Создание клиент-серверных приложений в СУБД SQL Server.

4.1 Средства администрирования SQL Server

4.1 Инструментальные средства

1. SQL Server Management Studio

SQL Server Management Studio – это главный рабочий инструмент администратора в SQL Server 2005, предназначен для ведения работы по созданию, модификации, обслуживанию баз данных и их объектов.

2. Business Intelligence Development Studio

Business Intelligence Development Studio – это второе важнейшее графическое средство для работы с SQL Server 2005. Business Intelligence дословно переводится как "бизнес-разведка", и, вообще говоря, этот термин традиционно относится к технологии Data Mining – добычи данных.

Business Intelligence Development Studio, как и SQL Server Management Studio, объединяет в себе возможности сразу нескольких программных средств, которые в предыдущих версиях SQL Server существовали по отдельности. Главное слово здесь – Development (разработка): это средство предназначено для работы с программными проектами. При помощи Business Intelligence Development Studio можно работать с проектами следующих типов:

1. проекты Analysis Services, которые представляют собой базы данных OLAP с необходимыми компонентами: кубами, общими измерениями, моделями добычи данных и т. п.;
2. проекты Integration Services, предназначенные для разработки средств преобразования и переноса данных между различными источниками – базами данных, текстовыми файлами, электронными таблицами и т.д.;
3. проекты Report Project – это отчеты по данным баз данных;
4. проекты Report Model – специальный тип отчета, предназначенный для того, чтобы наглядно представить структуру источника данных. Главные компоненты такого проекта – это Data Source Views (фактически это диаграммы баз данных) и Report Models (описание сущностей, атрибутов и связей между ними в базе данных).

Интерфейс Business Intelligence Development Studio вряд ли заслуживает отдельного описания, поскольку при запуске этого

приложения вам просто открывается среда разработки Visual Studio 2005. В ней можно создать или открыть проект нужного вам типа и работать с ним стандартными средствами Visual Studio.

3. SQL Server Configuration Manager

SQL Server Configuration Manager – это средство конфигурирования служб и сетевых параметров сервера.

4.2 Службы сервера

SQL Server 2005, как и все серверные продукты Microsoft, реализован в виде набора служб:

- SQL Server – это сам SQL Server, ядро базы данных. Оно ответственно за работу с файлами базы данных, прием пользовательских подключений, выполнение запросов и т. п.
- SQL Server Agent – специальная служба, которая ответственна за автоматизацию работы с SQL Server. Она отвечает за выполнение заданий по расписанию, за предупреждения и другие служебные операции. Для хранения информации этой службы на SQL Server создается специальная служебная база данных MSDB. Обратите внимание: если вы принимали значения по умолчанию на экранах мастера установки SQL Server 2005, то эта служба автоматически запускаться не будет.
- Report Server – эта служба представляет серверный компонент Reporting Services. Она ответственна за генерацию отчетов, предоставление их пользователям, выполнение различных служебных операций с отчетами.
- Analysis Server – ядро сервера баз данных OLAP. Эта служба полностью ответственна за работу с базами данных OLAP и их компонентами, например, с кубами.
- DTS Server – это служба, ответственная за работу с новой средой DTS (т. е. за операции загрузки, выгрузки и преобразования данных, которые проводятся при помощи пакетов DTS).
- msftesql – эта служба раньше называлась Microsoft Search. Ее главная задача – работа с полнотекстовыми индексами (SQL Server 2005 поддерживает и русскоязычный полнотекстовый поиск).

В состав SQL Server 2005 входят еще две службы, но в Configuration Manager они не отображаются.

- SQL Browser – эта служба ответственна за формирование списка серверов SQL Server в сети.

- SQL Writer – работает с теневыми копиями (shadow copies) баз данных SQL Server 2005 и используется для проведения резервного копирования в оперативном режиме, без отключения пользователей.

4.2 Типы данных SQL-Server

Вид	Тип	Интервал значений	Размер
Двоичные	*binary*(n) *varbinary*(n | max) *image*	до 8 000 байт n – до 8 000 байт | max – до 2^{31} байт до 2 Гбайт (для совместимости с Server 2000, рекомендуется *varbinary*(max))	n n до 2 Гбайт
Символьные	*char*(n) *varchar*(n | max) *nchar*(n) *nvarchar*(n | max)	n – до 8 000 байт n – до 8 000 байт | max – до 2^{31} байт n – до 4 000 байт Unicode n – до 4 000 байт Unicode | max – до 2^{31} байт)	n n n n
Текст	*text* *ntext*	до 2 Гбайт до 1 Гбайт Unicode (для совместимости с Server 2000, рекомендуется *varchar*(max) и *nvarchar*(max))	до 2 Гбайт до 1 Гбайт Unicode
Дата и время	*datetime* *smalldatetime*	01.01.1753-31.12.9999г. до 3,33 мс 01.01.1900-06.06.2079г. до 1 мин.	8 байт 4 байта
Точное представление чисел	*decimal*(p, s) *numeric*(p, s)	p <= 38, s <= p p <= 38, s <= p	При p = 2 – 2 байта, при p = 38 – 17 байт
Числа с плавающей точкой	*float*(n) *real*	$1,8*10^{308}$ $1,8*10^{308}$	n = 1 – 53 n = 24 n – число бит мантиссы

