Операционные системы и базы данных

Карпук Анатолий Алексеевич,
доцент кафедры ПОСТ
e-mail: A_Karpuk@mail.ru

Лекция 2.
Введение в базы данных

Вопросы:

1. Информационное обеспечение АИС и организация внутри машинной информационной базы АИС
2. Концепция базы данных
3. Введение в теорию проектирования реляционных баз данных
4. Индексирование и физическая организация данных
5. Основы языка SQL
6. Основы языка Transact-SQL. Хранимые процедуры. Курсоры
7. Взаимодействие с базой данных из языков программирования

1.1. Виды обеспечения АИС

Обеспечивающие подсистемы АИС:

- организационное обеспечение;
- информационное обеспечение;
- техническое обеспечение;
- математическое обеспечение;
- программное обеспечение.

1.2. Информационное обеспечение АИС

Информационное обеспечение **регламентирует потоки и подготовку информации**, **организацию информационной базы**, **систему классификации и кодирования**, определяет **технологический процесс обработки информации** в АИС

1.3. Состав информационного обеспечений АИС

Подсистемы:

- **сбора и передачи информации**;
- **классификации и кодирования**;
- **организации внемашинной информационной базы**;
- **организации внутримашинной информационной базы**

Внемашинную информационную базу образует совокупность всех документированных данных и сообщений используемых в АИС.

Внутримашинную информационную базу образует совокупность всех данных на машинных носителях, сгруппированных по определенному признаку.

1.4. Организация внутримашинной информационной базы АИС

Включает:

- логическую организацию данных;
- физическую организацию данных;
- методы доступа к данным;
- средства ведения данных;
- методы и средства обеспечения целостности и достоверности данных;
- методы и средства защиты данных.

1.5. Логическая и физическая организация данных

Логическая организация данных учитывает лишь те конструкции данных и операции над ними, которые находятся в распоряжении программы, использующей данные.

Физическая организация данных учитывает размещение и связи данных в среде хранения.

Методы доступа к данным – это совокупность соглашений и средств, с помощью которых реализуется заданный вид доступа к физическим записям.

1.6. Ведение ИБ, обеспечение целостности, защита данных

Ведением информационной базы называется деятельность по обновлению, восстановлению и перестройке ее структуры с целью обеспечения целостности, сохранности и эффективности использования данных.

Методы и средства обеспечения целостности и достоверности данных обеспечивают корректность и непротиворечивость данных при выполнении операций над данными.

Защита данных включает организационные, программные и технические методы и средства, направленные на удовлетворение ограничений по доступу к данным.

1.6. Методы организации внутри машинной ИБ АИС

Этап 1: традиционная организация данных

Этап 2: единая информационная база

Этап 3: база данных

1.7. Недостатки методов организации внутримашинной ИБ АИС

Недостатки традиционной организации данных – многократное дублирование данных и невозможность оперативного внесения изменений в группы данных, дублированные в нескольких файлах.

Недостатки метода единой информационной базы – включение в АИС новой ПЗ, не предусмотренной при проектировании информационной базы, может привести к ее изменению и к изменению программ разработанных ПЗ.

2.1. База данных и СУБД

Базой данных (БД) называется совокупность данных, организованная по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ.

Системой управления базами данных (СУБД) называется совокупность программ и языковых средств, предназначенных для управления данными в БД, ведения БД и обеспечения взаимодействия ее с прикладными программами.

2.2. Отличительные признаки БД

- интеграция данных, используемых в различных прикладных задачах (ПЗ);
- учет взаимосвязей между данными;
- минимальная избыточность данных;
- независимость программ ПЗ и глобальной логической организации данных от физической организации данных (физическая независимость данных);
- независимость программ ПЗ от глобальной логической организации данных (логическая независимость данных).

2.3. Модели данных в БД

Модель данных – это совокупность **правил порождения структур данных в БД**, **операций над данными**, а также **ограничений целостности**, определяющих последовательность изменения, допустимые связи и допустимые значения данных.

Концепция БД предусматривает **три уровня описания данных**: **внешний, концептуальный и внутренний**. На каждом уровне используется соответствующая модель данных.

Описание БД в контексте конкретной модели данных называется **схемой БД**.

2.4. Схемы БД

Внешние схемы БД применяется для описания данных в виде, используемом программами ПЗ.

Концептуальная схема БД определяет представление БД, единое для всех ПЗ и не зависящее от используемого в СУБД представления данных в среде хранения и путей доступа к ним.

Внутренняя схема БД определяет представление данных в среде хранения и пути доступа к ним.

Внешние и концептуальная схема относятся к **логической организации данных**, а внутренняя схема – к **физической организации данных**.

2.5. Языковые средства СУБД

При описании БД **внешние, концептуальная и внутренняя схемы БД** описываются на **языке описания данных (ЯОД)**, входящем в состав языковых средств СУБД.

Каждая ПЗ формирует запросы на добавление, удаление, обновление и поиск данных в БД в соответствии со своей внешней схемой на **языке манипулирования данными (ЯМД).**

2.6. Возможности СУБД

СУБД обеспечивают:

- **уменьшение избыточности** хранимых данных за счет минимизации дублирования данных;
- **достоверность** хранимых данных за счет автоматической корректировки всех дублируемых элементов данных;
- **стандартизацию** данных в различных ПЗ;
- **совместное использование** хранимых данных различными ПЗ в мультипрограммном режиме;

2.7. СУБД обеспечивают:

- **физическую независимость данных**, заключающуюся в возможности модификации внутренней схемы БД без изменения концептуальной схемы БД, внешних схем и программ ПЗ;
- **логическую независимость данных**, заключающуюся в возможности модификации концептуальной схемы БД без изменения внешних схем и программ ПЗ;

2.8. СУБД обеспечивают:

- **целостность данных** за счет проверки ограничений целостности при добавлении, удалении и обновлении данных, связанных изменений данных и механизма обработки транзакций;
- **разграничение доступа к данным** за счет защиты данных СУБД и за счет описания во внешней схеме каждой ПЗ только тех данных, которые она использует;
- **сохранность данных** при возникновении сбоев и отказов сервера за счет средств резервного копирования и восстановления БД.

