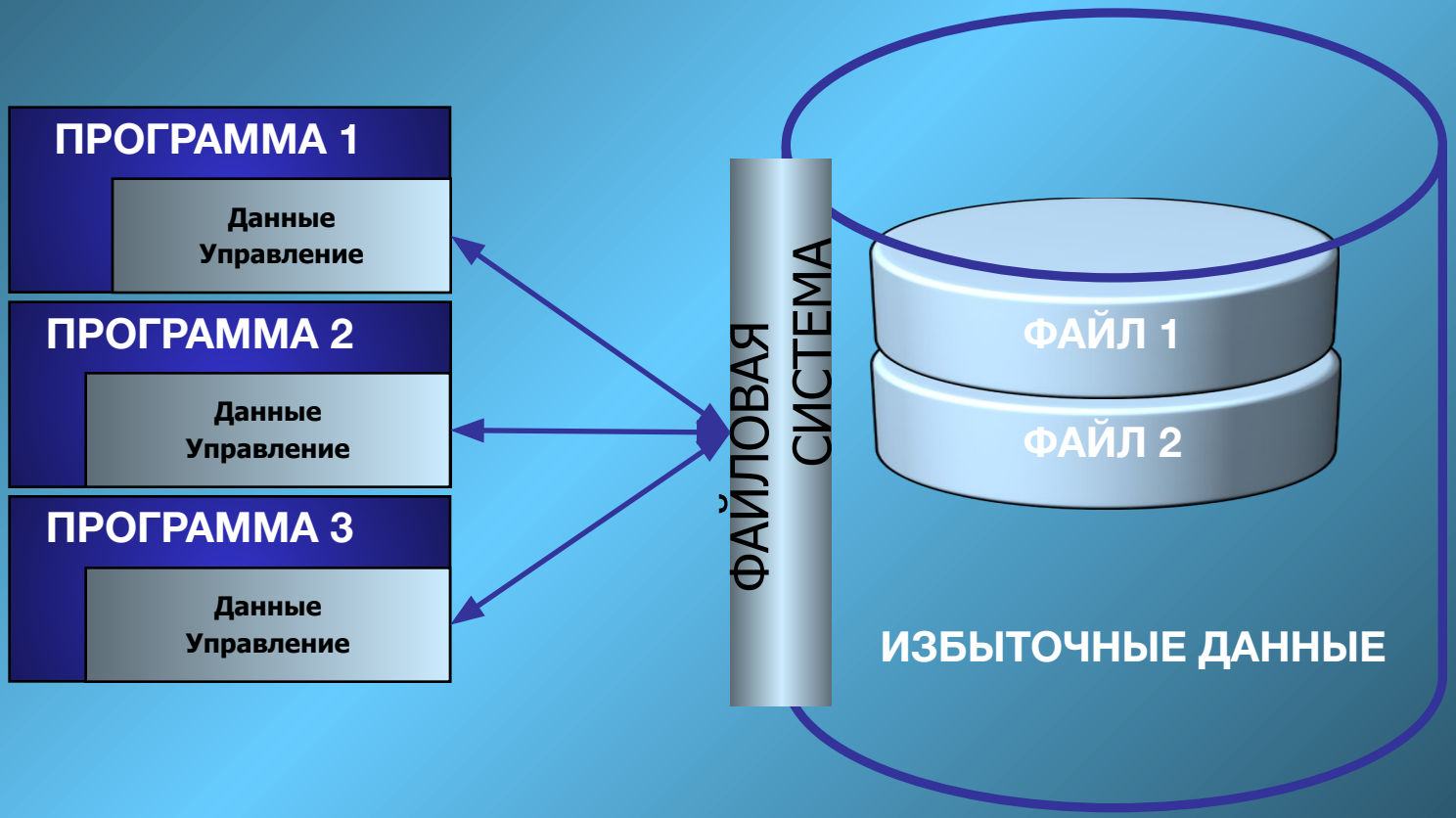
БАЗЫ ДАННЫХ

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

БАЗЫ ДАННЫХ
ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ



БАЗЫ ДАННЫХ
ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

ПРОГРАММА 1

Данные
Управление

ПРОГРАММА 2

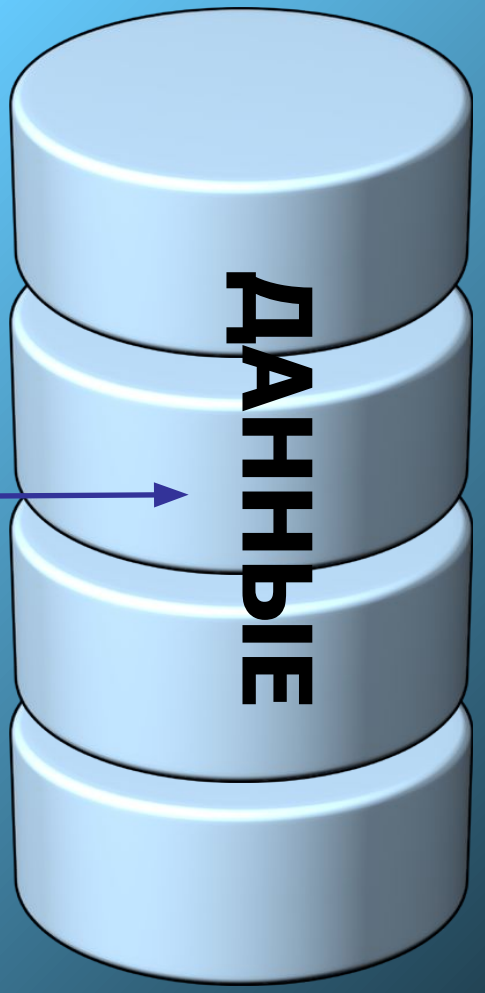
Данные
Управление

СИСТЕМА УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ

ОБРАБОТЧИК ЗАПРОСОВ

МЕНЕДЖЕР ТРАНЗАКЦИЙ

...



ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ БАЗ ДАННЫХ

- Независимость данных – сокращение размеров программной поддержки (внутри отдельных программ)
- Увеличение эффективности разработки приложений
- Возможность создания и использования стандартов.
- Минимальная избыточность хранения данных.
- Увеличение плотности данных и совместного доступа к данным.
- Улучшенный доступ к данным и их соответствие конкретным решаемым задачам.
- Увеличение качества данных.
- Безопасность, сохранение и восстановление.

БАЗЫ ДАННЫХ
ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

БАЗА ДАННЫХ (БД) – именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ (СУБД) – совокупность языковых и программных средств, предназначенных для создания, ведения и совместного применения БД многими пользователями.

СИСТЕМА БАЗЫ ДАННЫХ (система БД) – СУБД с наполненной базой данных, управляемой её средствами.

БАНК ДАННЫХ (БнД) – основанная на технологии баз данных система языковых, программных, организационных и технических средств, предназначенных для централизованного накопления и коллективного использования данных.

ИНФОРМАЦИОННАЯ СИСТЕМА (ИС) – система, реализующая автоматизированный сбор, обработку и манипулирование данными и включающая технические средства обработки данных, программное обеспечение и соответствующий персонал.

ТРЕБОВАНИЯ К СОВРЕМЕННЫМ СУБД