Целочисленные типы	*int* *smallint* *tinyint* *bigint*	$2*10^{10}$ 32 767 0 – 255 $9*10^{19}$	4 байта 2 байта 1 байт 8 байт
Денежные типы	*money* *smallmoney*	$9*10^{15}$, 4 знака после запятой 214 748.3648	8 байт 4 байта
Специальные	*Bit*	0 или 1 (логический)	1 бит
	Timestamp	уникальный идентификатор, генерируется автоматически при вставке или изменении записи	8 байт
	uniqueidentifier	глобально-уникальный идентификатор (GUID), генерируется функцией NewID()	16 байт
	sql_variant	переменный тип данных	
	Table	таблица, столбцы которой описаны в соответствии с синтаксисом оператора CREATE TABLE	
	Cursor	указатель на курсор	1 байт
	xml	текст, содержащий данные XML	

Все типы данных используются в программах на языке T-SQL. При определении типов данных колонок таблиц запрещено использование типов данных Table, Cursor.

Установка формата даты: SET DATEFORMAT dmy
Установка первого дня недели: SET DATEFIST 1 (1 – понедельник, 7 – воскресенье)

Преобразование типов данных:
`CAST(expr AS type)`
`CONVERT(type, expr, [style])` style – при преобразовании в дату: 4 – dd.mm.yy, 104 – dd.mm.yyyy

Синонимы типов данных SQL-Server 2000:

- *binary varying* для *varbinary*

- *character* для *char*
- *character* для *char (1)*
- *character* (n) для *char (n)*
- *character varying (n)* для *varchar (n)*
- *dec* для *decimal*
- *integer* для *int*
- *double precision* для *float*
- *float* (n) для n = 1-7 для *real*
- *float* (n) для n = 8-15 для *float*

4.3 Файловая структура базы данных SQL Server

Главные и дополнительные файлы баз данных в SQL Server 2005, применение неформатированных разделов, оптимальное размещение файлов баз данных и журналов транзакций

Одно из самых важных решений, которое необходимо принять при создании базы данных – это решение о структуре и размещении файлов данных и журналов транзакций. Неверное решение может существенно снизить производительность и надежность работы приложения.

Для любой базы данных создаются файлы самой базы данных и файлы журналов транзакций. В файлах базы данных хранится вся информация о самой базе данных. В файлы журналов транзакций производится последовательная запись всех изменений, которые вносятся в базу данных. Минимальный набор файлов для любой базы данных (он же используется по умолчанию) содержит один файл для самой базы данных и один файл для журнала транзакций.

В каждой базе данных обязательно есть один главный (*primary*) файл. По умолчанию для него используется расширение mdf (хотя использовать именно такое расширение не обязательно – для любых файлов баз данных и журналов транзакций расширения могут быть любыми, а могут и отсутствовать). Удалять этот файл нельзя. Для базы данных можно создать и дополнительные файлы (*secondary*), для которых по умолчанию используется расширение ndf. Точно так же есть главный и дополнительные файлы у журналов транзакций, для них по умолчанию используется расширение ldf.

В принципе, база данных может вообще обходиться без файлов. Вся необходимая информация при этом будет храниться на неформатрированном диске. Такой вариант называется **использованием неформатированных разделов** (*raw partitions*).

А теперь подробнее остановимся на тех решениях, которые необходимо принимать при создании базы данных.
Идеальный вариант, с точки зрения размещения файлов баз данных, – поместить их на отдельный внешний аппаратный RAID-массив. Причем на этом RAID-массиве не должно быть ничего, кроме файлов базы данных, и, кроме того, на нем должно быть как минимум 50% пустого пространства. Такой вариант обеспечивает ряд преимуществ:

- RAID-массив (в зависимости от выбранного уровня) обеспечивает высокую производительность и отказоустойчивость;
- внешний RAID-массив делает процесс восстановления больших баз данных предельно простым и быстрым. Если на сервере возникли какие-то проблемы, а с файлами базы данных все в порядке, достаточно просто подключить RAID-массив к другому серверу и присоединить базу данных;
- если выбрать внешний RAID-массив, который входит в список совместимого оборудования (Hardware Compatibility List) для кластеров Windows Server, то при необходимости можно еще больше повысить отказоустойчивость за счет создания кластера;
- если на диске будет не менее половины пространства свободно, то этим обеспечивается отсутствие проблем при выполнении различных служебных операций, например, при перестроении кластеризованных индексов для больших таблиц.

Конечно, внешний RAID-массив – это идеальный вариант. Но на многих предприятиях денег на него может просто не быть. В этом случае рекомендуется, по крайней мере, использовать для файлов баз данных отдельный быстрый жесткий диск. Категорически не рекомендуется помещать на тот же диск, где находятся файлы баз данных, программные файлы операционной системы и SQL Server. Следует помнить также, что на контроллерах доменов для разделов, на которые помещается база данных Active Directory (по умолчанию она находится в каталоге C:\Windows\NTDS), отключается кэширование на запись: падение производительности может быть просто устрашающим.
Конечно же, никогда нельзя сжимать рабочие файлы баз данных средствами NTFS или помещать их на сжатые диски.
Теперь о размещении файлов журналов транзакций. Требования по производительности к ним намного меньше, чем к файлам баз данных (одна из причин связана с тем, что в файлы журналов транзакций записываются только изменения, которые вносятся в базу данных – команды SELECT на них, за исключением специальных случаев,

влияния не оказывают). Как размещать эти файлы, зависит от требований к базе данных и бюджета предприятия. Далее представлены разные варианты, начиная от наиболее желательного и заканчивая наименее удачным:

- второй RAID-массив;
- отдельный набор дисков на том же RAID-массиве, что и файлы баз данных;
- два обычных диска, которые зазеркалированы по отношению друг к другу;
- просто обычный отдельный диск;
- размещение на том же диске, на котором размещены файлы баз данных (этот вариант наименее желателен, но именно он по умолчанию выбирается SQL Server 2005 в расчете на однодисковые системы).