2.9. Жизненный цикл БД и группа АБД

Жизненный цикл базы данных включает этапы **планирования, проектирования и эксплуатации**.

На каждом из этих этапов требуется решить ряд задач, связанных с **интеграцией** пользовательских представлений о данных, **выбором** программных и технических средств ведения БД, **проектированием** концептуальной, внутренней и внешних схем БД, **управлением** средствами обеспечения целостности и достоверности данных и средствами защиты данных, **улучшением** эксплуатационных характеристик СУБД.

Решение этих задач производится **группой администрирования базы данных** (АБД).

2.10. Задачи планирования БД

- **описание предметной области АИС** в виде **информационно-логической (инфологической)** модели;
- **описание создаваемой АИС** в виде информационной модели;
- **распределение программ ПЗ и данных** в сети серверов и ПЭВМ;
- **определение состава хранимых данных** в каждом сервере;
- **выбор метода организации внутримашинной информационной базы** в каждом сервере;
- **выбор СУБД** для каждого сервера.

2.11. Задачи проектирования БД

- **разработка концептуальной и внешних схем БД** для каждого сервера, обеспечивающих заданное время решения ПЗ при наличии минимальной избыточности данных;
- **разработка внутренней схемы БД** для каждого сервера, обеспечивающей требуемую производительность СУБД при решении всех ПЗ при минимальной стоимости хранения данных в БД;
- **выбор средств управления целостностью БД**;
- **выбора режимов и процедур защиты БД** от несанкционированного доступа.

2.12. Методы управления целостностью данных в БД

- проверка ограничений, описанных в концептуальной схеме БД;
- проверка ограничений, не описанных в концептуальной схеме БД;
- управление обработкой транзакций;
- управление параллельной обработкой;
- управление копированием и восстановлением.

2.13. Задачи эксплуатации БД

- **перепроектирование БД** при включении в систему новых ПЗ или изменении требований к существующим ПЗ;
- **оптимизация обработки данных** в ПЗ с целью уменьшения времени выполнения ПЗ за счет оптимизации количества и порядка обращений к СУБД;
- **реорганизация БД** с целью повышения эффективности работы СУБД за счет изменения физического расположения данных на внешних носителях.

2.14. Классификация СУБД

По **месту запуска ядра** СУБД:

- локальные (dBase, FoxPro, Paradox, Access);
- клиент-серверные (Firebird, Interbase, DB2, SQL Server, Sybase, Oracle, PostgreSQL, MySQL, Cache).

По **месту хранения данных**:

- централизованные (локальные);
- распределенные;
- облачные.

По **концептуальной модели данных**:

- иерархические и сетевые;
- реляционные;
- объектно-реляционные;
- объектно-ориентированные;
- многомерные.

3.1. Методология проектирования БД

3.2. Требования к инфологической модели предметной области

Инфологическая модель данных, применяемая для описания предметной области, должна удовлетворять следующим требованиям:

1) средствами модели должны описываться все объекты, предметы, явления предметной области и все отношения между ними, используемые при решении ПЗ;

2) средствами модели должны описываться все ограничения целостности данных, имеющиеся в предметной области;

3) средствами модели должны задаваться все объемные характеристики данных, необходимые для проектирования логической и физической структуры БД;

3.3. Требования к инфологической модели предметной области

4) базис понятий модели должен быть максимально приближенным к базису понятий постановок и алгоритмов ПЗ, а также к лексике специалистов по предметной области;

5) должны существовать методы объединения описаний фрагментов предметной области, соответствующих отдельным ПЗ, в глобальное описание всей предметной области;

6) средствами модели должен описываться процесс решения каждой ПЗ с указанием, какие операции над какими данными и с какой частотой будут выполняться;

7) должны существовать методы и методики отображения глобального описания предметной области в логические, внешние и физические модели данных применяемых СУБД.

3.4. Диаграмма «сущность-связь»

Для построения **инфологической модели предметной области** используются **диаграммы «сущность-связь» (ER-диаграммы)**.

Множество **допустимых структурных компонентов** модели данных «сущность-связь»: **сущность, связь между сущностями, атрибут сущности, первичный ключ сущности, внешний ключ сущности, функциональные зависимости (ФЗ) между атрибутами сущности, состав многозначного атрибута сущности, ФЗ между элементами многозначного атрибута сущности**.

3.5. Сущности

Сущность – это множество **реальных** или **абстрактных объектов** (людей, предметов, документов и т.п.), **обладающих общими атрибутами или характеристиками**. Любой объект системы может быть представлен только одной сущностью, которая должна быть уникально идентифицирована. Именование сущности осуществляется с помощью существительного в единственном числе. При этом имя сущности должно отражать **тип** или **класс** объекта, а не его **конкретный экземпляр.**

3.6. Связи между сущностями

Сущности не существуют отдельно друг от друга. Между ними имеются **отношения**, которые должны быть отражены в инфологической модели предметной области в виде **связей**.

Связь представляет собой **соединение двух сущностей**. Связь описывается **вербальными фразами** в виде глаголов в двух направлениях. Каждая связь должна иметь свое **уникальное имя связи**.

3.7. Классификация связей

Степень связи (Cardinality) – число экземпляров сущности, участвующих с каждой стороны связи. Могут быть связи «**один к одному**», «**один ко многим**», «**многие к одному**», «**многие ко многим**».

Связь «один ко многим» между **родительской** («один») и **подчиненной** («многие») сущностями **идентифицирующая**, если у любого экземпляра подчиненной сущности имеется экземпляр родительской сущности, иначе связь **не идентифицирующая**. Связь «один к одному» всегда идентифицирующая.

3.8. Атрибуты сущностей

Атрибут сущности – свойство сущности, имеющее **имя, область допустимых значений** (тип и формат) и **признак обязательности** атрибута. Значением атрибута может быть **неделимый элемент данных, вектор из элементов данных, структура из элементов данных, повторяющаяся группа из элементов данных, векторов или структур**.

Сущность должна иметь обязательный **атрибут или комбинацию обязательных атрибутов**, чьи значения **однозначно определяют** каждый экземпляр сущности. Эти атрибуты образуют **первичный ключ сущности (Primary Key, PK)**. Если сущность имеет несколько таких подмножеств атрибутов, то одно из них объявляется **первичным ключом**, а каждое из остальных называется **возможным** или **альтернативным ключом**.

