин широкого внедрения баз данных NoSQL, включая:

- потребность в больших возможностях масштабирования, чем у реляционных баз данных, включая обработку очень больших наборов данных или очень большую пропускную способность по записи;
- предпочтение свободного программного обеспечения вместо коммерческих продуктов;
- специализированные запросные операции, плохо поддерживающиеся реляционной моделью;
- стремление к более динамичным и выразительным моделям данных.

Однако каждая из NoSQL моделей вместе с возможностями по эффективной обработке большого объема данных несет в себе существенные ограничения по применимости. В итоге у каждого приложения имеются свои требования, и оптимальная технология может различаться в зависимости от сценария использования. В ближайшем будущем реляционные базы данных будут продолжать задействовать возможности разнообразных нереляционных баз данных. Такой способ называется *применение нескольких систем хранения данных в одном приложении* (*polyglot persistence*).

Также в настоящее время разрабатываются новые системы управления базами данных, категория которых носит общее название NewSQL и характеризуется следующими особенностями:

- SQL как основной механизм для взаимодействия;
- ACID поддержка транзакций;
- механизм управления без применения блокировок, таким образом считающие данные в реальном времени не будут находиться в противоречии с записывающими;
- удобное масштабирование, способное управлять большим количеством узлов, не перенося узкие места.

46. Основные принципы, лежащие в основе темпоральных баз данных. Понятие времени в темпоральных базах данных. Модели, используемые в темпоральных базах данных (TRM, HDM).

Основные принципы временных баз данных

1. Расширяется реляционная модель. К двумерным объектам (кортежи, атрибуты) добавляется третье измерение – время, в результате чего мы получаем трехмерное временное отношение (рис. 7.2). При этом время автоматически встраивается в само отношение.

2. Уточняется понятие «время».

Термины	Понятия
Эффективное время (действительное время)	Время, когда факт вступает в силу в действительности
Время регистрации (время транзакции)	Время, когда значение фактически помещается в базу данных
Время, определенное пользователем	Обычный домен времени, не представимый в виде временного куба

В модели TRM используется концепция временной нормальной формы TNF (Time Normal Form): отношение находится в TNF тогда и только тогда, когда оно находится в нормальной форме Бойса–Кодда и не существует каких-либо временных зависимостей между неключевыми атрибутами.

2. Historical Data Model (HDM)

Данная модель временной базы данных связана с так называемыми периодами жизни, которые ассоциируются с объектами различной гранулярности (рис. 7.4).

47. Безопасность баз данных. Проблема безопасности. Построения модели угроз.

Основные термины и понятия, связанные с информационной безопасностью.

Защита баз данных. Внутренние и внешние источники угроз. Принципы построения защищенных систем баз данных. Угрозы безопасности баз данных. Организационно-административные и компьютерные средства защиты базы данных.

Угрозы безопасности баз данных

- | | |
|---------------------------------------|--|
| Угрозы безопасности баз данных | - внешние источниками угроз
- внутренними источниками угроз |
|---------------------------------------|--|
- Внешние источниками угроз:**

 - умышленные действия лиц с целью искажения, уничтожения или хищения программ и данных системы;
 - искажения в каналах передачи информации, поступающей от внешних источников;
 - сбои и отказы в аппаратуре вычислительных средств;
 - вирусы и иные деструктивные программные элементы, распространяемые с использованием систем телекоммуникаций.

Внутренними источниками угроз:

 - ошибки при постановке целей и задач проектирования ИС, алгоритмов безопасности, допущенные при формулировке требований к функциям безопасности системы;
 - ошибки и несанкционированные действия пользователей, административного и обслуживающего персонала в процессе эксплуатации системы;
 - недостаточная эффективность используемых методов и средств обеспечения ИБ в штатных или особых условиях эксплуатации системы.

Принципы построения защищенных систем БД:

- экономическая оправданность механизма защиты;
- принцип открытого проектирования;
- принцип распределения полномочий;
- принцип минимально возможных привилегий для пользователей и администраторов;
- принцип управляемости;
- принцип психологической приемлемости работы средств защиты.

Система является **безопасной** (*security system*), если она управляет доступом к информации таким образом, что только авторизованные лица или действующие от их имени процессы (программы) имеют право читать, изменять, создавать или удалять информацию.

Анализ проблемы безопасности

1. Комплексная проблема
2. Сложность задачи → множество источников угроз: ОТСС, ВТТС, человеческий фактор
→ различные способы защиты
3. Цена вопроса
4. Усложнение в использовании
5. Длинная окупаемость



Безопасность баз данных

Конфиденциальная информация (sensitive information) – информация, требующая защиты.