Главное, что вам нужно постараться обеспечить, – это размещение журналов транзакций на другом физическом диске по отношению к файлам баз данных. В случае отказа жесткого диска можно провести восстановление на момент сбоя (конечно, при условии, что резервное копирование все-таки производилось). Если же файлы баз данных и журналов транзакций находились на одном диске, и этот диск отказал, восстановиться базу данных удастся только на момент создания последней резервной копии. В этом случае пользователям придется заново вносить данные, например, за последний день или за последнюю неделю – в зависимости от того, когда последний раз производилось резервное копирование.

Второе решение, которые следует принять при создании базы данных, – выбор размера файлов баз данных и журналов транзакций.

Конечно, размер файлов баз данных полностью зависит от задачи, для которой используется эта база данных. Однако есть выбор – сразу создать большие файлы баз данных или настроить для них режим автоматического приращения, когда файлы при необходимости будут автоматически увеличиваться. Если такая возможность имеется, всегда нужно с самого начала создавать файлы максимального размера (или, по крайней мере, настраивать автоприращение сразу большими частями, например, в несколько Гбайт), даже не смотря на то, что в течение продолжительного времени значительная часть этих файлов использоваться не будет. Таким образом, снижается фрагментация файлов баз данных, повышая производительность. По умолчанию для файлов баз данных настраивается худший вариант – автоприращение маленькими "порциями": 1 Мбайт для файлов базы

данных и 10% от существующего размера для файлов журналов транзакций.

Однако нужно помнить, что при уменьшении размера файла (например, при удалении информации из базы данных) сделать его меньше исходного размера не получится.

Настройку режима автоприращения при помощи графического интерфейса можно выполнить в окне **New Database** при создании новой базы данных или на вкладке **General** свойств базы данных в SQL Server Management Studio. Режим автоприращения устанавливается в соответствующей строке для каждого файла базы данных нажатием на кнопку в столбце **Autogrowth** (Автоматический рост) свойств данного файла.

Эти же самые принципы относятся и к настройке автоприращения файлов журналов транзакций. Однако для файлов журналов транзакций возникает еще один вопрос: какой размер журнала транзакций нужен для конкретной базы данных? Ответ на этот вопрос зависит от множества факторов:

- первый фактор — в каком режиме используется база данных. Если это база данных OLTP (т. е. данные в ней изменяются постоянно, чаще всего пользователями при помощи клиентских приложений), к которой относится абсолютное большинство используемых на предприятиях баз данных, то Microsoft рекомендует устанавливать для журналов транзакций размер от 10 до 25% от общего размера файлов баз данных. Для баз данных Data Warehouse (архивные хранилища, которые в обычном режиме используются только на чтение и пополняются, как правило, средствами массовой загрузки данных или пакетами SSIS/DTS) достаточно будет и нескольких процентов от объема файлов баз данных;
- второй фактор – какой режим восстановления настроен для базы данных. Режим восстановления **Simple** предъявляет минимальные требования к размеру файлов журнала (поскольку старые записи в журнале сразу же перезаписываются), а режим восстановления **Full** требует файлов журнала намного большего размера;
- если база данных работает в режиме восстановления **Full**, то записи в журнале транзакций будут копиться до бесконечности, пока не будет произведено резервное копирование журнала транзакций или журнал транзакций не будет очищен вручную. Поэтому при определении необходимого размера файлов журналов транзакций нужно учитывать и частоту резервного копирования.

Файловые группы баз данных SQL Server 2005, применение, файловая группа по умолчанию

При создании файла базы данных можно указать, к какой файловой группе он будет относиться. Файловая группа (*Filegroup*) – это способ организации файлов базы данных. По умолчанию для любой базы данных создается файловая группа PRIMARY, и все создаваемые файлы базы данных по умолчанию будут относиться именно к ней. В каких ситуациях вам может потребоваться создание дополнительных файловых групп?

Первая ситуация – оптимизация резервного копирования. Все таблицы базы данных можно условно разделить на две части:

- пользовательские таблицы, которые постоянно изменяются пользователями;
- таблицы справочников, которые меняются очень редко (например, когда приходят обновления от разработчиков).

Предположим, что у пользовательских таблиц объем небольшой, а таблицы справочника, наоборот, занимают много места. В то же время, с точки зрения резервного копирования, намного важнее пользовательские таблицы. Оптимизировать резервное копирование в этой ситуации можно так:

- при создании базы данных создаем дополнительную файловую группу USERS. Создаем новый файл данных, например, users.mdf, и определяем, что он будет относиться к этой группе;
- при создании пользовательских таблиц и индексов к ним определяем, что они будут принадлежать файловой группе USERS (для этой цели в командах CREATE TABLE и CREATE INDEX используется ключевое слово ON с указанием имени файловой группы). Обратите внимание, что назначать таблицам и индексам конкретные файлы нельзя, а файловые группы можно;
- таблицы справочников оставляем в файловой группе PRIMARY или создаем для них свою файловую группу.