3.9. Внешние ключи сущностей

Если между двумя сущностями имеется **связь «один к одному» или «один ко многим» («многие к одному»)**, то **атрибуты первичного ключа** родительской сущности **наследуются** в качестве **атрибутов подчиненной сущности**. Эти атрибуты называются **внешним ключом (Foreign Key, FK)**.

Если связь между сущностями **идентифицирующая**, то атрибуты **внешнего ключа** входят в состав **первичного ключа подчиненной сущности**, либо входят в состав **альтернативного ключа** этой сущности.

Если связь между сущностями **не идентифицирующая**, то атрибуты **внешнего ключа** входят в состав **не ключевых атрибутов подчиненной сущности**.

3.10. ФЗ между атрибутами сущностей

Пусть **X** – некоторое **подмножество атрибутов сущности**, **Y – атрибут сущности**, не входящий в X.

Множество атрибутов **X функционально определяет** атрибут **Y** (атрибут **Y функционально зависит** от атрибутов **X**), обозначается **X → Y**, если **любой комбинации** значений атрибутов из **X** соответствует **одно значение** атрибута **Y**. Другими словами, для любых двух экземпляров сущности из **равенства значений** атрибутов из **X** следует **равенство значений** атрибута **Y**.

В **левой части ФЗ** содержится **один или более** атрибутов. Если в **правой части ФЗ** записано более одного атрибута, то **каждый из этих атрибутов функционально зависит** от левой части ФЗ.

3.11. Многозначные атрибуты сущностей

Если значением **многозначного атрибута** является **вектор или структура**, то каждому **элементу** вектора или структуры ставится в соответствие **атомарный атрибут**. В инфологической модели ПрО следует задать **имена, типы и форматы этих атомарных атрибутов**.

Если значением **многозначного атрибута** является **повторяющаяся группа** из элементов данных, векторов или структур, то в инфологической модели ПрО следует задать **имена, типы и форматы атомарных атрибутов, входящих в повторяющуюся группу**. Кроме того, среди атомарных атрибутов повторяющейся группы следует выделить **ключевые и не ключевые атрибуты**.

3.12. Нормализация сущностей и таблиц БД

Таблица находится в **1НФ**, если значения **всех ее атрибутов являются атомарными** (неделимыми с точки зрения СУБД).

Таблица находится во **2НФ**, если она находится в **1НФ**, и все не ключевые атрибуты функционально **полно** зависят от **первичного ключа** (не может быть зависимости от части ключа).

Таблица находится в **3НФ**, если она находится во **2НФ**, и любой **не ключевой атрибут** не зависит от **других не ключевых атрибутов** (не должно быть зависимости между не ключевыми атрибутами)

4.1. Индексы в БД

Индекс представляют собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы на основе значений одного или более ее столбцов .

Наличие индекса может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания.

Хотя индекс и связан с конкретным столбцом (или столбцами) таблицы, он является **самостоятельным объектом базы данных.**

Поскольку индексы должны обновляться системой при каждом внесении изменений в их базовую таблицу, они создают **дополнительную нагрузку на систему**.

