Введение
в базы данных

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

Лекция 3.
Физическая организация данных в СУБД. Процедурное расширение языка SQL

Вопросы:

1. Механизмы физической организации данных
2. Структура хранимых записей
3. Внутренний идентификатор записи
4. Управление пространством памяти
5. Виды адресации хранимых записей
6. Индексы в базе данных
7. Физическая организация данных в СУБД Microsoft SQL Server
8. Основные физические структуры СУБД Oracle
9. Сравнение размеров страниц, блоков и экстентов СУБД Microsoft SQL Server и Oracle
10. Состав языка Transact-SQL
11. Виды констант языка Transact-SQL
12. Идентификаторы в языке Transact-SQL
13. Именование объектов базы данных
14. Локальные переменные
15. Операторы выражений
16. Управляющие структуры
17. Специальные логические операторы
18. Хранимые процедуры
19. Хранимые функции
20. Триггеры

1. Механизмы физической организации данных

При добавлении записи:

- поиск свободного места для размещения новой записи в пространстве памяти;
- выделение необходимого объема памяти под запись;
- размещение записи в отведенном месте памяти;
- формирование связей с другими записями (зависит от модели данных).

При поиске записи:

- поиск места размещения записи в пространстве памяти по заданным значениям атрибутов;
- выборка записи для обработки в оперативную память (в буфер данных).

При изменении атрибутов записи:

- поиск записи и считывание её в ОП;
- изменение значений атрибута (атрибутов) записи;
- сохранение записи на диск.

При удалении записи:

- удаление записи с освобождением памяти (*физическое удаление*) или без освобождения (*логическое удаление*);
- разрушение связей с другими записями (механизм зависит от модели данных).

2. Структура хранимых записей

Единица хранения данных в БД – **хранимая запись**.

Хранимая запись состоит из двух частей:

1. *Служебная часть*. Используется для идентификации записи, задания её типа, хранения признака логического удаления, для кодирования значений элементов записи, для установления структурных ассоциаций между записями и проч. Никакие пользовательские программы не имеют доступа к служебной части хранимой записи.
2. *Информационная часть*. Содержит значения элементов данных. Поля хранимой записи могут иметь *фиксированную* или *переменную длину*. Хранение полей переменной длины осуществляется одним из двух способов: размещение полей через разделитель или хранение размера значения поля. Наличие полей переменной длины позволяет не хранить незначащие символы и снижает затраты памяти на хранение данных; но при этом увеличивается время на извлечение записи.

3. Внутренний идентификатор записи

Каждой хранимой записи БД система присваивает внутренний идентификатор, называемый (по стандарту CODASYL) **ключом базы данных** (КБД).

В Oracle используется термин *идентификатор строки*, RowID.

Значение КБД формируется системой при размещении записи и содержит информацию, позволяющую однозначно определить место размещения записи (преобразовать значение КБД в адрес записи).

Примеры:

1) Формат DBF: 1 таблица – 1 файл, записи фиксированной длины – в качестве КБД выступает последовательный номер записи в файле (относительная адресация).

2) СУБД Oracle – совокупность номера экстента, блока и номера строки в блоке (относительная адресация).

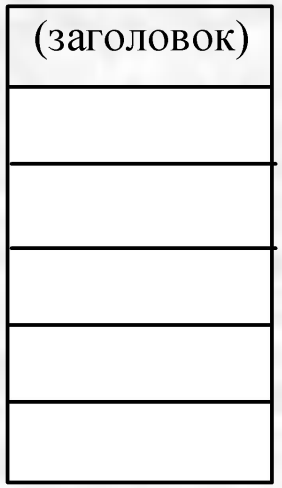
3) Абсолютный адрес в памяти (СУБД Adabas).

4. Управление пространством памяти

Для обеспечения естественной структуризации хранимых данных, более эффективного управления ресурсами и/или для технологического удобства всё пространство памяти БД обычно разделяется на части (области, сегменты и др.). В каждой *области памяти*, как правило, хранятся данные одного объекта БД (одной таблицы). Сведения о месте расположения данных таблицы (ссылка на область хранения) СУБД хранит в словаре-справочнике данных (ССД). Области разбиваются на пронумерованные *страницы (блоки)* фиксированного размера. В большинстве систем обработку данных на уровне страниц ведёт операционная система (ОС), а обработку записей внутри страницы обеспечивает только СУБД.

Страницы представляются в среде ОС блоками внешней памяти или секторами, доступ к которым осуществляется за одно обращение. Некоторые СУБД позволяют управлять размером страницы (блока) для базы данных.