Далее будем производить резервное копирование отдельных файловых групп с разным расписанием. Например, резервное копирование файловой группы USERS можно производить каждый день, а файловой группы PRIMARY раз в месяц.

Единственный момент, который при этом нужно учесть, связан с тем, что для обеспечения целостности данных между файловыми группами SQL Server не будет очищать журнал транзакций до тех пор, пока не будет произведено резервное копирование всех файловых

групп (используется система так называемых поколений резервных копий). Таким образом, очистка журнала транзакций в этой ситуации будет производиться раз в месяц.

Вторая ситуация, когда может потребоваться использовать дополнительные файловые группы, – ручное распределение нагрузки в дисковой подсистеме. Предположим, например, что на сервере есть два жестких диска: быстрый и относительно медленный. Файлы из файловой группы USERS можно разместить на быстром диске, а файлы для редкоиспользуемых справочников – на медленном, и, таким образом, можно повысить скорость работы системы для пользователей.

Третья ситуация – ручное распараллеливание запросов в дисковой подсистеме. Разместив, например, в разных файловых группах таблицу и индексы к ней, можно добиться, чтобы в обслуживании запроса к этой таблице принимали участие оба диска. Выигрыш, конечно, получится небольшим (RAID-массив, с точки зрения производительности, дает намного большие преимущества), тем не менее, такая возможность существует.

Создать новые файловые группы можно или при помощи графического интерфейса SQL Server Management Studio (на вкладке **Filegroups** (Файловые группы) свойств базы данных), или при помощи команд CREATE DATABASE, ALTER DATABASE. Создаваемую файловую группу можно сделать файловой группой по умолчанию. В этом случае все новые таблицы и индексы в базе данных при создании будут по умолчанию помещаться в эту файловую группу.

4.4 Объекты базы данных SQL Server. Язык определения данных SQL-DDL.

Команды языка SQL делятся на две категории: язык определения данных DDL, предназначенный для создания и управления объектами БД, и язык управления данными DML, предназначенный для выборки и обработки данных содержащихся в таблицах БД.

База данных – на логическом уровне – это система объектов, на физическом – совокупность одного или нескольких файлов данных (первичные – mdf, вторичные – ndf) и одного или нескольких файлов журнала транзакций (ldf). Синтаксис команды для создания БД:

```
CREATE DATABASE 'ИмяБД'
ON [PRIMARY]
```

```
   ([NAME = 'ЛогическоеИмяФайлаДанных',]
   FILENAME = 'ФизическоеИмяФайлаДанных'
   [, SIZE = НачальныйРазмерФайлаДанных [KB, MB, GB,
TB]
   [, MAXSIZE = МаксимальныйРазмерФайлаДанных [KB, MB,
GB, TB]]
   [, FILEGROWTH = РазмерПриращенияФайлаДанных [KB,
MB, GB, TB | Percent]])
   [LOG ON
   ([NAME = 'ЛогическоеИмяФайлаЖурнТранз',]
   FILENAME = 'ФизическоеИмяФайлаЖурнТранз'
   [, SIZE = НачальныйРазмерФайлаЖурнТранз [KB, MB,
GB, TB]
   [, MAXSIZE = МаксимальныйРазмерФайлаЖурнТранз [KB,
MB, GB, TB]]
   [, FILEGROWTH = РазмерПриращенияФайлаЖурнТранз [KB,
MB, GB, TB | Percent]])]
```

Логические имена служат для управления файловой структурой БД с помощью операторов модификации БД. Начальный размер файла данных по умолчанию равен размеру файла БД Model. Если не указывается максимальный размер файла, то он считается сколь угодно большим (естественно, ограничен размерами диска). Размер приращения файла БД может быть задан в байтах (кило, мега и т.д.) или процентах (при указании ключевого слова `Percent`). Если опция LOG ON не указывается, то журнал транзакции создается в виде одного файла, для которого определяется размер, равный 25% от файла данных.

Модификация базы данных производится с помощью команды:

```
   ALTER DATABASE ИмяБД
   | ADD FILE
   (ОписаниеНовогоФайлаДанных)
   | ADD LOG FILE
   (ОписаниеНовогоФайлаЖурнТранз)
   | REMOVE FILE 'ЛогическоеИмяФайлаДанных'
   | REMOVE LOG FILE 'ЛогическоеИмяФайлаЖурнТранз'
   |                  M O D I F Y                  F I L E
'ЛогическоеИмяФайлаДанных' (НовоеОписаниеФайлаДанных)
   | MODIFY LOG FILE 'ЛогическоеИмяФайлаЖурнТранз'
   (НовоеОписаниеФайлаЖурнТранз)
```

Рассмотрим основные объекты базы данных SQL Server 2000 и связанные с ними команды SQL-DDL.