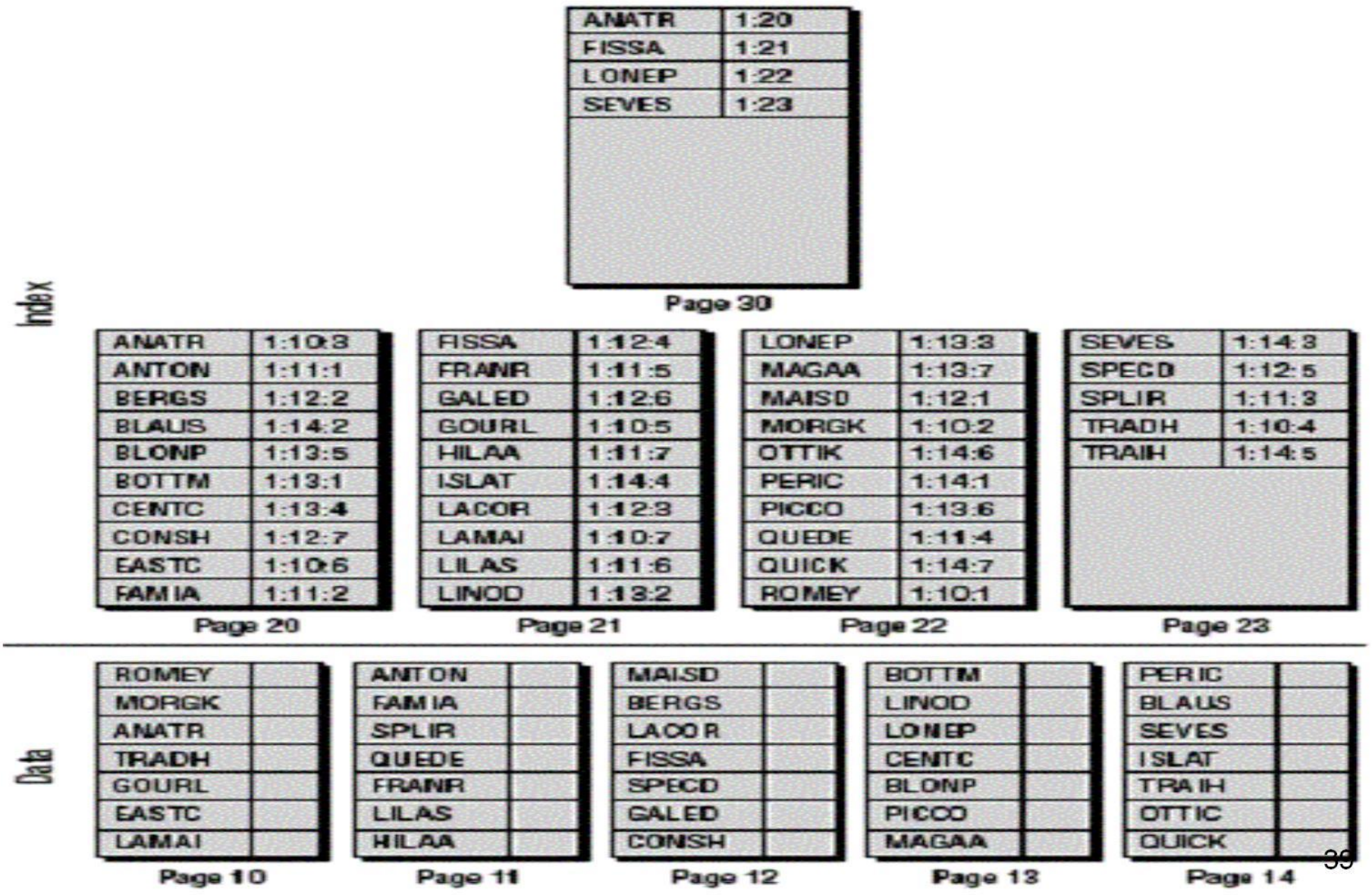
4.2. Основные типы индексов

Некластерный индекс – не перестраивают физическую структуру таблицы, а лишь организуют ссылки на соответствующие строки. Использует специальные указатели, включающие в себя: информацию об идентификационном номере файла, в котором хранится строка; идентификационный номер страницы; номер искомой строки на соответствующей странице.

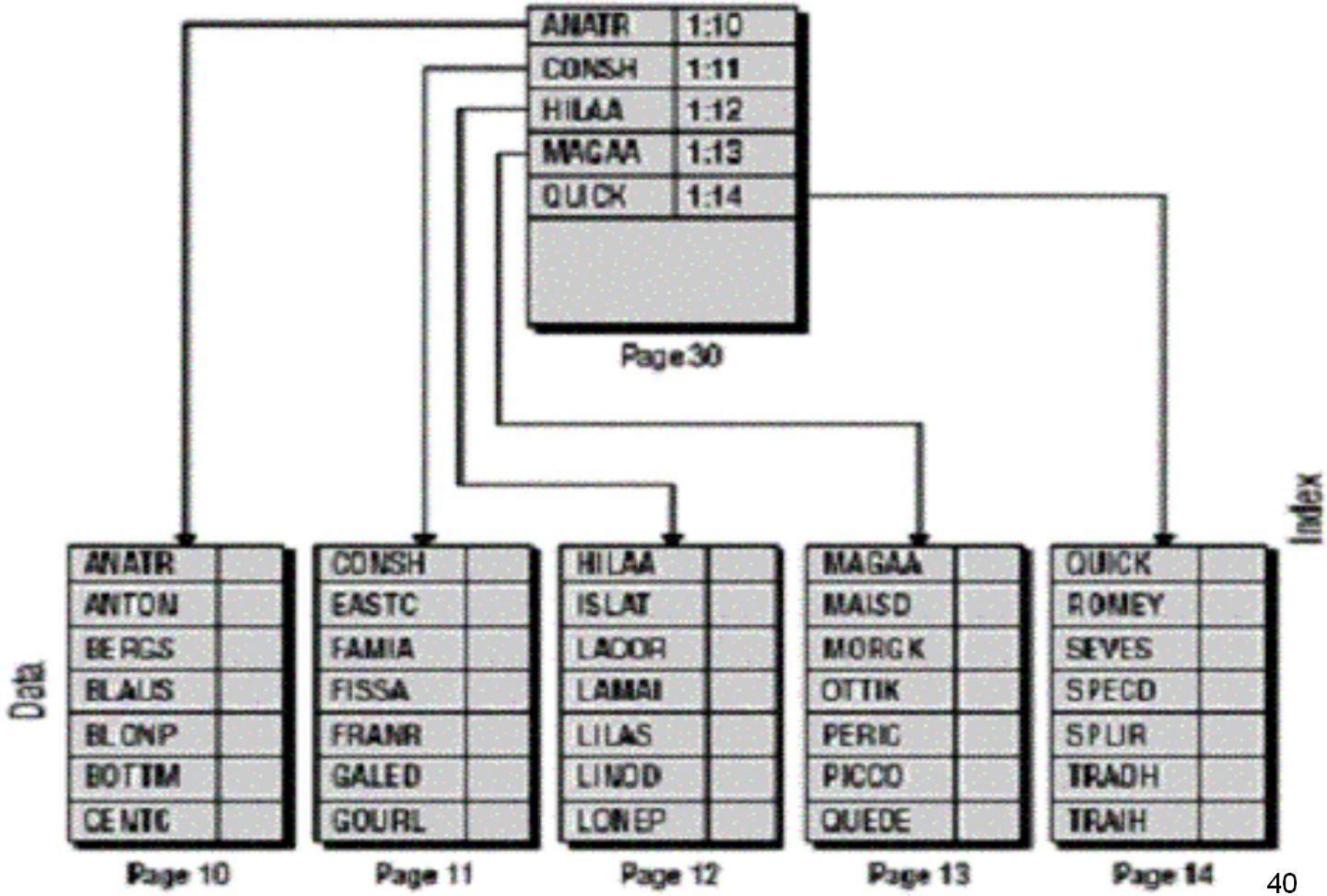
Кластерный индекс – при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса.

Уникальный индекс – сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в таблице появились два одинаковых значения индекса.

4.3. Пример некластерного индекса



4.4. Пример кластерного индекса



4.5. Физическая организация данных в Microsoft SQL Server

На физическом уровне БД в Microsoft SQL Server представляется набором файлов операционной системы. Данные и сведения журналов транзакций размещаются в разных файлах. Каждый файл используется только одной БД. Файловые группы представляют собой именованные коллекции файлов.

Базы данных SQL Server содержат файлы трех типов:

- **Первичные файлы данных**. В каждой БД имеется один первичный файл данных с расширением MDF.
- **Вторичные файлы данных**. Все остальные файлы БД с данными. Используется расширение NDF.
- **Файлы журналов**. Содержат сведения журналов, используемые для восстановления БД.

4.6. Страницы и экстенты

Страница – основная единица хранения данных и обмена между внешней и оперативной памятью в SQL Server. **Размер страницы** – 8 кБ. Страница имеет **заголовок** в 96 байт, содержащий номер, тип и объем свободного места страницы. Всего 8 типов страниц.