Доступ к информации (access to information) – ознакомление с информацией, ее обработка, модификация, уничтожение.

Субъект доступа (access subject) – лицо или процесс, действия которого регламентируются правилами разграничения доступа.

Объект доступа (access object) – единица информации автоматизированной системы, доступ к которой регламентируется правилами разграничения доступа.

Правила разграничения доступа (security policy) – совокупность правил, регламентирующих права субъектов доступа к объектам доступа.

Санкционированный доступ (authorized access to information) – доступ к информации, который не нарушает правил разграничения доступа.

Несанкционированный доступ (unauthorized access to information) – доступ к информации, который нарушает правила разграничения доступа с использованием штатных средств, предоставляемых средствами вычислительной техники или автоматизированными системами.

Идентификатор доступа (access identifier) – уникальный признак объекта или субъекта доступа.

Идентификация (identification) – присвоение объектам и субъектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов.

Пароль (password) – идентификатор субъекта, который является его секретом.

Безопасность баз данных

Аутентификация (authentication) – проверка принадлежности субъекту доступа предъявленного им идентификатора, подтверждение подлинности.

Уровень полномочий субъекта доступа (subject privilege) – совокупность прав доступа субъекта доступа.

Нарушитель правил разграничения доступа (security policy violator) – субъект доступа, который осуществляет несанкционированный доступ к информации.

Модель нарушителя (security policy violator model) – абстрактное (формализованное или неформализованное) описание нарушителя правил разграничения доступа.

Целостность информации (information integrity) – способность средства информационной системы обеспечить неизменность информации в условиях случайного и (или) преднамеренного искажения (разрушения).

Метка конфиденциальности (sensitivity label) – элемент информации, характеризующий конфиденциальность объекта.

Многоуровневая защита (multilevel secure) – защита, обеспечивающая разграничение доступа субъектов с различными правами доступа к объектам различных уровней конфиденциальности.

Угрозы безопасности баз данных

Защита баз данных – это комплекс мер по обеспечению защищенности баз данных против любых угроз и опасностей с помощью различных компьютерных и некомпьютерных средств



Задача: снижение вероятности реализации потенциальных угроз до приемлемого для конкретной системы уровня

Угрозы безопасности баз данных

Угрозы безопасности баз данных



I. Угрозы конфиденциальности информации, специфичные для баз данных:

1. Инъекции SQL
2. Логический вывод на основе функциональных зависимостей $R(A1, A2, \dots, An) \quad X \rightarrow Y$
3. Логический вывод на основе ограничений целостности
4. Использование оператора Update для получения конфиденциальной информации

II. Угрозы целостности информации, специфичные для баз данных:

1. Неблагоприятные стечения обстоятельств и состояние внешней среды
2. Неадекватные действия пользователей: ошибки при вводе данных
3. Проблемы, возникающие при многопользовательской работе

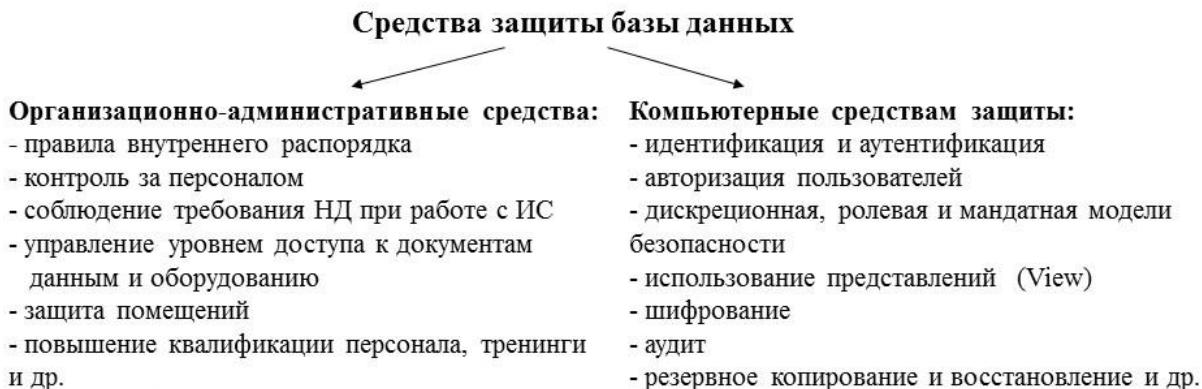
III. Угрозы доступности информации, специфичные для баз данных:

1. Перегрузка системы
2. Действия, направленные на остановку критически важных процессов
3. Использование свойств первичных и внешних ключей $select * from tab1, tab1 order by 1$
Мощность $tab1 = 10\ 000$
4. Блокировка записей при изменении
Мощность результата – $100\ 000\ 000$
5. Загрузка системы бессмысленной работой

Безопасность баз данных

Политика безопасности – совокупность норм и правил, регламентирующих процесс обработки информации, выполнение которых обеспечивает защиту от определенного множества угроз и составляет необходимое (а иногда и достаточное) условие безопасности системы.