Структура страницы памяти



4. Управление свободным пространством памяти

Способы управления свободным пространством памяти на страницах:

- ведение списков свободных участков;
- динамическая реорганизация страниц.

При **динамической реорганизации страниц** записи БД плотно размещаются вслед за заголовком страницы, а после них расположен свободный участок. Смещение начала свободного участка хранится в заголовке страницы.

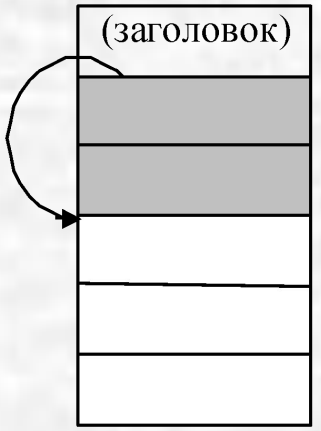
Достоинство такого подхода – отсутствие фрагментации.

Недостатки:

- ✔ Адрес записи может быть определён с точностью до адреса страницы, т.к. внутри страницы запись может перемещаться.
- ✔ Поиск места размещения новой записи может занять много времени. Система будет читать страницы одну за другой до тех пор, пока не найдёт странницу, на которой достаточно места для размещения новой записи.

Ведение инвентарных страниц.

динамическая реорганизация страниц



4. Ведение списка свободных участков памяти

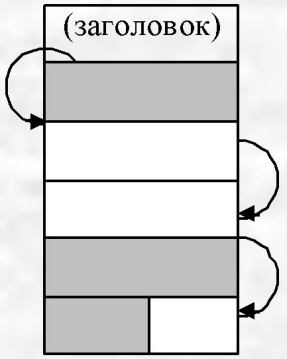
Ведение **списков свободных участков**.

Здесь можно рассмотреть два варианта:

1. Ссылка на первый свободный участок на странице хранится в заголовке страницы, и каждый свободный участок хранит ссылку на следующий (или признак конца списка). Каждый освобождаемый участок включается в список свободных участков на странице.
2. Списки свободных участков реализуются в виде отдельных структур. Эти структуры также хранятся на отдельных *инвентарных страницах*. Каждая инвентарная страница относится к области (или группе страниц) памяти и содержит информацию о свободных участках в этой области. Список ведётся как стек, очередь или упорядоченный список. В последнем случае упорядочение осуществляется по размеру свободного участка, что позволяет при размещении новой записи выбирать для неё наиболее подходящий по размеру участок.

Достоинства и недостатки.

списки свободных участков на странице



списки свободных участков в виде отдельных структур

5. Виды адресации хранимых записей

Рассмотрим три вида адресации: прямую, косвенную и относительную.

- **Прямая адресация** предусматривает указание непосредственного местоположения записи в пространстве памяти (например, в системе ADABAS). Недостатки: большой размер адреса; прямая адресация не позволяет перемещать записи в памяти без изменения КБД, что ведёт к фрагментации памяти.
- **Косвенная адресация**. В качестве КБД выступает не сам "адрес записи", а адрес места хранения "адреса записи".
- **Относительная адресация**. Общий принцип относительной адресации заключается в том, что адрес отсчитывается от начала той области памяти, которую занимают данные объекта БД. Если память разбита на страницы (блоки), то адресом может выступать номер страницы (блока) и номер записи на странице (или смещение от начала страницы). В случае относительной адресации перемещение записи приведёт к изменению КБД и необходимости корректировки индексов, если они есть.

5. Пример косвенной адресации

Часть адресного пространства страницы выделяется под индекс страницы. Число статей (слотов) в нём одинаково для всех страниц. В качестве КБД записи выступает совокупность номера нужной страницы и номера требуемого слота в индексе этой страницы (значения ***N, i***). В ***i***-м слоте на ***N***-й странице хранится собственно адрес записи (смещение от начала страницы).

6. 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**.