Экстент – это коллекция, состоящая из восьми физически непрерывных страниц или 64 Кб. Все страницы хранятся в экстентах. В одном МБ БД SQL Server содержится 16 экстентов (128 страниц). Имеется 2 типа экстентов:

- однородные экстенты – все 8 страниц используются для одной таблицы или одного индекса;
- смешанные экстенты – страницы экстента используются для различных таблиц и индексов (одна страница – для 1 таблицы или 1 индекса).

4.7. Типы страниц в SQL Server

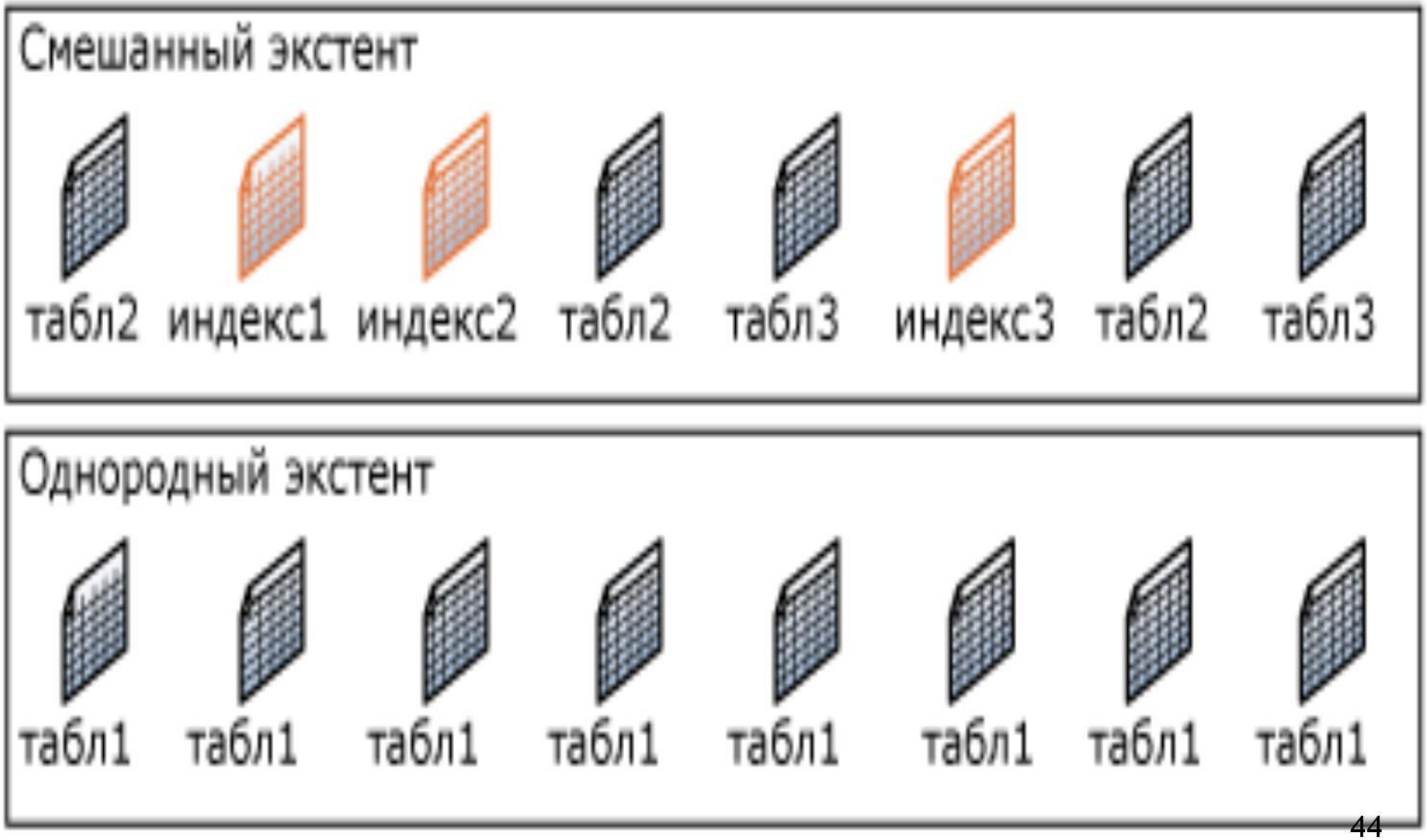
Страницы, относящиеся к хранению и поиску данных:

- страницы данных;
- индексные страницы;
- текстовые страницы;
- страницы журнала транзакций;

Страницы размещения:

- карты распределения блоков (основная и вторичная);
- карты свободного пространства;
- индексные карты размещения.

4.8. Пример смешанного и однородного экстентов



4.9. Секции

По умолчанию таблица или индекс имеет одну **секцию**, которая содержит все страницы таблицы или индекса. Секция располагается в одной файловой группе.

Если таблица или индекс используют несколько секций, данные секционируются горизонтально, так что группы строк сопоставляются отдельным секциям, основываясь на **указанном столбце**. Секции могут храниться в одной или нескольких файловых группах в БД. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений. Секция состоит из фрагментов одного или нескольких файлов.

4.10. Физическая организация данных в СУБД Oracle

Файлы – это файлы операционной системы, выделенные для хранения базы данных.

Табличная область (TABLESPACE, табличное пространство) – область памяти, предназначенная для хранения всех объектов БД. Табличная область имеет имя и занимает один или более файлов операционной системы. Создается командой CREATE TABLESPACE.

Сегмент (SEGMENT) – область памяти, занимаемая данными одного объекта БД. Имя совпадает с именем объекта.

Экстент (EXTENT) – непрерывная область памяти, относящаяся к одному сегменту.

Блок (BLOCK) – область памяти, которая считывается и записывается на диск за одно физическое чтение.

5.1. Формы операторов SQL

SQL реализуется в следующих формах: **интерактивный SQL, встроенный SQL,** который может быть **статическим или динамическим**.

Интерактивный SQL позволяет конечному пользователю в интерактивном режиме выполнять SQL-операторы.