- Масштабируемость – отсутствие существенного снижения скорости выполнения пользовательских запросов при пропорциональном росте количества запросов и аппаратных ресурсов используемых данной СУБД (таких как объём оперативной памяти, количество процессоров и серверов);
- Доступность – возможность всегда выполнить запрос;
- Надёжность – минимальная вероятность возникновения сбоев, наличие средств восстановления данных после сбоев, инструментов резервного копирования и дублирования данных, обеспечивающих возможность осуществлять эти операции не прерывая работу пользователей;
- Управляемость – простота администрирования, наличие средств автоматического конфигурирования (включает: средства создания баз данных и их объектов, средства описания правил репликации данных, утилиты управления пользователями, средства мониторинга событий, утилиты миграции из других СУБД и т.д.);
- Наличие средств защиты данных от потери и несанкционированного доступа;
- Поддержка доступа к данным с помощью веб-служб (веб-сервисов);
- Поддержка стандартных механизмов доступа к данным: ODBC, JDBC, OLE, DB, ADO .NET и др.

ОБЛАСТИ ПРИМЕНЕНИЯ БАЗ ДАННЫХ

- Data Mining
- Material Requirement Planning – планирование потребностей в материалах;
- Manufacturing Resource Planning – планирование ресурсов производства;
- Enterprise Resource Planning – планирование ресурсов производства + управление финансами и кадрами предприятия;
- Enterprise Resource and Relationship Processing – планирование ресурсов предприятия + взаимоотношения с клиентами и поставщиками, управление цепочками поставок;
- Decision-Support Systems - Системы принятия решений;
- Data Analysis - Анализ данных;
- Data Warehousing - Организация хранилищ данных;
- Spatial and Geographic Databases - Географические базы данных;
- Multimedia Databases - Мультимедиа базы данных;
- Mobility and Personal Databases - Мобильные и персональные БД;
- Information-Retrieval Systems – Информационные системы;
- Distributed Information Systems - Распределенные информационные системы;
- The World Wide Web (WWW) - Система World Wide Web.