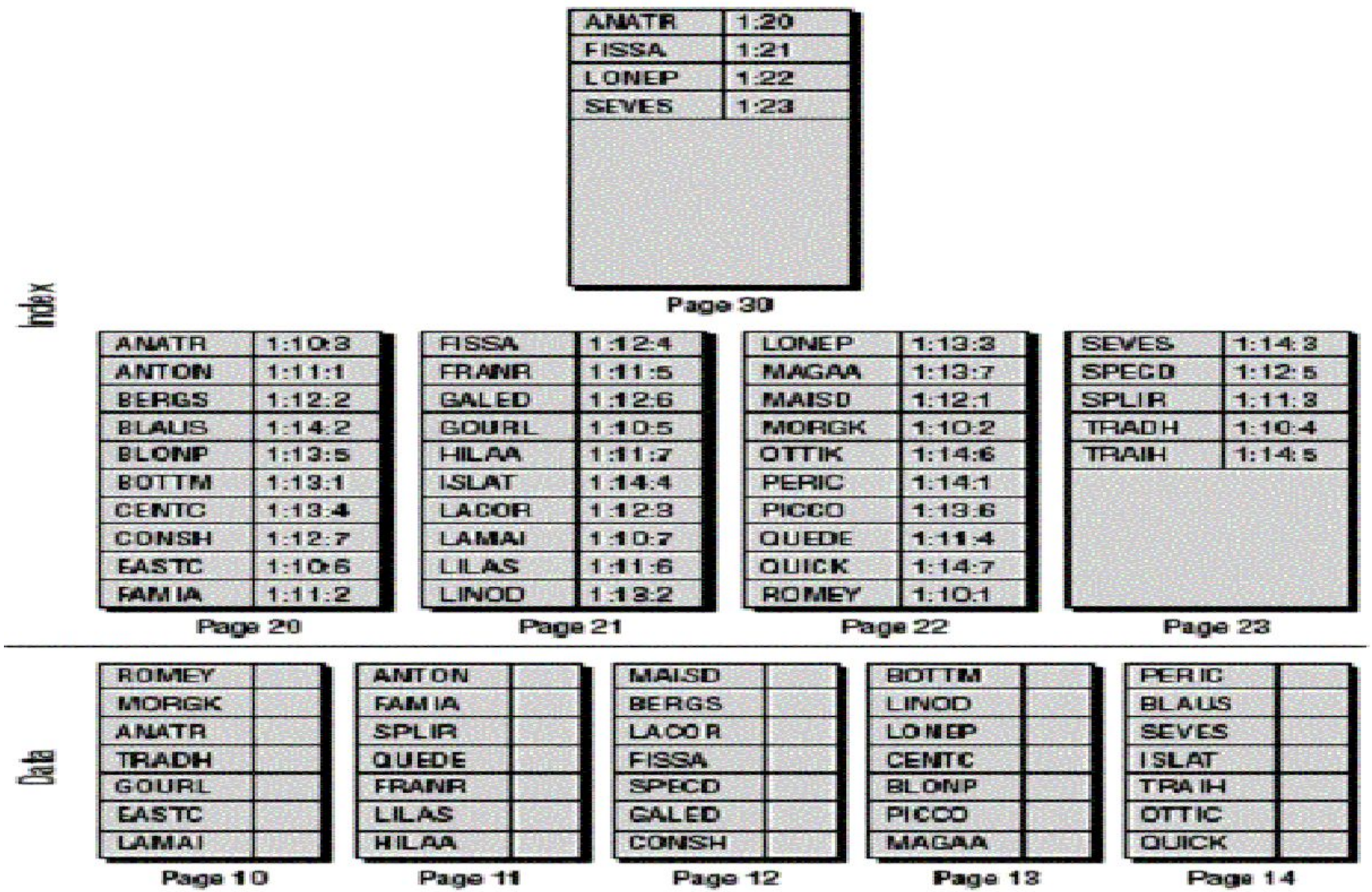
6. Основные типы индексов

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.