Встроенный SQL позволяет включать операторы SQL в код программы на языке программирования (например, C++).

Операторы **статического SQL** определены в момент компиляции программы.

Динамический SQL позволяет формировать операторы SQL во время выполнения программы.

.

5.2. Группы операторов SQL

SQL определяет: **операторы языка**, называемые иногда командами языка SQL; **типы данных**; **набор встроенных функций**.

Операторы SQL разбиваются на **язык определения данных DDL** (Data Definition Language), **язык манипулирования данными DML** (Data Manipulation Language) и **язык управления данными DCL** (Data Control Language).

Язык определения данных включает операторы, управляющие объектами базы данных (таблицы индексы, представления и т. д.).

Язык манипулирования данными включает операторы, управляющие содержанием таблиц базы данных и извлекающими информацию из этих таблиц.

Язык управления данными служит для управления правами доступа к данным и работой в многопользовательской среде.

5.3. Операторы DDL

CREATE DATABASE	(создать базу данных)
CREATE TABLE	(создать таблицу)
CREATE VIEW	(создать представление)
CREATE INDEX	(создать индекс)
CREATE TRIGGER	(создать триггер)
CREATE PROCEDURE	(создать хранимую процедуру)
ALTER DATABASE	(модифицировать базу данных)
ALTER TABLE	(модифицировать таблицу)
ALTER VIEW	(модифицировать представление)
ALTER INDEX	(модифицировать индекс)
ALTER TRIGGER	(модифицировать триггер)
ALTER PROCEDURE	(модифицировать хранимую процедуру)
DROP DATABASE	(удалить базу данных)
DROP TABLE	(удалить таблицу)
DROP VIEW	(удалить представление)
DROP INDEX	(удалить индекс)
DROP TRIGGER	(удалить триггер)
DROP PROCEDURE	(удалить хранимую процедуру)

5.4. Операторы DML и DCL

INSERT	(вставить)
SELECT	(выбрать)
UPDATE	(обновить)
DELETE	(удалить)

GRANT	(дать права)
REVOKE	(забрать права)

5.5. Оператор CREATE TABLE

CREATE TABLE имя_таблицы
({<определение_столбца>|
<определение_ограничения_таблицы>}
[,…,{<определение_столбца>|
<определение_ограничения_таблицы >}])

Определение_столбца:
<Имя_столбца> <тип_данных>
[<ограничение_столбца>]
[,…,<ограничение_столбца>]

5.6. Ограничения столбца

<ограничение_столбца>::=
[CONSTRAINT <имя_ограничения >]
{[DEFAULT <выражение>]
| [NULL | NOT NULL]
| [PRIMARY KEY | UNIQUE]
| [FOREIGN KEY
REFERENCES
<имя_главной_таблицы>[(<имя_столбца> [,…,n])]
[ON DELETE {CASCADE | NO ACTION | SET NULL |
SET DEFAULT }]
[ON UPDATE {CASCADE | NO ACTION | SET NULL |
SET DEFAULT }]]
| [CHECK (<логическое_выражение>)]}

5.7. Ограничения столбца

Ограничения для столбца могут указываться следующими фразами:

- **NOT NULL** - в любой добавляемой или изменяемой строке столбец всегда должен иметь значение, отличное от NULL;
- **UNIQUE** - все значения столбца должны быть уникальны;
- **PRIMARY KEY** - устанавливает один столбец как первичный ключ и одновременно подразумевает, что все значения столбца будут уникальны;
- **CHECK** (condition) - указываемое в скобках условие использует для сравнения значение столбца и возвращает TRUE, FALSE или UNKNOWN. Если при попытке выполнения SQL-оператора возвращаемое значение равно FALSE, то оператор выполнен не будет;
- **REFERENCES** table (fields_list) - ограничение требует совпадения значений столбцов данной таблицы с указанными столбцами родительской таблицы.

5.8. Ограничения на уровне таблицы

Ограничения могут указываться следующими фразами:

- **CHECK** (condition) - указываемое в скобках условие использует для сравнения значение столбца и возвращает TRUE, FALSE или UNKNOWN. Если при попытке выполнения SQL-оператора возвращаемое значение равно FALSE, то оператор выполнен не будет;
- **FOREIGN KEY** (fields_list) - это ограничение по внешнему ключу аналогично ограничению REFERENCES для столбцов и гарантирует, что все значения, указанные во внешнем ключе будут соответствовать значениям родительского ключа, обеспечивая ссылочную целостность. Типы данных столбцов, используемых в этом ограничении, должны совпадать.

5.9. Оператор ALTER TABLE

- добавить в таблицу определение нового столбца;
- удалить столбец из таблицы;
- изменить значение по умолчанию для какого-либо столбца;
- добавить или удалить первичный ключ таблицы;
- добавить или удалить внешний ключ таблицы;
- добавить или удалить условие уникальности;
- добавить или удалить условие на значение.

5.10. Оператор SELECT

Условия выборки данных в предложении **WHERE** представляют собой выражения над значениями столбцов таблиц и представлений, соединенные **операторами сравнения и логическими операторами**. В выражениях используются арифметические операторы, строковые операторы, математические и строковые функции, функции работы с датой и временем, агрегатные функции.

Могут использоваться **операторы сравнения** = , < , > , <= , >= , != или < >, !< , !>.

Логические операторы проверяют истинность заданного условия. Логические операторы возвращают значение типа Boolean: TRUE, FALSE или UNKNOWN.

5.11. Оператор SELECT

Предложение **FROM** table_list содержит список таблиц и представлений, из которых будут получены данные результирующего набора. Предложение FROM может также содержать спецификации соединения. Они задают определенный путь для СУБД, который должен использоваться при навигации от одной таблицы к другой.

Предложение **WHERE** search_conditions является фильтром, задающим условия, которым должна соответствовать каждая строка в исходных таблицах и представлениях, чтобы быть выбранной оператором SELECT.

Предложение **GROUP BY** group_by_list делит результирующий набор на группы на основании значений в столбцах group_by_list.

5.12. Оператор SELECT

Предложение **HAVING** search_conditions является дополнительным фильтром, который применяется к результирующему набору. Предложения HAVING обычно используются с предложением GROUP BY для задания условий отбора сформированных групп.