1. Таблицы – массивы, хранящие данные, на логическом уроне представляют собой двумерные таблицы, состоящие из столбцов (колонок, полей) и строк (записей).

```
CREATE TABLE ИмяТаблицы
(ОпределениеКолонки1, …)
ОграниченияНаУровнеТаблицы
ON ГруппаФайлов | DEFAULT
TEXTIMAGE ON ГруппаФайлов | DEFAULT
```

Здесь:

- ОпределениеКолонки – описание свойств каждой колонки в виде:

```
ИмяКолонки ТипДанных
     [[PRIMARY KEY | UNIQUE CLUSTERED |
NONCLUSTERED
          WITH FILLFACTOR =
     ПроцентФактораЗаполнения] |
[FOREING KEY REFERENCES МастерТаблица
(МастерКолонка)
ON DELETE CASCADE | NO ACTION
ON UPDATE CASCADE | NO ACTION]]
     [DEFAULT Выражение | ИмяУмолчания]
     [IDENTITY (НачЗначение, Приращение)]
     [ROWGUIDCOL]
     [NULL | NOT NULL]
[CHECK ЛогическоеВыражение | ИмяПравила]
[AS ВыражениеДляВычисляемогоСтолбца]
```

- ОграниченияНаУровнеТаблицы – применяются, если необходимо создать первичный индекс или внешний ключ не по одному столбцу, а по нескольким:

Первичный ключ:

```
PRIMARY KEY | UNIQUE CLUSTERED | NONCLUSTERED
(ИмяКолонки1 ASC | DESC , …)
     WITH FILLFACTOR = ПроцентФактораЗаполнения
     ON ГруппаФайлов | DEFAULT
```

Или внешний ключ:

```
FOREING KEY
(ИмяКолонки1 , …)
     REFERENCES МастерТаблица (МастерКолонка1 ,
…)
ON DELETE CASCADE | NO ACTION
ON UPDATE CASCADE | NO ACTION
```

- ГруппаФайлов | DEFAULT – группа файлов в которой будет сохранена таблица или, в случае TEXTIMAGE ON – группа файлов, в которой будут сохранены столбцы типов Text и Image.

Атрибуты столбцов таблиц:

1. Column Name – имя колонки
2. Data Type – тип данных
3. Length – длина n для символьных видов
4. Allow Nulls – разрешение значения NULL
5. Description – описание
6. Default Value – значение по умолчанию (н-ер: NewId() для *uniqueidentifier*, GetDate() для *datetime*)
7. Precision – точность (общее количество знаков), p – для *decimal* и *numeric*
8. Scale – масштаб (количество знаков после запятой), s – для *decimal* и *numeric*
9. Identity – признак счетчика
10. Identity Seed – начальное значение счетчика
11. Identity Increment – шаг приращения счетчика
12. Is RowGuid – признак глобального идентификатора
13. Formula – для вычисляемых столбцов (н-ер: Цена * Количество)
14. Collation – сопоставление для сравнения и сортировки строк

Модификация таблиц производится с помощью оператора ALTER TABLE:

```
ALTER TABLE ИмяТаблицы
| ALTER COLUMN ИмяИзменяемойКолонки
(НовоеОпределениеКолонки)
| ADD COLUMN ОпределениеНовойКолонки
| DROP COLUMN ИмяУдаляемойКолонки
| ADD CONSTRAINT ОпределениеНовогоОграничения
| DROP CONSTRAINT ИмяУдаляемогоОграничения
```

2. Представления – виртуальные таблицы, создаваемые выборкой данных из таблиц БД. Служат для ограничения прямого доступа пользователей к таблицам и для вывода данных из нескольких связанных таблиц (или других представлений) в виде одной таблицы.

Представления создаются оператором

```
CREATE VIEW ИмяПредставления [(ИмяКолонки1, …)]
[WITH ENCRYPTION SCHEMABINDING]
```

```
AS SQL_Select
[WITH CHECK OPTION]
```

Здесь

- Колонка1, … - имена колонок в представлении, если в качестве таковых выступают вычисляемые выражения или данные выбираются из различных таблиц из колонок имеющих одинаковые имена;
- ENCRYPTION – указание шифровки текста запроса;
- SCHEMABINDING – указание сохранить схему зависимых объектов;
- SQL_Select – текст SQL-оператора SELECT;
- WITH CHECK OPTION – указание осуществлять проверку изменений данных представления пользователем на соответствие критериям, определенным в предложениях WHERE или HAVING оператора SELECT. Если такие изменения не соответствуют критериям, что повлечет за собой исключение строки из представления, то изменения будут отвергнуты.

3. Хранимые процедуры – наборы команд T-SQL, предназначенные для управления сервером, изменения структуры базы данных и обработки данных в таблицах. В хранимых процедурах и в функциях пользователя, как правило, реализуется бизнес-логика приложений баз данных, т.е. с их помощью создаются приложения сервера.

Хранимые процедуры создаются командой

```
CREATE PROCEDURE
  ИмяПроцедуры [@Параметр1 ТипДанных1
[=ЗначПоУмолчанию] [OUT[PUT]], …]
[WITH [SCHEMABINDING] [ENCRYPTION]]
AS
Операторы_TSQL
```

Здесь:

- Операторы_TSQL – набор операторов языка T-SQL.
- Слово OUTPUT (возможно сокращение до OUT) служит для передачи параметра по ссылке (объявления выходного параметра).

Запускаются хранимые процедуры на выполнение командой

```
EXEC[UTE] ИмяПроцедуры [Выражение1 [OUTPUT], …]
```

Если при создании процедуры некоторый параметр объявлен как выходной, то и при вызове процедуры для соответствующего параметра должно быть использовано слово OUTPUT.