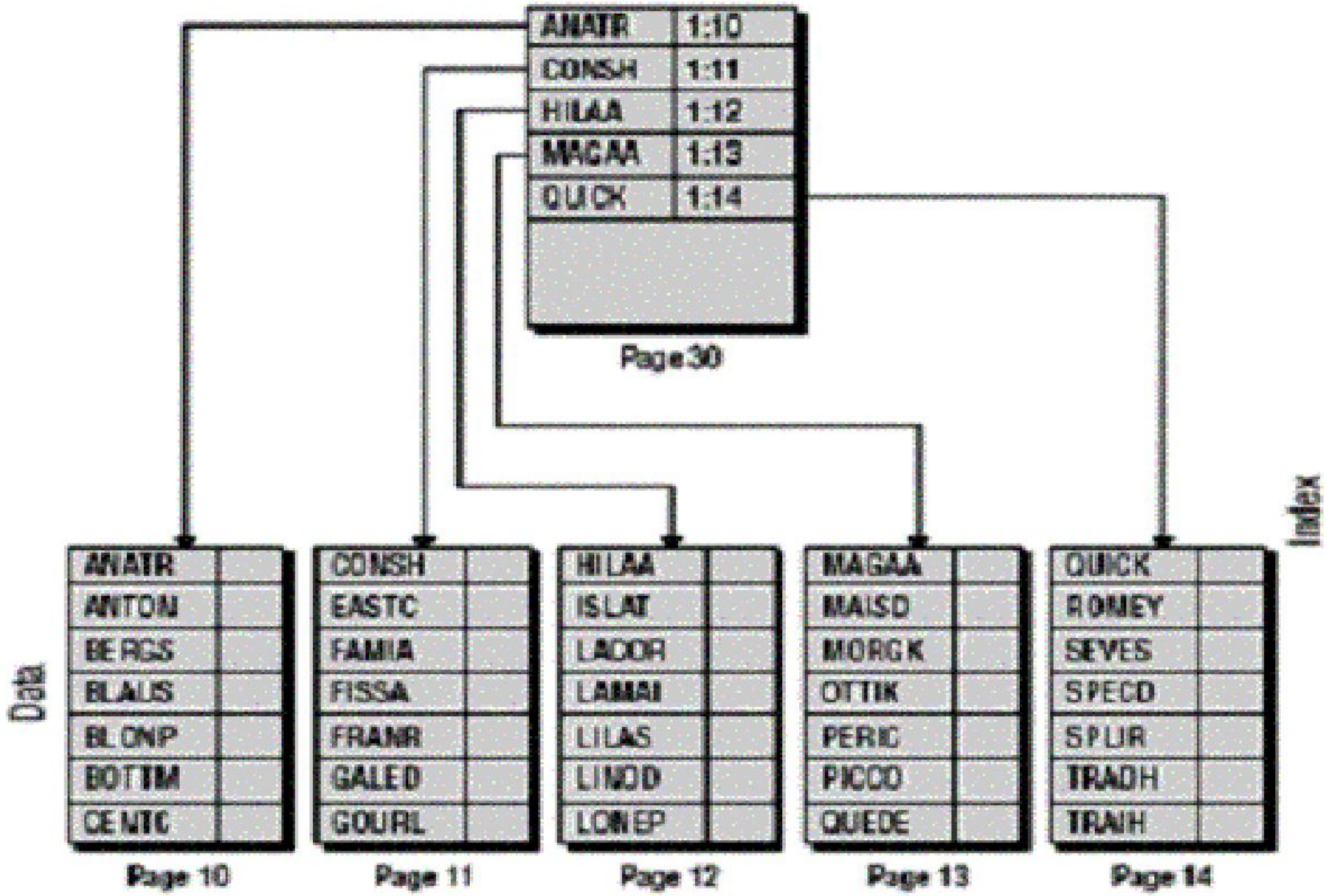
Кластерный индекс – при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса.

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.

6. Пример некластерного индекса



6. Пример кластерного индекса



7. Физическая организация данных в СУБД 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.

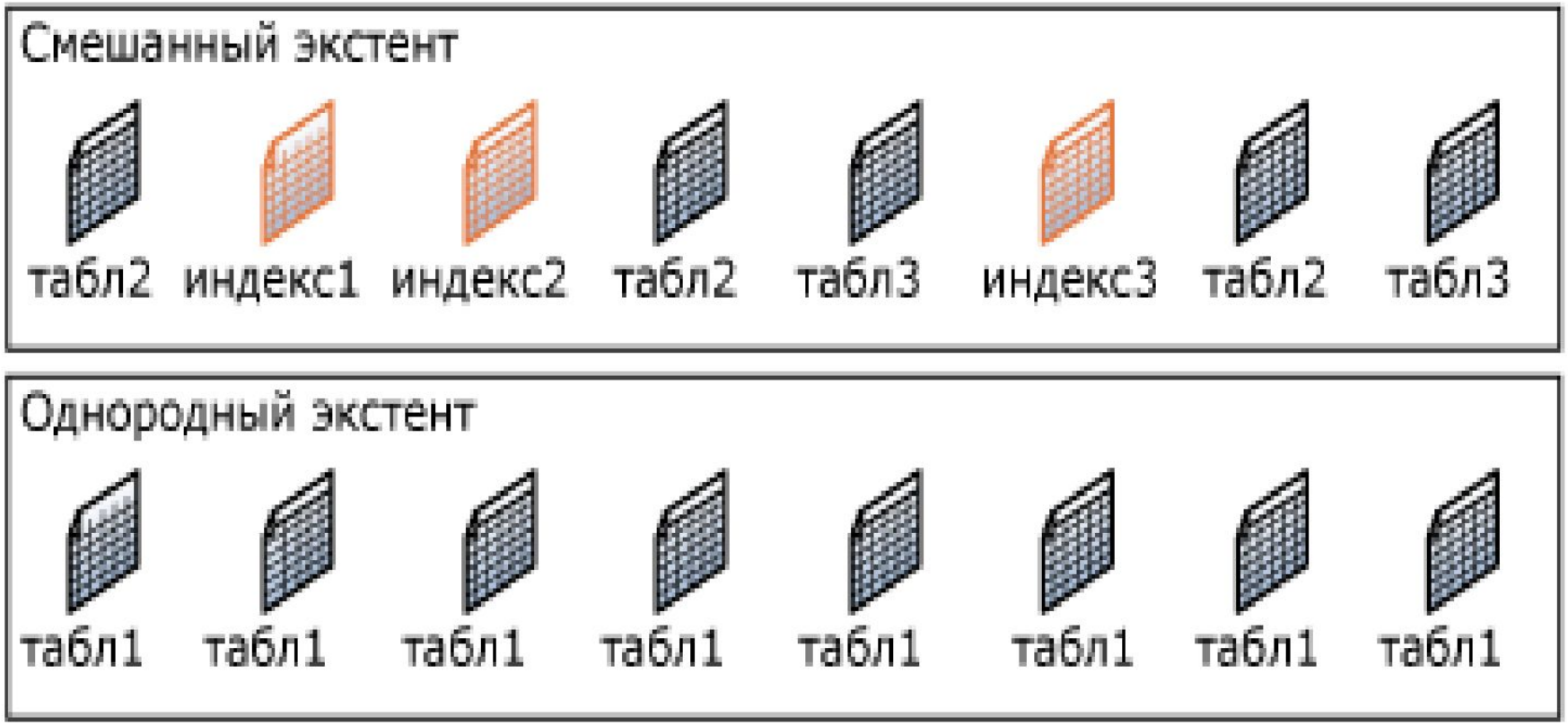
7. Страницы и экстенты в СУБД Microsoft SQL Server