Предложение **ORDER BY** order_list определяет порядок, в котором отсортированы строки в результирующем наборе.

Параметр order_list указывает столбцы результирующего набора, которые составляют список сортировки. Ключевые слова ASC и DESC используются для указания того, в какой последовательности сортируются строки – по возрастанию или по убыванию.

5.13. Оператор SELECT

Условия выборки данных в предложении **WHERE** представляют собой выражения над значениями столбцов таблиц и представлений, соединенные **операторами сравнения и логическими операторами**. В выражениях используются арифметические операторы, строковые операторы, математические и строковые функции, функции работы с датой и временем, агрегатные функции.

Могут использоваться **операторы сравнения** = , < , > , <= , >= , != или < >, !< , !>.

Логические операторы проверяют истинность заданного условия. Логические операторы возвращают значение типа Boolean: TRUE, FALSE или UNKNOWN.

5.14. Агрегатные функции

Функция	Описание
COUNT(*)	Возвращает количество строк источника записей.
COUNT(<имя поля>)	Возвращает количество значений в указанном столбце.
SUM(<имя поля>)	Возвращает сумму значений в указанном столбце.
AVG(<имя поля>)	Возвращает среднее значение в указанном столбце.
MIN(<имя поля>)	Возвращает минимальное значение в указанном столбце.
MAX(<имя поля>)	Возвращает максимальное значение в указанном столбце.

5.15. Связывание таблиц в фразе FROM

<связка_таблиц> ::=

<левая_таблица><тип_связывания><правая_таблица>
ON <условие_связывания>

<тип_связывания> ::=

[INNER | {{LEFT | RIGHT | FULL } [OUTER] }] **JOIN**

SELECT Наименование, Семестр, Количество_часов

FROM Учебный_план INNER **JOIN** Дисциплины

ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина

WHERE Количество_часов > 60

5.16. Оператор INSERT

INSERT [INTO]
<имя_таблицы>
{ [(<список_колонок>)]
{ VALUES
({ DEFAULT | NULL | <выражение> } [, ..., n])
| <результирующая_таблица>
}
}
| DEFAULT VALUES

5.17. Операторы UPDATE и DELETE

UPDATE <имя_таблицы>

SET { <имя_колонки> = { <выражение> | DEFAULT | NULL }}[,...,n]

{ [FROM { <имя_исходной_таблицы> } [,...,n]]

[WHERE <условие_отбора>] }

DELETE <имя_таблицы>

[WHERE <условие_отбора>]

6.1. Состав языка Transact-SQL

Язык Transact-SQL включает следующие средства:

- данные различного типа из баз данных и переменных;
- константы, стандартные и ограниченные идентификаторы;
- арифметические и логические выражения, включающие следующие операнды: константы, переменные, имена столбцов таблиц, функции, подзапросы и условные выражения, а также выражения, взятые в круглые скобки;
- SQL – команды для создания, изменения и удаления баз данных и их объектов, а также для определения запросов на ввод, обработку и извлечение данных;
- управляющие программные структуры, определяющие условия и порядок выполнения команд в заданной последовательности или пакете команд;
- встроенные (системные) и определяемые пользователем функции;
- встроенные (системные) и определяемые пользователем хранимые процедуры.

6.2. Локальные переменные

Объявление локальных переменных:

DECLARE {@ имя локальной переменной тип данных}[,…n]

Знак @ является признаком имени локальной переменной. Этот же знак используется для определения имен параметров функций и хранимых процедур. Часть синтаксиса [,…n] означает повторение синтаксической конструкции, взятой в фигурные скобки:

DECLARE @Ivar int,

DECLARE @IBit bit

Значения переменным можно присвоить с помощью команд SET и SELECT. Командой SET можно присвоить значение только одной переменной:

SET @Ivar = 5

SET @IBit = 0

6.3. Управляющие структуры

BEGIN…END – для создания блока последовательных команд.

IF…ELSE – для определения условия выбора команды или блока.

CASE…END – для реализации условного выражения с несколькими альтернативами: CASE…WHEN…WHEN…ELSE…END.

COALESCE – для обработки совместных выражений (возвращают первое непустое значение в списке);

WHILE…BREAK…CONTINUE – для организации и управления циклически выполняемых команд.

6.4. Понятие хранимой процедуры

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. Выполнение хранимых процедур вместо отдельных операторов SQL дает пользователю следующие преимущества:

- необходимые операторы уже содержатся в базе данных;
- все они прошли этап синтаксического анализа и находятся в исполняемом формате, SQL Server генерирует для них план исполнения, выполняет их оптимизацию и компиляцию;
- хранимые процедуры поддерживают модульное программирование, позволяют разбивать задачи на более мелкие и удобные части;
- хранимые процедуры могут вызывать другие хранимые процедуры и функции;
- хранимые процедуры могут быть вызваны из прикладных программ других типов;
- хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;
- хранимые процедуры позволяют уменьшить размер запроса, посылаемого от клиента на сервер, т.е. нагрузку на сеть.

6.5. Оператор создания и изменения хранимой процедуры

{CREATE | ALTER } [PROCEDURE] имя_процедуры
[;номер]
[{@имя_параметра тип_данных } [VARYING]
[=default] [OUTPUT]] [,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION]
AS
sql_оператор [...n]

6.6. Вызов хранимой процедуры

[[EXEC [UTE] имя_процедуры [;номер]

[[@имя_параметра=]{значение | @имя_переменной}

[OUTPUT]|[DEFAULT]][,...n]

Если вызов хранимой процедуры не является единственной командой в пакете, то **присутствие команды EXECUTE обязательно**. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.

Использование **ключевого слова OUTPUT** при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом OUTPUT.

Когда при вызове процедуры для параметра указывается **ключевое слово DEFAULT**, то будет использовано значение по умолчанию.

При вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

6.7. Понятие курсора

Курсор в SQL – это область в памяти базы данных, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к базе данных, в памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой курсора. Указанная область в памяти поименована и доступна для прикладных программ.