БАЗЫ ДАННЫХ «МЕСТО» В КОМПЬЮТЕРНОЙ СИСТЕМЕ

КЛАССИФИКАЦИЯ БАЗ ДАННЫХ

- ***По модели данных***
  - На основе инвертированных списков
  - Иерархические
  - Сетевые
  - Реляционные
  - Объектно-ориентированные и мультимедийные (постреляционные)
- ***По количеству пользователей***
  - Персональные (настольные)
  - Уровня рабочей группы
  - Масштаба предприятия
  - Корпоративные
- ***По организации системы***
  - Распределённые
  - Централизованные
- ***По…***
  - Фактографические
  - Полнотекстовые

ИНВЕРТИРОВАННЫЕ СПИСКИ

Типичный пример: **Adabas/Natural**, фирма Softvare AG, Германия 1966 г.

Другой типичный представитель: **Datacom/DB** компании Applied Data Research, Inc. (ADR).

Также построены на инвертированных списках: **dBase, Clipper, FoxPro, Paradox**.

Основные особенности:

Данные хранятся в таблицах. Записи таблиц фиксированной длины, состоящие из множества различных типов полей, упорядочены в некоторой последовательности.

Для каждой таблицы строится *произвольное число индексов* (инвертированных списков), хранимых в отдельных файлах. Индексы автоматически поддерживаются системой.

Пути доступа к таблицам и спискам видны пользователям.

Общие определения целостности базы данных отсутствуют.

Типовые операторы манипулирования:

НАЙТИ ПЕРВУЮ ЗАПИСЬ, НАЙТИ ПЕРВУЮ ПО КЛЮЧУ, НАЙТИ СЛЕДУЮЩУЮ, ОБНОВИТЬ, УДАЛИТЬ, ДОБАВИТЬ ПУСТУЮ ЗАПИСЬ, ИЗМЕНИТЬ ПОЛЕ ЗАПИСИ.

ИЕРАРХИЧЕСКИЕ БАЗЫ ДАННЫХ

Типичный пример: **Information Management System (IMS)** фирмы IBM (1968).

Основные особенности:

Строится иерархия (предки, потомки).

Предки и потомки. Один предок – множество потомков.

Частичная поддержка целостности в переделах дерева - основное правило: никакой потомок не может существовать без своего родителя.

Типовые операторы манипулирования:

найти указанное дерево БД; перейти от одного дерева к другому; перейти от одной записи к другой внутри дерева; перейти от одной записи к другой в порядке обхода иерархии; вставить новую запись в указанную позицию; удалить текущую запись.

НОВАЯ ЖИЗНЬ IMS

IBM обновила свой старейший программный продукт - систему управления иерархическими базами данных и транзакциями Information Management System (IMS). Система IMS, разработанная тридцать лет назад для поддержки космической программы Apollo, завоевала широкое признание в отрасли. В настоящее время почти 95% компаний из списка Fortune 1000 используют систему IMS для решения самых ответственных задач по управлению данными на мэйнфреймах IBM System z, с использованием баз данных IMS ежедневно совершается более 50 млрд. транзакций. В новой версии IMS 10 реализована поддержка технологий Xquery, Enhanced XML и Web-сервисов, что обеспечит доступ к данным IMS с помощью стандартных инструментов сторонних поставщиков. Расширенный инструментарий XML и Java предоставляет возможность разработки новых приложений. Динамическое определение ресурсов обеспечивает повышение управляемости и упрощает установку и использование системы, способствуя уменьшению совокупной стоимости владения. Улучшены также характеристики системы, как масштабируемость, доступность, восстанавливаемость, производительность, безопасность и емкость. Кроме того, усовершенствован шлюз IMS SOAP Gateway, представляющий транзакции IMS в виде Web- сервисов для обеспечения взаимодействия с приложениями – клиентами вне зависимости от их местоположения, языка программирования и платформы.

Инновации в технологиях и бизнесе №1/2008 г.

СЕТЕВЫЕ БАЗЫ ДАННЫХ

Типичный пример: **Integrated Database Management System** (IDMS), компания Cullinet Software, Inc., США, 1971 г. Предназначенная для использования на мэйнфреймах IBM. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL).

Основные особенности:

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

Ограничения целостности не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

Манипулирование данными:

найти конкретную запись в наборе однотипных записей; перейти от предка к первому потомку по некоторой связи; перейти к следующему потомку в некоторой связи; перейти от потомка к предку по некоторой связи; создать новую запись; уничтожить запись; модифицировать запись; включить в связь; исключить из связи; переставить в другую связь и т.д.

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ

SQL/DS, DB2 Universal Database – IBM;

Oracle;

MS SQL Server;

Adaptive Server Enterprise и Adaptive Server IQ – компании Sybase;

Teradata Database;

Netezza Performance Server 8000 Series;

Postgress;

MySQL;

Firebird…

NoSQL

NoSQL (англ. not only SQL, не только SQL) — термин, обозначающий ряд моделей баз данных, имеющих существенные отличия от используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL.