Страница – основная единица хранения данных и обмена между внешней и оперативной памятью в Microsoft 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).

7. Примеры смешанного и однородного экстентов

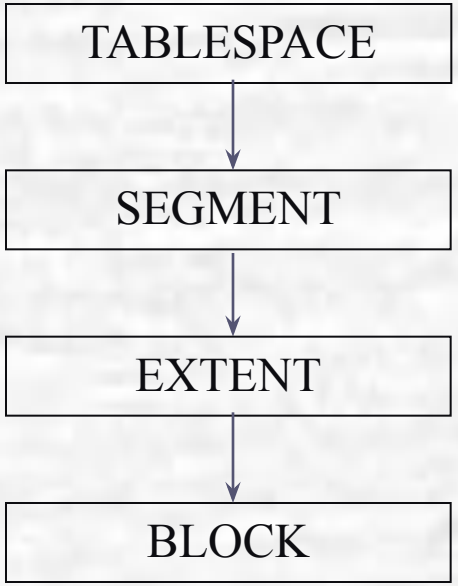


7. Секции в СУБД Microsoft SQL Server

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.

8. Основные физические структуры 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.

db_block_size – размер блока

8. Формат блока данных СУБД Oracle

Заголовок (общий и переменный)

- Заголовок содержит общую информацию блока, такую как адрес блока и тип сегмента (сегмент данных, сегмент индекса или сегмент отката). Заголовок составляет накладные расходы блока, которые имеют переменный размер. В среднем, суммарные накладные расходы фиксированной и переменной частей блока составляют от 84 до 107 байт.

Оглавление таблиц

- Часть блока, составляющая оглавление таблиц, содержит информацию о том, какие таблицы имеют строки в этом блоке.

Оглавление строк

- Эта часть блока содержит информацию о действительных строках в блоке (включая адреса каждой порции строки в области данных строк).

  После того, как в оглавлении строк распределено пространство, это пространство не освобождается при удалении строки. Это пространство используется повторно лишь тогда, когда в блок вставляются новые строки.

Данные строк

- Эта порция блока содержит данные таблицы или индекса.

Свободное пространство

- Свободное пространство в блоке используется для вставки новых строк и для обновлений строк, требующих дополнительного пространства.

Структура блока

8. Основные физические объекты СУБД Oracle

- **Кластер** (CLUSTER) – объект, задающий способ совместного хранения данных нескольких таблиц, содержащих информацию, обычно обрабатываемую совместно. Создается командой CREATE CLUSTER. Включает таблицы с данными.
- **Таблица** (TABLE) является базовой структурой реляционной модели. Таблица может быть пустой или состоять из одной или более строк значений атрибутов.. Создается командой CREATE TABLE, может быть создана в кластере.
- **Индекс** (INDEX) – это объект базы данных, создаваемый для повышения производительности выборки данных. Индекс создается для столбца (столбцов) таблицы и обеспечивает более быстрый доступ к данным этой таблицы за счет упорядочения данных столбца (столбцов) по значению. Создается командой CREATE INDEX.