Моделью безопасности – формальное выражение политики безопасности



48. Идентификация и аутентификация. Атаки, специфичные для баз данных. Авторизация. Модель политики безопасности. Дискреционная модель безопасности. Ролевая модель безопасности.

Безопасность баз данных

Идентификация – операция различение субъектов, объектов, процессов в форме имен: присвоение субъектам и объектам доступа личного идентификатора и сравнение его с заданным перечнем

CREATE USER IDENTIFIED BY пароль → **Login-процедура**

Аутентификации – процедура подтверждении подлинности пользователя, представившего идентификатор

Авторизация – операция определения перечня информационных ресурсов, с которыми аутентифицированному пользователю разрешена работа

Схемы процедур аутентификации

- аутентификация, основанная на знании;
- аутентификация, основанная на наличии;
- аутентификация, основанная на проверке характеристик.

Методы, уменьшающие угрозу компрометации паролей

1. Ограничение срока действия пароля
2. Ограничение на содержание пароля
3. Блокирование терминала.
4. Блокирование пользователя
5. Разовый пароль

Атаки, специфичные для баз данных

I. Подбор паролей как метод реализации несанкционированного доступа

1. Тотальный перебор
2. Тотальный перебор, оптимизированный по статистике встречаемости символов
3. Тотальный перебор, оптимизированный с помощью словарей
4. Подбор паролей с использованием знаний о пользователе

Модели политики безопасности

Авторизация – определение прав в системе для конкретного лица

Модель безопасности



I. Дискреционное управление доступом

Дискреционное управление доступом (discretionary access control) – разграничение доступа между поименованными субъектами и поименованными объектами.

субъект доступа - тип доступа - объект доступа

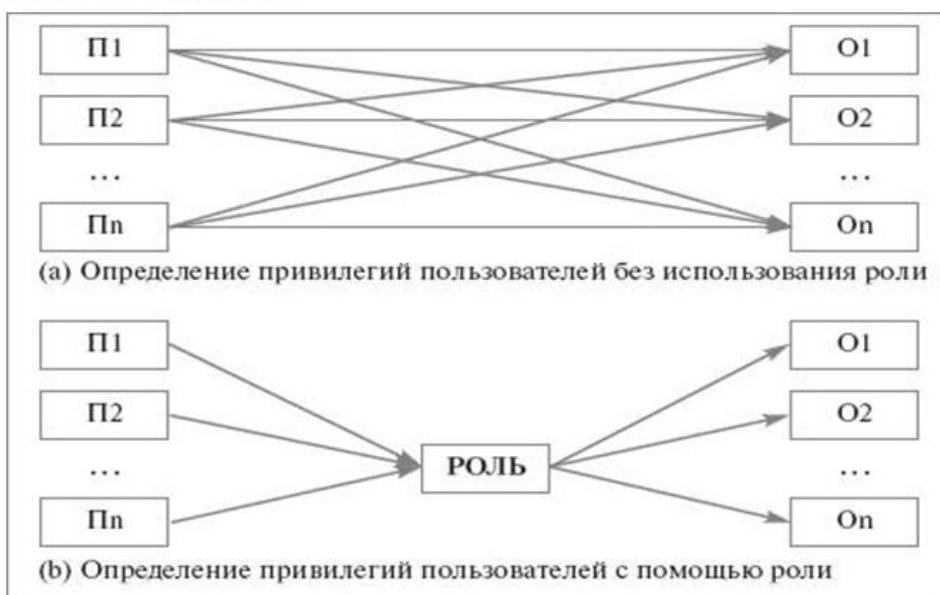
	Объект1 (продажи)	Объект2 (выплаты)	Объект3	Объект4	Объект5	
Иванов	ЧМ	ЧМСУ		ЧМСУ		Ч - чтение
Петров	ЧСМ		ЧМСУ		ЧСМ	М – модификация
Сидоров	ЧМСУ					С – создание
Попов	ЧМСУ	ЧМСУ	ЧМСУ		ЧМ	У – удаление
Козлов		Ч	Ч			
.....						

Объект защиты – таблица, представление, хранимая процедура и т.д.

Субъект защиты – пользователь, группа пользователей, а также хранимая процедура

Модели политики безопасности

II. Ролевое управление доступом



Роль идентифицирует динамически образуемую группу пользователей, каждый из которых обладает – привилегией на исполнение данной роли;
– всеми привилегиями данной роли для доступа к объектам базы данных.