Обычно курсоры используются для **выбора из базы данных некоторого подмножества хранимых данных**. В каждый момент времени прикладной программой может быть проверена одна строка курсора. Курсоры часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.

6.8. Работа с курсором

При работе с курсором можно выделить следующие основные действия:

- **создание или объявление** курсора;
- **открытие** курсора, т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- **выборка** из курсора и **изменение** с его помощью строк данных;
- **закрытие** курсора, после чего он становится недоступным для пользовательских программ;
- **освобождение** курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

6.9. Управление курсором

Управление курсором реализуется путем выполнения следующих операторов:

- **DECLARE** – создание или объявление курсора;
- **OPEN** – открытие курсора, т.е. наполнение его данными;
- **FETCH** – выборка из курсора и изменение строк данных с помощью курсора;
- **CLOSE** – закрытие курсора;
- **DEALLOCATE** – освобождение курсора, т.е. удаление курсора как объекта.

7.1. Способы использования SQL в прикладных программах

Внедренные SQL-операторы. Отдельные SQL-операторы внедряются прямо в исходный текст программы и смешиваются с операторами базового языка. Специальные программы-предкомпиляторы преобразуют исходный текст с целью замены SQL-операторов соответствующими вызовами подпрограмм СУБД, затем он компилируется и собирается обычным способом.

Использование **прикладного интерфейса программирования (API)**. Альтернативный вариант состоит в предоставлении программисту стандартного набора функций, к которым можно обращаться из создаваемых им программ. Конкретный вариант API может предоставлять тот же набор функциональных возможностей, который существует при подключении встроенных операторов, однако при этом устраняется необходимость предкомпилирования исходного текста. Кроме того, в этом случае используется более понятный интерфейс и созданный программный текст более удобен с точки зрения его сопровождения.

7.2. Интерфейс ODBC

В **интерфейс ODBC** включены следующие элементы:

библиотека функций, вызов которых позволяет приложению подключаться к базе данных, выполнять SQL-операторы и извлекать информацию из результирующих наборов данных ;

стандартный метод подключения и регистрации в СУБД;

стандартное представление для данных различных типов;

стандартный набор кодов ошибок;

типовой синтаксис SQL-операторов, построенный на использовании спецификации X/Open и ISO CGI.

7.3. Архитектура ODBC

Архитектура ODBC включает четыре элемента:

1. **Приложение**. Выполняет обработку данных и вызов функций библиотеки ODBC для отправки SQL-операторов СУБД и выборки возвращаемых данных.
2. **Менеджер драйверов**. Выполняет загрузку драйверов по требованию приложения.
3. **Драйверы и агенты баз данных**. Обрабатывают вызовы функций ODBC и направляют SQL-запросы к конкретным источникам данных, а также возвращают полученные результаты приложению.
4. **Источники данных**. Содержат данные, доступ к которым необходим пользователю приложения. Данные хранятся в БД под управление целевой СУБД.

7.4. Протокол JDBC

Взаимодействие Java-программы с внешним сервером баз данных осуществляется посредством специализированного протокола, отвечающего за совместимость Java с базами данных (**Java Database Connectivity, JDBC)**. Он построен на принципах интерфейса ODBC и применяется для стандартизации Java-кода при организации доступа к различным СУБД. Созданный вслед за спецификацией ODBC, пакет JDBC стал одним из методов доступа к реляционным СУБД из Java-программ. Протокол JDBC, по сути, является посредником Java-кода и драйвером ODBC. Этап подключения к базе данных включает **загрузку драйвера** и **создание соединения**.

7.5. Драйвер JDBC

Конкретная база данных доступна с помощью одного или нескольких драйверов. Составная часть JDBC - драйвер для доступа из JDBC к источникам данных ODBC. Этот драйвер называется программой сопряжения JDBC-ODBC и реализован в виде **класса JdbcOdbc.class**. Программа сопряжения представляет собой надстройку над JDBC. На внутреннем уровне этот драйвер отображает методы Java в вызовы ODBC и тем самым взаимодействует с любым ODBC-драйвером. Драйвер JDBC-ODBC является мост-драйвером, т.к. создает мост между JDBC и другим интерфейсом уровня обращения (Call Level Interface, CLI). Он обрабатывает обращения JDBC и, в свою очередь, вызывает функции ODBC, которые передают запросы SQL источнику данных ODBC.

7.6. Классы JDBC для работы с БД

Класс DriverManager – загружает, регистрирует драйвер, ведет состояние драйвера.

Класс Connection (соединение) - управляет всеми аспектами установки связи. Connection - это класс, ведущий к драйверу СУБД через источник данных.

Объект **класса Statement** создается методом createStatement() из объекта Connection для SQL-операторов, выполняемых один раз.

Объект **класса ResultSet**, представляющий набор результатов, фактически является табличным набором данных, т.е. состоит из строк данных, организованных в унифицированные столбцы.

7.7. Функции технологии ADO

Технология Active Data Objects (ADO) - это программное расширение технологии активных серверных страниц ASP, реализованное с целью организации подключений к базам данных. В технологии ADO поддерживаются следующие основные функции:

- независимо создаваемые объекты;
- поддержка хранимых процедур с входными, выходными и возвращаемыми параметрами;
- курсоры различных типов (включая возможность поддержки разных специальных курсоров конечных пользователей);
- пакетное обновление;
- поддержка ограничений для числа возвращаемых строк или других параметров запроса;
- поддержка нескольких наборов данных, возвращаемых хранимыми процедурами или пакетными операторами.

7.8. Применение технологии ADO

Преимуществами технологии ADO является простота использования, высокая скорость, небольшие потребности в оперативной памяти и незначительные затраты дисковой памяти. Объектная модель ADO определяет набор (коллекцию) программируемых объектов, которые могут применяться с Visual Basic, Visual C++, VBScript, VBA.

ADO содержит семь классов объектов, которые инкапсулируют в себе большинство операций с базой данных - **Connection** (соединение), **Command** (команда), **Parameter** (параметр), **Recordset** (набор данных), **Field** (поле), **Property** (свойство) и **Error** (ошибка), - а также четыре набора объектов (коллекции) - **Fields** (поля), **Properties** (свойства), **Parameters** (параметры) и **Errors** (ошибки).