Описание схемы данных в случае использования NoSQL-решений может осуществляться через использование различных структур данных: хеш-таблиц, деревьев и других.

Одна из проблем классических реляционных БД, проблема при работе с данными очень большого объёма в проектах с высокой нагрузкой. Основная цель подхода — расширить возможности БД там, где SQL недостаточно гибок, и не вытеснять его там, где он справляется со своими задачами

NoSQL

NoSQL не является полным отрицанием языка SQL и реляционной модели, проект исходит из того, что SQL — это важный и весьма полезный инструмент, но при этом он не может считаться универсальным.

Одной из проблем, которую указывают для классических реляционных БД, являются проблемы при работе с данными очень большого объёма в проектах с высокой нагрузкой.

Основная цель подхода — расширить возможности БД там, где SQL недостаточно гибок, и не вытеснять его там, где он справляется со своими задачами.

NoSQL

В качестве одного из методологических обоснований подхода NoSQL используется эвристический принцип, известный как теорема CAP, утверждающий, что **в распределённой системе** невозможно одновременно обеспечить согласованность данных, доступность (в смысле наличия отклика по любому запросу) и устойчивость к расщеплению распределённой системы на изолированные части.

Таким образом, при необходимости достижения высокой доступности и устойчивости к разделению предполагается не фокусироваться на средствах обеспечения согласованности данных, обеспечиваемых традиционными SQL-ориентированными СУБД с транзакционными механизмами на принципах ACID.

NoSQL

В июле 2011 компания Couchbase, разработчик CouchDB, Memcached и Membase, анонсировала создание нового SQL-подобного языка запросов — UnQL (Unstructured Data Query Language).

Работы по созданию нового языка выполнили создатель SQLite Ричард Гипп (англ. Richard Hipp), и основатель проекта CouchDB Дэмиен Кац (англ. Damien Katz). Разработка передана сообществу на правах общественного достояния

NoSQL

Поколоночные СУБД:

- **Hadoop / HBase**
- **Cassandra**
- **Hypertable**
- **Cloudata**
- **Cloudera**
- **Amazon SimpleDB**
- **SciDB**

NoSQL

Хранилища «ключ-значение», кортежные хранилища:

- **Azure Table Storage**
- **MEMBASE**
- **Riak**
- **Redis**
- **Chordless**

NoSQL

Документо-ориентированные СУБД:

- **CouchDB**
- **MongoDB**
- **IBM Lotus Notes**
- **Terrastore**
- **ThruDB**
- **OrientDB**
- **RavenDB**
- **BaseX**
- **Citrusleaf**
- **SisoDB**

HADOOP

Hadoop — проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и программный каркас для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов.

Используется для реализации поисковых и контекстных механизмов многих высоконагруженных веб-сайтов, в том числе, для Yahoo! и Facebook.

Разработан в рамках вычислительной парадигмы MapReduce, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.

NoSQL

NoSQL исторически появился раньше SQL-а, собственно весь ынтырпрайз до 70-х им в жопу и долбился. Потом британский ученый изобрёл теорию РБД, появление которой привело к немедленному выметанию всего этого ёбаного хаоса с рынка, стандартизации и тотальному овладиванию SQL-а в рекордно короткие сроки. Побочным эффектом стало то, что всякое быдло начало пихать SQL туда, где он не очень-то и нужен, и очень, блядь, страдать, от того, что их гостевухи стали долго загружаться. Потом кто-то сделал фундаментальное открытие — оказывается хранить профили пользователей гостевухи в хешь-таблице в памяти и доставать их оттуда по имени гораздо быстрее, чем реализовывать EAV поверх РБД. После этого переворота в мозгах гостевушников они приняли радостно сверкать новым базвордом по своим блогам и хабро-хабрикам, радуясь, что им в очередной раз удалось повернуть стрелку прогресса вспять и укусить себя за жопу.