Кластеры, таблицы и индексы называются *объектами, занимающими память*, т.к. в них хранятся фактографические данные.

9. Размеры блоков и экстентов в СУБД Oracle и сравнение с СУБД Microsoft SQL Server

Размер блока в СУБД Oracle – 2 кБ, 4 кБ, 8 кБ, 16 кБ или 32 кБ.

Размер экстента – от 64 кБ с автоматическим увеличением с кратность 2 до 2 ГБ.

- Oracle
  - Database
  - Tablespace
  - Segment
  - Extent
  - Block

- SQL Server
  - Database
  - Filegroup
  - Extent (64 KB fixed)
  - Page (8 KB fixed)

10. Состав языка Transact-SQL

Язык Transact-SQL СУБД Microsoft SQL Server 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.

11. Виды констант языка Transact-SQL

В языке Transact -SQL имеются следующие виды констант:

- битовые: 0 и 1;
- логические: FALSE и TRUE;
- бинарные в шестнадцатеричном представлении: 0x9E70DA;
- символьные: ‘ABC’; “ABC” (если QUOTED_IDENTIFIER = OFF); N ‘ABC’ (Unicode); N “ABC” (Unicode);
- целые: 1; 2; 175;
- с фиксированной точкой: 12.35; - 16.753;
- с плавающей точкой: 1.75E5; 3.84E – 3;
- для даты: “ April 15.2003”; “4/15/2003”; “20031207”;
- для времени: 14:30; 14:30:20:999; 4am; 4pm;
- денежные: $100.

12. Идентификаторы в языке Transact-SQL

Программные имена задаются идентификаторами двух типов: стандартными идентификаторами: Table X; Key Col; ограниченными идентификаторами: [My Table]; [Order]; “My Table”; “Order” (если QUOTED_IDENTIFIER = ON).

Длина идентификатора – от 1 до 128 символов. Идентификатором не может быть какое- либо зарезервированное ключевое слово языка.

Стандартный идентификатор в качестве первого символа может иметь любую латинскую или русскую букву, знаки #, ##, @, @@ и знак подчеркивания _. Последующими знаками, помимо указанных, могут быть еще и десятичные цифры.

Ограниченные идентификаторы могут включать и другие символы, в том числе зарезервированные слова. В этом случае они должны заключаться в квадратные скобки или двойные кавычки.

13. Именование объектов базы данных

В соответствии с идеологией Microsoft SQL Server каждый объект создается определенным пользователем и принадлежит той или иной базе данных. В свою очередь база данных расположена на конкретном сервере. Из имен объекта, пользователя, базы данных и сервера создается полное имя (complete name) или полностью определенное имя (full qualified name), записываемое в следующем виде:

[[[server.].[database].[owner_name].] object_name.

Варианты обращения к объектам базы данных: A.B.C.D; A.B..D; A..C.D; A..D; B.C.D; B..D; C.D; D.

Чтобы сослаться на конкретный столбец таблицы или представления, необходимо в полном имени указать пятый элемент: A.B.C.D.E.

14. 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, @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

15. Операторы выражений

Константы, переменные и параметры функций и хранимых процедур, вызовы функций, имена столбцов и подзапросы являются операндами арифметических и логических выражений.

Операторами выражения могут быть:
унарные (+ и -);
бинарные арифметические операторы (+, -, *, /, %);
оператор присваивания (=);
строковая операция конкатенации (+);
операторы сравнения (=, >, <, <=, >=, =, != или <>, !<, !>);
логические операторы (NOT, AND, OR, ALL, ANY, BETWEEN, EXIST, IN, LIKE, SOME);
битовые операторы (&, |, ^).

16. 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.

17. Специальные логические операторы

Оператор ALL

expr { = | <> | != | > | >= | !> | < | <= | !< } ALL (subquery)

Выполняется сравнение скалярного выражения со всеми значениями, возвращаемыми подзапросом. Если логическое условие выполняется для ***всех*** возвращаемых подзапросом значений, условие считается выполненным.

Операторы SOME и ANY

Разницы между использованием операторов ANY и SOME нет. Если хотя бы в ***одной*** строке содержится значение, равное скалярной величине, то условие выполняется.

Оператор BETWEEN

test_expression [NOT] BETWEEN begin_expression AND end_expression

Оператор дает ответ на вопрос, лежит ли величина в указанном диапазоне. Исходная величина задается аргументом test_expression. Аргумент begin_expression задает начало диапазона, а аргумент end_expression – конец диапазона

17. Специальные логические операторы

Оператор EXISTS

EXISTS (subquery)

Возвращает значение TRUE, если указанный подзапрос возвращает хотя бы одну строку. В противном случае возвращается значение FALSE.