Модификация хранимых процедур производится с помощью команды ALTER PROCEDURE.

4. Триггеры – специальные хранимые процедуры, автоматически срабатывающие при изменении данных в таблицах. Триггеры разделяются на триггеры вставки, изменения и удаления (соответственно INSERT, UPDATE, DELETE - триггеры).

```
CREATE TRIGGER ИмяТриггера
ON ИмяТаблицы
WITH ENCRYPTION
FOR | AFTER | INSTEAD OF INSERT, UPDATE, DELETE
AS
SQL_Statment
```

5. Индексы – структуры, предназначенные для повышения производительности работы с данными. Индекс содержит отсортированные значения одного или нескольких столбцов таблицы со ссылкой на соответствующую строку исходной таблицы. В отсортированной последовательности поиск нужной строки с помощью специальных алгоритмов производится многократно быстрее, чем последовательный построчный перебор значений. Индексы делятся на:

- кластерные – перестраивающие структуру таблиц, таким образом, что бы строки таблицы физически располагались в порядке изменения значений индекса. Для каждой таблицы кластерный индекс может быть только один. По умолчанию, кластерный индекс создается для первичного ключа таблицы;
- некластерные – не перестраивают структуру таблиц, а только организуют ссылки на соответствующие строки;
- уникальные – некластерные индексы, гарантирующие уникальность значений в индексном столбце.

Индексы создаются несколькими способами:

1. автоматически при создании первичного ключа таблицы.
2. автоматически при создании ограничения целостности UNIQUE;
3. определение индекса при создании таблицы командой CREATE TABLE;
4. командой

```
     CREATE UNIQUE | CLUSTERED | NONCLUSTERED INDEX
ИмяИндекса
     ON ИмяТаблицы (ИмяКолонки1 ASC | DESC, …)
     WITH PAD_INDEX, FILLFACTOR = ПроцентЗаполнения,
IGNORE_DUP_KEY
```

PAD_INDEX – указание серверу резервировать на каждой странице свободное пространство для вставки новых записей.

FILLFACTOR – задает степень заполнения индексных страниц в процентах.

6. Пользовательские типы данных – типы данных, создаваемые пользователем на основе встроенных типов данных сервера. Служат для создания однотипных столбцов в разных таблицах. Пользовательские типы данных создаются системной хранимой процедурой:

Sp_addtype ИмяПТД, БазовыйТип, NULL | NOT NULL

7. Функции пользователя – функции, созданные пользователем для поверки вводимых или существующих данных. В зависимости от возвращаемого значения, существует три вида функций пользователя: скалярные, In-line и Multy-stasment.

Скалярная:

```
     CREATE FUNCTION ИмяФункции ([@Параметр1
ТипДанных1 [=ЗначПоУмолчанию], …])
     RETURNS СкалярныйТип
[WITH [SCHEMABINDING] [ENCRYPTION]]
     AS
     BEGIN
     Операторы
     RETURN ЗначениеСкалярногоТипа
          END
```

Функция In-Line (в одну линию):

```
     CREATE FUNCTION ИмяФункции ([@Параметр1
ТипДанных1 [=ЗначПоУмолчанию], …])
     RETURNS TABLE
[WITH [SCHEMABINDING] [ENCRYPTION]]
          AS
     RETURN (Оператор_SELECT)
          GO
```

Функция Multy-stasment (многострочная):

```
CREATE FUNCTION ИмяФункции ([@Параметр1
ТипДанных1 [=ЗначПоУмолчанию], …])
RETURNS @ИмяПеремТипаТабл (ОписаниеСтруктуры)
[WITH [SCHEMABINDING] [ENCRYPTION]]
     AS
BEGIN
Операторы
RETURN
     END
```

8. Ограничения целостности данных – объекты, обеспечивающие логическую целостность данных в связанных таблицах. Создание ограничений целостности производится для каждой таблицы отдельно при ее создании или модификации с помощью предложения FOREIGN KEY REFERENCES.

8. Ключи – объекты, обеспечивающие логическую целостность данных и однозначную идентификацию строк в пределах одной таблицы. Создание ключевого объекта выполняется для каждой таблицы отдельно при ее создании или модификации (предложение PRIMARY KEY команды CREATE или ALTER TABLE).

9. Правила – объекты, обеспечивающие контроль данных на уровне столбца таблиц. Правила представляют собой логические выражения, накладываемые на значение того или иного столбца таблицы. Если при изменении данных в таком столбце соответствующие логическое выражение не выполняется, то изменения данных в строке будут отвергнуты сервером.

Правила создаются командой

```
CREATE RULE ИмяПравила AS ЛогическоеВыражение
```

Здесь ЛогическоеВыражение – выражение, определяющее условие, накладываемое на значение столбца. В выражении допустима одна локальная переменная с произвольным именем, значение которой трактуется как значение поля, к которому применено правило.

Созданное правило может быть связано с любым столбцом любой таблицы по его имени, указанному в ограничении целостности CHECK.

10. Умолчания – объекты, автоматизирующие заполнение новых строк таблиц значениями «по-умолчанию».

```
CREATE DEFAULT ИмяУмолчания AS Выражение
```

Здесь Выражение – константа или выражение соответствующего типа.

Созданное умолчание может быть связано с любым столбцом соответствующего типа любой таблицы по его имени, указанному в предложении DEFAULT команды CREATE или ALTER TABLE.