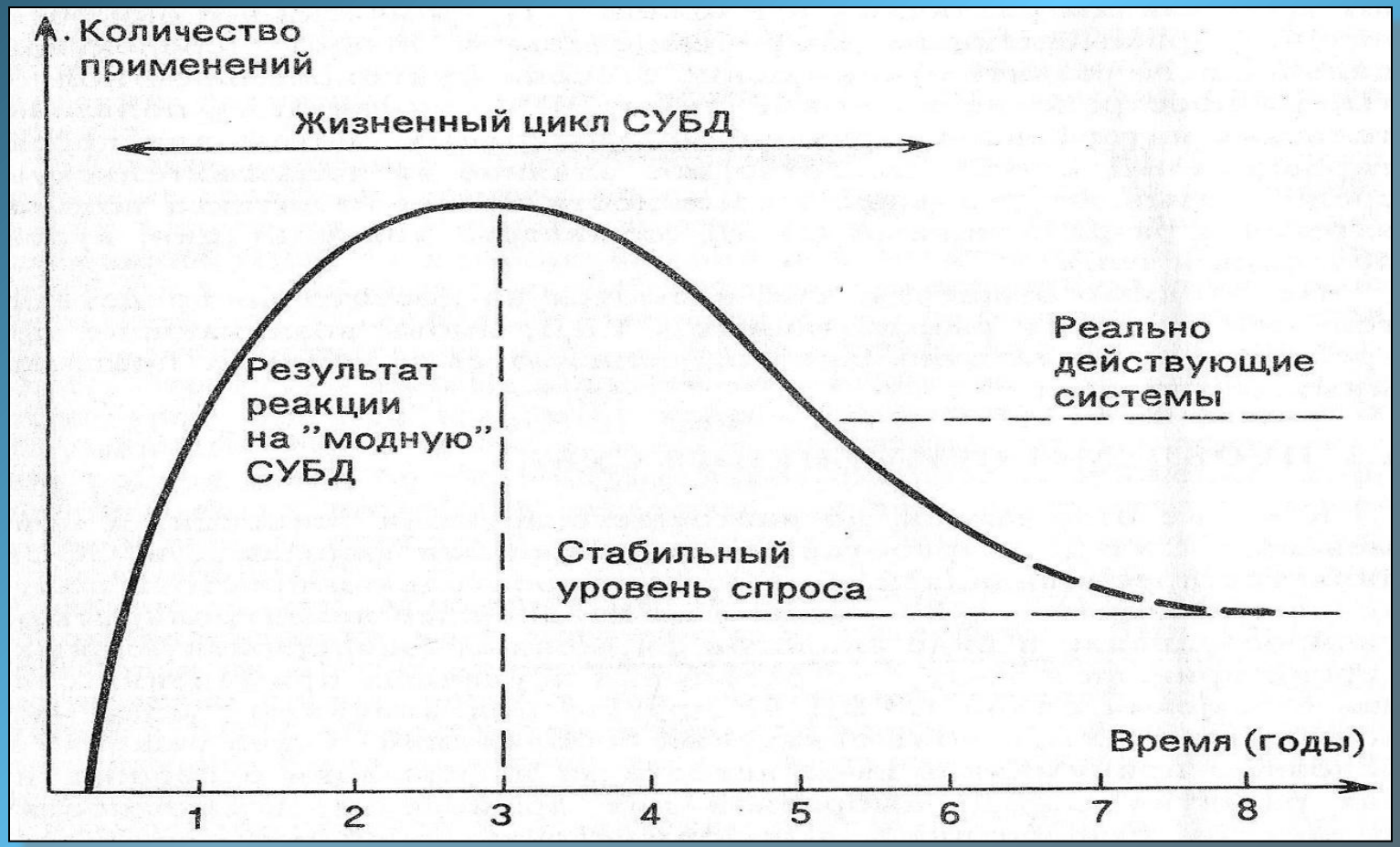
http://habrahabr.ru/post/144613/

ИСТОРИЯ ВОЗНИКНОВЕНИЯ СОВРЕМЕННЫХ БАЗ ДАННЫХ

- Списки - 1964 г. Файловые системы – наборы данных, OS/360, IBM, США.
- СУБД на основе инвертированных списков – 1966 г. Adabas/Natural, Softvare AG, Германия.
- Иерархические СУБД – 1969 г. IMS, IBM, США.
- Сетевые СУБД – 1971 г. IDMS, Cullinet, США.
- Реляционные СУБД – 1975 г. SQL/DS (теперь DB2), IBM, США. 1979 г. Oracle.
- 1976: Питер Чен определил модель Сущность-связь (ER)
- Полнотекстовые СУБД – STAIRS, IBM, США.

- Объектно-ориентированные и мультимедийные СУБД (постреляционные СУБД) - ~1990 г.
- NoSQL

ДИНАМИКА СПРОСА НА ПРОМЫШЛЕННО-СОПРОВОЖДАЕМЫЕ СУБД



ЖИЗНЕННЫЙ ЦИКЛ БАЗЫ ДАННЫХ

СУБД – ОДИН ИЗ НАИБОЛЕЕ «ДОЛГОЖИВУЩИХ» ПРОДУКТОВ

- СУБД ADABAS/NATURAL, фирма Software AG, Германия – выпуск первой версии в 1966 г.
- СУБД IMS, фирма IBM, США – выпуск первой версии в 1968 г.
- СУБД IDMS, фирма Cullinet, США – выпуск первой версии в 1971 г.
- СУБД SQL/DS (DB2), фирма IBM, США – выпуск первой версии в 1975 г.
- СУБД Oracle – выпуск первой версии в 1979 г.