Оператор IN

test_expression [NOT] IN (subquery | expression [,...n])

Проверяет, соответствует ли выражение test_expression одному из перечисленных выражений или значений, возвращаемых подзапросом.

Оператор LIKE

match_expression [NOT] LIKE pattern [ESCAPE escape_character]

Выполняет сравнение выражения с заданным шаблоном. Аргумент match_expression задает исходное выражение, для которого необходимо выполнить проверку. Шаблон для сравнения, включающий символы-заменители, задается с помощью аргумента pattern.

18. 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.

18. Типы хранимых процедур

Системные хранимые процедуры предназначены для выполнения различных административных действий. Системные хранимые процедуры имеют префикс **sp_**, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

Пользовательские хранимые процедуры реализуют те или иные действия. Хранимые процедуры – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

Временные хранимые процедуры существуют некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. **Локальные временные** хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей дается имя, начинающееся с одного символа #. **Глобальные временные** хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

18. Создание хранимых процедур

- **Определение типа** создаваемой хранимой процедуры**: временная или пользовательская**. Кроме этого, можно создать свою собственную **системную** хранимую процедуру, назначив ей имя с префиксом sp_ и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера;
- **Планирование прав доступа**. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;
- **Определение параметров** хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами;
- **Разработка кода** хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

18. Оператор создания/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]

18. Ключевые слова оператора

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для определения **типа данных**, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем. Однако тип данных CURSOR может быть использован только как выходной параметр хранимой процедуры, т.е. с указанием ключевого слова OUTPUT.

Указание **ключевого слова OUTPUT** предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра.

Ключевое слово VARYING применяется совместно с параметром OUTPUT, имеющим тип CURSOR. Оно определяет, что выходным параметром будет результирующее множество.

18. Ключевые слова оператора

Ключевое слово DEFAULT представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Указание **ключевого слова RECOMPILE** предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Параметр FOR REPLICATION требуется при репликации данных и включении создаваемой хранимой процедуры в качестве статьи в публикацию.

Ключевое слово ENCRYPTION предписывает серверу выполнить шифрование кода хранимой процедуры.

Ключевое слово AS размещается в начале тела хранимой процедуры, т.е. набора команд SQL. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры можно осуществить посредством команды RETURN.

18. 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.

18. Примеры хранимых процедур без параметров

1. Выдать наименования моделей радиосредств дуплексного типа.

CREATE PROC my_proc1 AS

SELECT RS02 FROM RS_RS02 WHERE RS04 = ’дуплекс’

2. Выдать наименования моделей радиосредств и диапазоны рабочих частот приемников для радиосредств симплексного типа.

CREATE PROC my_proc2 AS

SELECT RS_RS02.RS02, RR.RR02, RR.RR03 FROM RS_RS02, RR

WHERE RS_RS02.RS02 = RR.RS02 AND RS_RS02.RS04 = ’ симплекс’

18. Примеры хранимых процедур с входными параметрами

1. Выдать наименования моделей радиосредств заданного типа.

CREATE PROC my_proc1p @rs_type NCHAR(8) AS
SELECT RS02 FROM RS_RS02 WHERE RS04 = @rs_type

или с использованием значения по умолчанию:

CREATE PROC my_proc1p @rs_type NCHAR(8) = 'симплекс' AS
SELECT RS02 FROM RS_RS02 WHERE RS04 = @rs_type

Вызов процедуры:

EXEC my_proc1p @rs_type = 'дуплекс' или

EXEC my_proc1p 'дуплекс'

2. Выдать наименования моделей радиосредств и диапазоны рабочих частот приемников для радиосредств заданного типа.

CREATE PROC my_proc2p @rs_type NCHAR(8) AS
SELECT RS_RS02.RS02, RR.RR02, RR.RR03 FROM RS_RS02, RR
WHERE RS_RS02.RS02 = RR.RS02 AND RS_RS02.RS04 = @rs_type

18. Примеры хранимых процедур с выходными параметрами

1. Выдать количество моделей радиосредств дуплексного типа.

```
CREATE PROC my_proc1c @col INT OUTPUT  AS
SELECT @col = COUNT(RS02) FROM RS_RS02
WHERE RS04 = ’дуплекс’
```

Вызов процедуры:

```
DECLARE @c INT
EXEC my_proc1c @c OUTPUT
SELECT @c
```

2. Выдать количество моделей радиосредств заданного типа.

```
CREATE PROC my_proc2c @col INT OUTPUT, @rs_type NCHAR(8)  AS
SELECT @col = COUNT(RS02) FROM RS_RS02
WHERE RS04 = @rs_type
```

19. Хранимые функции

Определяемая пользователем **хранимая функция** представляет собой подпрограмму языка Transact-SQL, которая принимает параметры, выполняет заданные действия, например, сложные вычисления, а затем возвращает результат этих действий в виде значения.