CREATE ROLE идентификатор_роли IDENTIFIED BY пароль_роли

49. Мандатная модель безопасности. Модель многоуровневой безопасности Белла Лападула и ее основные свойства. Структура метки доступа. Последовательность шагов управления мандатным доступом в Oracle. Принцип многоэкземплярности.

Модели политики безопасности

III. Мандатное управление доступам

Модель многоуровневой защиты (multilevel secure) – защита, обеспечивающая разграничение доступа субъектов с различными правами доступа к объектам различных уровней конфиденциальности.

Цель: обеспечение условий, при которых невозможно создание информационных потоков от субъектов с более высоким уровнем доступа к объектам с более низким уровнем доступа

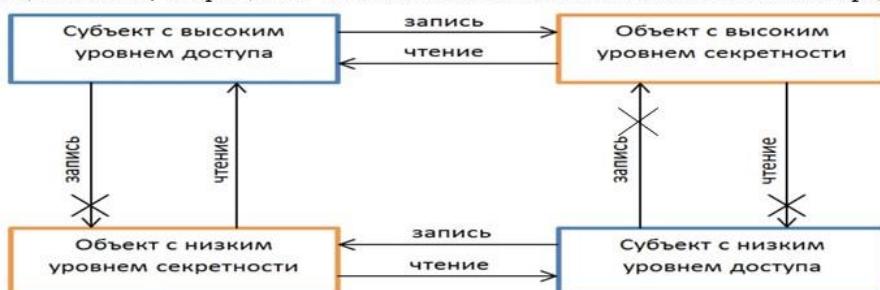
Модель Белла-Лападула (Bell-LaPadula): объекты базы данных классифицируются по уровням конфиденциальности, а каждый субъект причисляется к одному из классов доступа объекта БД

Инструменты разграничения доступа субъектов к объектам данных:

- метки конфиденциальности информации, содержащейся в объектах;
- и разрешения субъектов обращаться к информации данного уровня конфиденциальности.

Правила разграничения доступа в модели Белла-Лападула:

- субъект, которому разрешён доступ только к объектам с более низкой меткой конфиденциальности, не может получить доступ к объекту с более высокой меткой конфиденциальности;
- субъект, которому разрешён доступ только к объектам с более высокой меткой конфиденциальности, запрещается запись в объекты с более низкой меткой конфиденциальности.



Модели политики безопасности

III. Мандатное управление доступам

Метка объекта:

- Группа субъекта, который внес данный объект (класс субъекта).
- Уровень доступа на чтение – RAL (Read Access Level).
- Уровень доступа на запись – WAL (Write Access Level).

Метка субъекта:

- Группа, к которой принадлежит субъект (класс субъекта);
- RAL-уровень субъекта; пользователь может получать (читать) информацию, RAL-уровень которой не выше его собственного уровня доступа;
 - **принцип простого свойства секретности (simple security property) - No Read Up**
- WAL-уровень субъекта; пользователь может вносить информацию только в объекты такого же WAL-уровня доступа.
 - ***-свойство - No Write Down**

Принцип многоэкземплярности в модели Белла-Лападула: в рамках одного отношения может существовать множество кортежей с одним и тем же значением первичного ключа.

50. Дополнительные инструменты защиты базы данных. Представления. Аудит. Резервное копирование и восстановление. Виды резервного копирования. Методы восстановления. Метод наката. Метод отката. Контрольные точки.

Дополнительные инструменты для защиты базы данных

1. Представление – виртуальная таблица, с помощью которых действительная структура базы данных может быть скрыта от тех или иных групп пользователей.

CREATE VIEW

2. Аудит БД – процедура выявления некорректных или неавторизованных действий пользователей.

3. Резервное копирование и восстановление

Резервное копирование (backup) – периодически выполняемая процедура получения копии базы данных и её журнала транзакций на носителе, сохраняемом отдельно от системы.

Полное резервное копирование базы данных (full backup)
Дифференциальное резервное копирование (differential backup)
Инкрементальное (добавочное) резервирование (incremental backup)
Резервное копирование файлов и групп файлов
Резервное копирование журнала транзакций

Безопасность баз данных

3. Резервное копирование и восстановление

Мягкие сбои → метод отката (rollback) **Жесткие сбои → метод наката (rollforward)**

Метод отката: 1. Повторное выполнение завершенных транзакций (redo)
 2. Отмена незавершенных транзакций (undo)

Контрольная точка (checkpoint) - точка времени синхронизации между БД и журналом транзакций

Метод наката: 1. Загрузка архивной копии с помощью утилиты восстановления
 2. Перезапуск всех транзакций, которые закончились к моменту сбоя