Все эти СУБД используются и в настоящее время.

Постоянно выходят их новые версии.

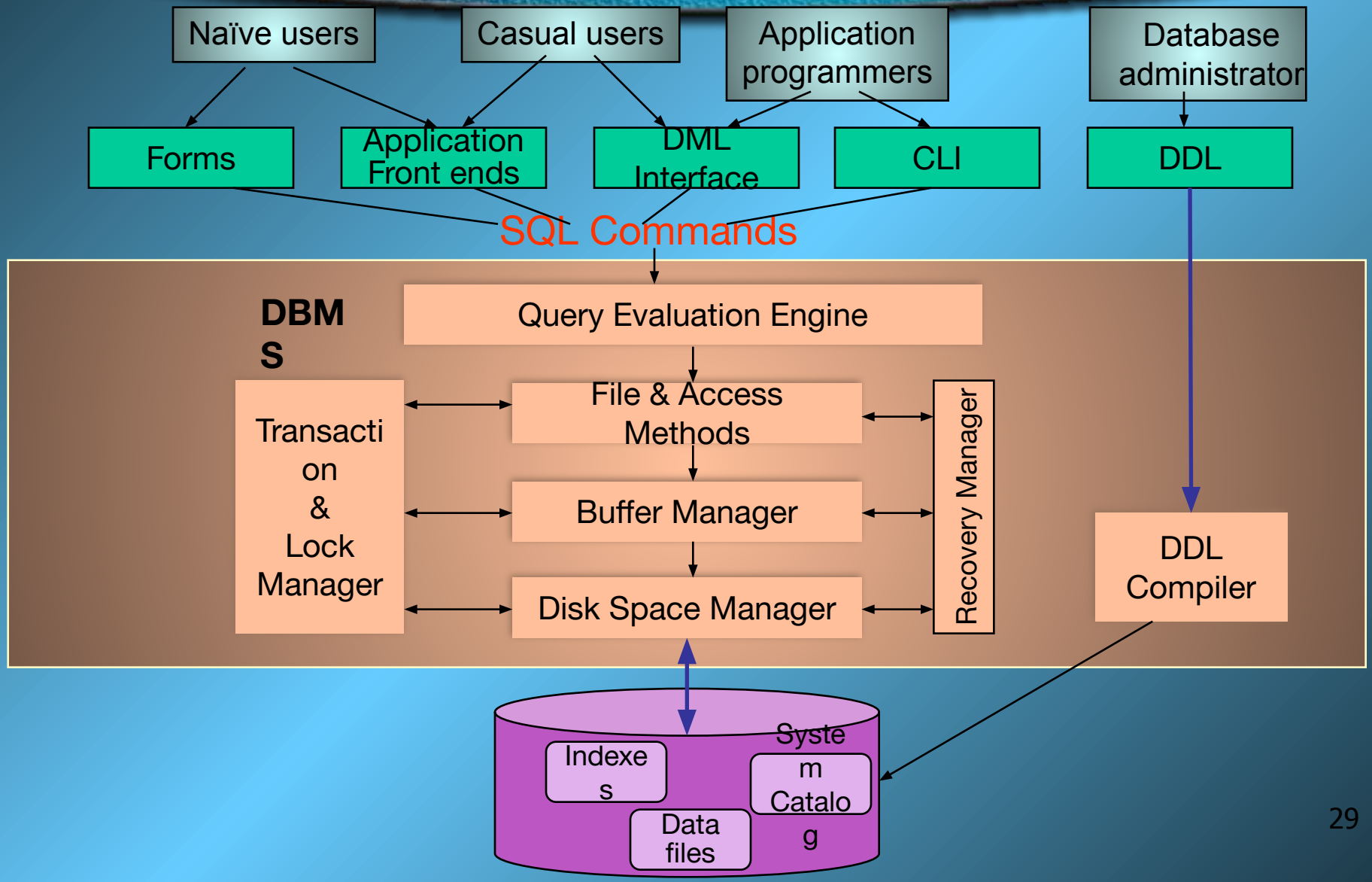
ПЕРЕХОД НА НОВУЮ СУБД

Объективными причинами перехода на новую СУБД являются:

- Недостаточно развитые функциональные возможности
- Малая производительность
- Низкая надёжность

Под ***конвертацией*** систем БД понимается процесс переноса данных из существующей (исходной) БД во вновь создаваемую (целевую) БД с последующим переносом прикладных программ в среду целевой СУБД для работы с созданной БД.