Возвращаемое значение может быть скалярным значением или таблицей. Хранимую функцию можно использовать в операторах языка Transact-SQL, например, в операторе SELECT, а также в хранимых процедурах и других хранимых функциях.

19. Отличия хранимым функций от процедур

- функция всегда возвращает значение, а процедура может ничего не возвращать;
- в функции можно использовать только входные параметры, а в процедуре – входные и выходные параметры;
- функцию можно использовать в операторах языка Transact-SQL, а процедуру можно только вызвать оператором EXECUTE;
- в функции можно использовать только оператор SELECT, операторы INSERT, UPDATE, DELETE использовать нельзя, а в процедуре можно использовать все перечисленные операторы;

19. Отличия хранимым функций от процедур

- из функции нельзя вызвать хранимую процедуру, а из процедуры можно вызвать как хранимые функции, так и другие хранимые процедуры;
- в функции нельзя использовать операторы управления обработкой транзакций, а в процедуре можно;
- в функции нельзя использовать временные таблицы, а в процедуре можно;
- в функции нельзя использовать динамический SQL, а в процедуре можно;
- функция может возвращать только один результирующий набор (таблицу), а процедура – несколько наборов;
- в функции нельзя использовать временные таблицы, а в процедуре можно.

19. Создание хранимой функции

Для создания хранимой функции используется оператор CREATE FUNCTION.

В определении хранимой функции за ключевыми словами CREATE FUNCTION следует имя функции и определения ее параметров, которые заключаются в круглые скобки. Затем записывается ключевое слово RETURNS, после которого указывается тип данных возвращаемого значения функции.

Ключевое слово AS указывает начало тела функции. Если тело функции содержит более одного оператора, то оно заключается в операторные скобки BEGIN и END. При вызове функции указываются ее имя и фактические значения параметров в круглых скобках

20. Триггеры

Триггеры – это предварительно определенное действие или последовательность действий, автоматически осуществляемых при выполнении операций обновления, добавления или удаления данных.

Триггер – это специальный вид хранимой процедуры. Триггеры обеспечивают проверку любых изменений на корректность, прежде чем эти изменения будут приняты.

Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

20. Выполнение и назначение триггеров

Триггер выполняется неявно в каждом случае возникновения триггерного события. Приведение его в действие называют запуском триггера. С помощью триггеров достигаются следующие **цели**:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом.

20. Типы триггеров

Существует три типа триггеров:

1. **INSERT** – определяет действия, которые будут выполняться после добавления новой записи в таблицу или вместо добавления записи.
2. **UPDATE** – определяет действия, которые будут выполняться после изменения записи таблицы или вместо изменения записи.
3. **DELETE** – определяет действия, которые будут выполняться после удаления записи из таблиц или вместо удаления записи.

20. Создание триггеров

CREATE TRIGGER [имя_триггера]
ON имя_таблицы
{FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE] [,]
[DELETE]}
[WITH ENCRYPTION] AS
SQL_операторы

CREATE TRIGGER [имя_триггера]
ON имя_таблицы
{FOR | AFTER | INSTEAD OF} {[INSERT] [,] [UPDATE]}
[WITH ENCRYPTION] AS
IF UPDATE (имя_столбца) [{AND | OR} UPDATE
(имя_столбца)...]
SQL_операторы

20. Ограничения на триггеры

- Нельзя использовать в теле триггера операции создания объектов базы данных (новой базы данных, новой таблицы, новой хранимой процедуры, нового триггера, новых представлений);
- Нельзя использовать в триггере команду удаления объектов DROP для всех типов базовых объектов базы данных;
- Нельзя использовать в теле триггера команды изменения базовых объектов ALTER TABLE, ALTER DATABASE;
- Нельзя изменять права доступа к объектам базы данных, т.е. выполнять команду GRAND или REVOKE;
- Нельзя создать триггер для представления (VIEW).

20. Таблицы INSERTED и DELETED

- При INSERT в таблице INSERTED содержатся все строки, которые пользователь пытается вставить в таблицу. В таблице DELETED не будет ни одной строки;
- При DELETE в таблице DELETED будут содержаться все строки, которые пользователь пытается удалить. Триггер может проверить каждую строку и определить, разрешено ли ее удаление. В таблице INSERTED не окажется ни одной строки;
- При UPDATE в таблице DELETED находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице INSERTED. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.