Для всех объектов, поддерживающих команду CREATE, существует команда ALTER - изменение существующего объекта и DROP – удаление объекта. Например:

Удаление таблицы:

```
DROP TABLE ИмяТаблицы
```

Удаление представления:

```
DROP VIEW ИмяПредставления
```

Удаление правила:

```
DROP RULE ИмяПравила
```

И т.д.

4.5 Команды управления данными (SQL-DML) в SQL Server.

1. Команда выборки данных SELECT.

```
SELECT [Диапазон] СписокСтолбцов FROM СписокТаблицСУсловиямиОбъединения
[WHERE Условие]
    [GROUP BY УсловиеГруппировки [[HAVING УсловиеНаГруппу]]
    [ORDER BY УсловиеСортировки]
    [COMPUTE ИтоговоеВыражение]
    [UNION SELECT …]
```

Здесь:

Диапазон – задает диапазон строк, возвращаемый после выполнения запроса. Синтаксис:

```
{ALL | DISTINCT} {TOP n [PERCENT] [WITH TIES]}
```

где ALL | DISTINCT – включать или нет в результат выборки повторяющиеся строки. По умолчанию действует установка ALL;

TOP n [PERCENT] [WITH TIES] – прямое указание количества первых строк, которое должно быть возвращено запросом. Если указано ключевое слово PERCENT, то количество определяется не абсолютным значением, а в процентах от общего количества (естественно, в этом случае n <= 100). Если дополнительно указано предложение WITH TIES, то будут возвращены и строки, дублирующие последние во множестве, ограниченном значением n.

СписокСтолбцов – задает список столбцов в результирующем множестве. Список может быть задан символом * - выборка всех столбцов из всех таблиц участвующих в предложении FROM, или перечислением имен колонок и/или выражениями. Если в запросе участвую несколько таблиц, то имена колонок следует задавать в формате:

ИмяТаблицы.* - для выборки всех столбцов таблицы,

ИмяТаблицы.ИмяКолонки | АлиасТаблицы.ИмяКолонки – для включения конкретного столбца,

Выражение – для построения вычисляемых столбцов.

В двух последних случаях можно переопределить имя любого столбца с помощью предложения AS НовоеИмя. В качестве выражений могут выступать любые, в соответствии с типом данных столбцов операции, функции сервера или пользователя, агрегатные функции (Count(), Count(*), Sum(), Max(), Min(), Avg(), …).

FROM СписокТаблицСУсловиямиОбъединения – раздел, задающий имя таблицы или список таблиц, из которых выбираются данные в запросе и условия их объединения, если таблиц несколько. Синтаксис списка:

```
ИмяТаблицы1 [AS Алиас1]
INNER | { LEFT | RIGHT | FULL } [OUTER] JOIN
ИмяТаблицы2 [AS Алиас2]
ON ИмяТаблицы1.ИмяСтолбца1 = ИмяТаблицы2.ИмяСтолбца2
| ON Алиас1.ИмяСтолбца1 = Алиас2.ИмяСтолбца2
```

WHERE Условие – задает логическое выражение, ограничивающее отбор строк в результирующее множество. В качестве такого выражения может служить простое логическое условие, несколько логических условий объединенных логическими операторами. В качестве операндов – логических условий – константы, имена столбцов, выражения, результаты выполнения

подзапроса. В качестве операторов – простое (= ; > ; < и т.д.) или множественное (IN ; ALL ; ANY ; EXIST) логическое условие. Примеры:

```
Select Код, Улица From Улицы Where Код > 5

Select Код, Улица From Улицы Where Код > 5 AND Код < 100

Select Сотрудники.ТабНом, Сотрудники.Фамилия, Начисления.Начислено
From Сотрудники Inner Join Начисления
On Сотрудники.ТабНом = Начисления. ТабНом
Where Начисления.Начислено >= (Select Avg(Начислено) From Начисления)

Select Клиенты.ЛицевойНомер, Клиенты.Фамилия
From Клиенты Inner Join Льготы
    On Клиенты. ЛицевойНомер = Льготы. ЛицевойНомер
Where Клиенты.ЛицевойНомер Not In
(Select Distinc Льготы.ЛицевойНомер From Льготы Where
    Льготы.Код = '01')
```

С помощью оператора IN могут формироваться *многостолбцовые подзапросы*, которые содержат более одного атрибута в списке подзапроса. Такое же количество атрибутов должно быть указано в предложении WHERE главного запроса. Обязательным условие является попарное совпадение типа и размера данных атрибутов подзапроса и главного запроса. Такие запросы имеют следующий синтаксис:

```
SELECT <Атрибут1>, < Атрибут2> [, …] FROM <Таблица1>
WHERE (<Атрибут1>, < Атрибут2> [,…]) [NOT] IN
(SELECT < Атрибут1>, < Атрибут2> [, …] FROM <Таблица2>
WHERE <Условие>)
```

Предложение WHERE при использовании операторов ANY и ALL представляется следующим образом:

```
WHERE <Выражение> ANY | ALL (SELECT <Атрибут> FROM …)
```

где - опрератор сравнения (<, >, >= и т.д.). Например:

```
WHERE x > ANY (SELECT y FROM …)
```

Условие считается истинным, если *x* больше хотя бы одного значения *y* в результирующем множестве выполнения подзапроса. Очевидно, что в предложении

```
WHERE x <> ANY (SELECT y FROM …)
```

условие будет выполнено, если *x* не совпадает ни с одним из значений результата выполнения подзапроса. Оператор = ANY эквивалентен оператору IN.