БАЗЫ ДАННЫХ ФУНКЦИОНАЛЬНОСТЬ



ОСНОВНЫЕ ФУНКЦИИ СУБД

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

НЕПОСРЕДСТВЕННОЕ УПРАВЛЕНИЕ ДАННЫМИ ВО ВНЕШНЕЙ ПАМЯТИ

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

УПРАВЛЕНИЕ БУФЕРАМИ ОПЕРАТИВНОЙ ПАМЯТИ

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Например, подход Data in Memory в версиях СУБД IBM DB2 для IBM eServer zSeries.

УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

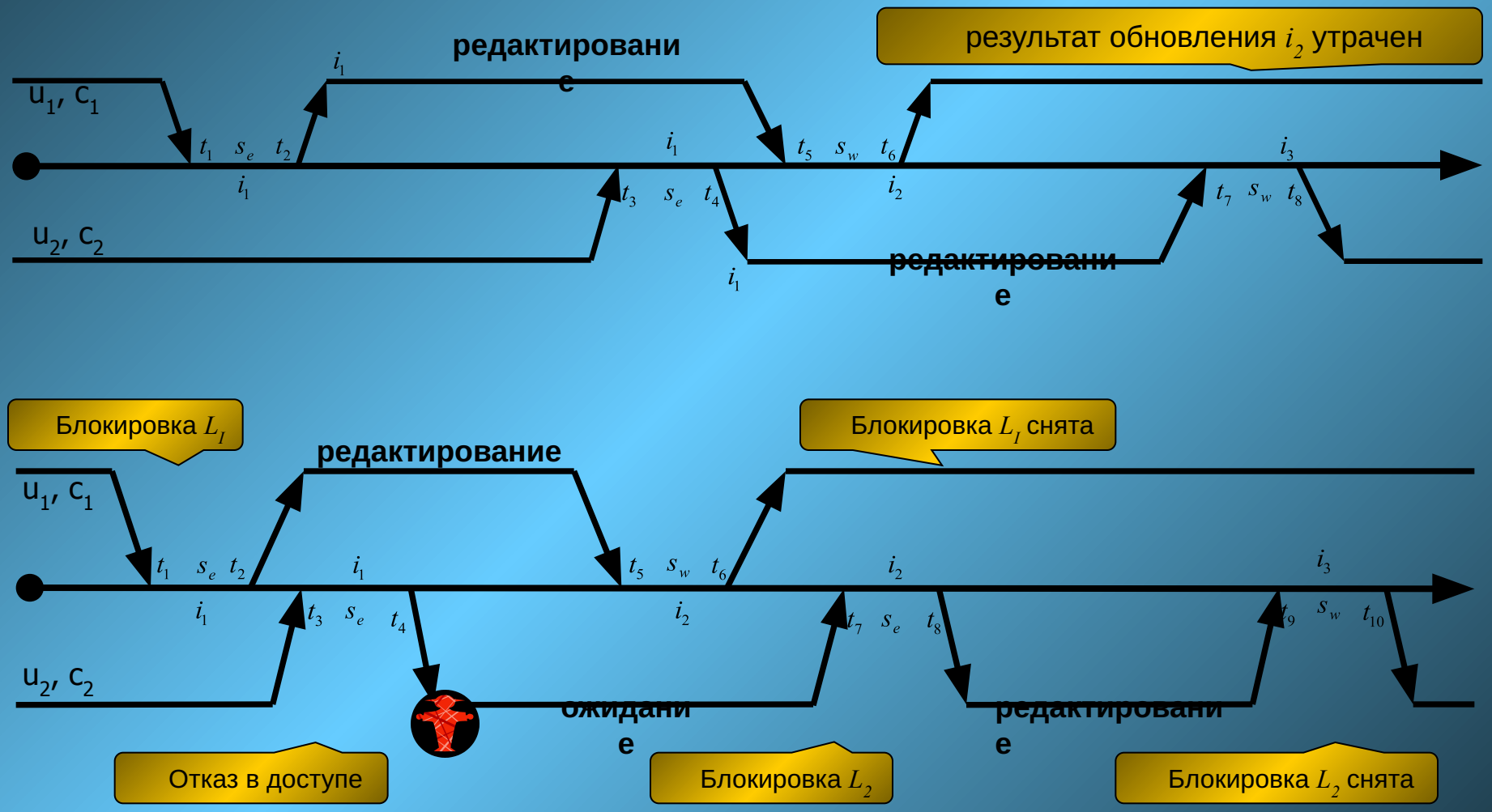
Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить наш пример информационной системы с файлами СОТРУДНИКИ и ОТДЕЛЫ, то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

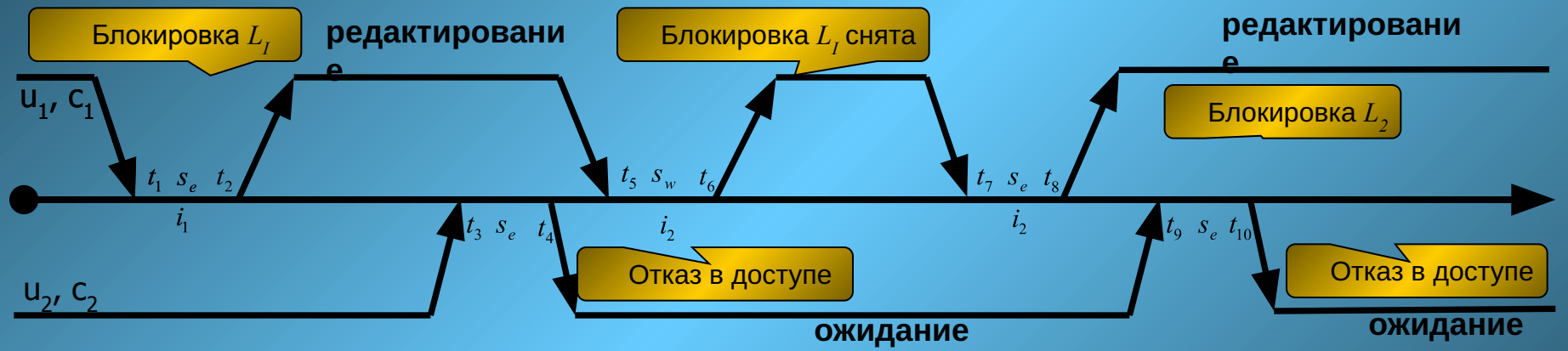
С управлением транзакциями в многопользовательской СУБД связаны важные понятия *сериализации транзакций* и *сериального плана выполнения смеси транзакций*. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ



УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ



ЖУРНАЛИЗАЦИЯ

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

БАЗЫ ДАННЫХ
ЯЗЫКИ СУБД

- **Язык Определения Данных (DDL)**
  - Определяет концептуальную схему, внешнюю схему, и внутреннюю схему, а так же как отношения между ними
  - Язык, используемый для каждого уровня может быть отличным
  - Определения и произведенная информация сохранены в каталоге системы
- **Язык Манипулирования Данными (DML)**
  - Может быть:
    - встроенный в систему язык запросов
    - "автономный" язык запросов
  - Может быть:
    - Процедурный: определяет где и как (навигационный)
    - Декларативный: определяет что

ПОДДЕРЖКА ЯЗЫКОВ БД

Для работы с базами данных используются специальные языки, в целом называемые *языками баз данных*. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - *язык определения схемы* БД *(SDL - Schema Definition Language)* и *язык манипулирования данными (DML - Data Manipulation Language).* SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные. Мы рассмотрим более подробно языки ранних СУБД в следующей лекции.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). В нескольких лекциях этого курса язык SQL будет рассматриваться достаточно подробно, а пока мы перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые при реализации интерфейса SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Более точное описание возможных реализаций этих функций на основе языка SQL будет приведено в лекциях, посвященных языку SQL и его реализации.

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Как можно было понять из первой части этой лекции, функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия (вспомните примеры из первой лекции). Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Применяются достаточно сложные методы оптимизации операторов, которые мы подробно рассмотрим в следующих лекциях. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

ФУНКЦИОНАЛЬНАЯ ХАРАКТЕРИСТИКА СУБД

1. ***Характеристика поддерживаемой модели (или моделей для мультимодельных систем) данных***
   - Класс структур данных
   - Класс операций манипулирования данными
   - Класс ограничений целостности данных
2. ***Средства администратора БД***
   - Ведение словаря-справочника данных
   - Управление представлением БД в среде хранения
   - Сбор и анализ статистики функционирования БД
   - Реорганизация (реструктуризация, реформатизация) БД
   - Контроль целостности и восстановление БД
   - Конвертирование данных и прикладных программ
3. ***Средства разработки приложений***
   - Проектирование БД
   - Интерфейс с языками (системами) программирования
   - Средства генерации (генератор программного кода, форм ввода-вывода, пользовательских интерфейсов)
   - Генерация отчётов
4. ***Средства конечного пользователя***
   - Тип интерфейса конечного пользователя (язык запросов, интерфейсы типа меню и т.п.)
   - Средства обучения, интерактивной помощи (HELP), подсказки, меню
5. ***Интерфейсы с ПС функционального назначения***
6. ***Интерфейсы с другими СУБД***
7. ***Средства работы в сети и создания распределённых БД***