В предложении

```
WHERE x > ALL (SELECT y FROM …)
```

условие будет истинным, если x больше всех значений y множества, формируемого подзапросом. Можно отметить, что оператор > ALL означает «больше, чем максимум», а < ALL – «меньше, чем минимум».

Многостолбцовые подзапросы выполняются один раз и возвращают в предложение WHERE главного запроса одну или несколько строк, со значениями атрибутов которых производится сравнение атрибутов главного запроса. *Коррелированные подзапросы*, в отличие от многостолбцовых, выполняются для каждого кортежа главного запроса. В коррелированных подзапросах применяется оператор существования. Синтаксис такого запроса в общем виде:

```
     SELECT <Атрибут1>, < Атрибут2> [, …] FROM
<Таблица1>
     WHERE EXISTS
     (SELECT * FROM <Таблица2>
     WHERE <Таблица1>.< АтрибутN> <Таблица2>.<
АтрибутM>
                              AND <Условие>)
```

Пример. Выбрать кафедры, на которых работают сотрудники в возрасте до 23 лет:

```
SELECT Кафедры.Наименование FROM Кафедры WHERE EXISTS
(SELECT * FROM Сотрудники WHERE Кафедры. Наименование =
Сотрудники.Кафедра
AND Сотрудники.Возраст < 23)
```

GROUP BY – раздел, выполняющий группировку строк в соответствии с некоторым условием. В качестве условия чаще всего выступает список столбцов, по значениям которых следует сгруппировать данные. Как правило, группировка имеет смысл, если для каждой группы производятся некоторые вычисления. Пример:

1. Выборка всех начислений для всех сотрудников за указанный период

```
          Select Сотрудники.ТабНом, Сотрудники.Фамилия,
     Начисления.Начислено
               From Сотрудники Inner Join Начисления
                    On Сотрудники.ТабНом = Начисления.
               ТабНом
```

```
            Where Начисления.Период Between '01.01.07'
        And '31.12.07'
```

2. Определение суммы начислений и среднего заработка для каждого сотрудника

```
    Select Сотрудники.ТабНом, Сотрудники.Фамилия,
        Sum(Начисления.Начислено),
    Avg(Начисления.Начислено)
From Сотрудники Inner Join Начисления On Сотрудники.ТабНом =
    Начисления.ТабНом
    Where Начисления.Период Between '01.01.01' And
'31.12.01'
    Group By Сотрудники.ТабНом
```

HAVING УсловиеНаГруппу – раздел, описывающий условие включения группы в результирующее множество.

ORDER BY УсловиеСортировки – определение сортировки результатов. Здесь

УсловиеСортировки - список столбцов в формате:

```
        ИмяКолонки1 [ASC | DESC], …
```

где ASC | DESC - указание порядка (по возрастанию или по убыванию) сортировки результата запроса по значениям столбца. По умолчанию используется ASC.

COMPUTE ИтоговоеВыражение – задает выражение, вычисляемое в качестве итогового для всего результата запроса или каждой группы, если в запросе используется группировка данных. В качестве выражения должны использоваться агрегатные функции SQL. Например:

```
    Select Сотрудники.ТабНом, Сотрудники.Фамилия,
Начисления.Начислено
From Сотрудники Inner Join Начисления
        On Сотрудники.ТабНом = Начисления.ТабНом
    Where Начисления.Период Between '01.01.01' And
'31.12.01'
    Order By Сотрудники.ТабНом Asc, Начисления.Период Asc
    Compute Sum(Начисления.Начислено),
Avg(Начисления.Начислено)
```

Результатом будет таблица со всеми начислениями всех сотрудников плюс две дополнительные строки с суммой всех начислений и средним значением.

```
    Select Сотрудники.ТабНом, Сотрудники.Фамилия,
Начисления.Начислено
    From Сотрудники Inner Join Начисления On
Сотрудники.ТабНом = Начисления.ТабНом
```

```
Where Начисления.Период Between '01.01.01' And '31.12.01'
Group By Сотрудники.ТабНом
Compute Sum(Начисления.Начислено), Avg(Начисления.Начислено)
```

Результатом будет таблица с данными для каждого сотрудника по всем его начислениям и, дополнительно, после данных каждого сотрудника будут добавлены две строки с суммой его начислений и средним заработком.

UNION – предложение, содержащее еще один оператор SELECT. В одном запросе допускается использование нескольких предложений UNION. Результаты выполнения всех операторов SELECT будут объединены в одно множество. Условием возможности такого объединения является совпадение имен колонок во всех запросах (при необходимости следует использовать алиасы) и типов данных (при необходимости следует использовать функции преобразования). При объединении результатов сортировка каждого запроса не имеет смысла, но допускается применение одного «общего» предложения ORDER BY. Пример:

```
Select ТабНомер As Код, Фамилия, 'Преподаватели' As Статус From Преподаватели
Union Select ТабНомер As Код, Фамилия, 'Сотрудники' As Статус From СотрудникиАХЧ
Union Select НомерЗачетки As Код, Фамилия, 'Студенты' As Статус From Студенты
Order By Статус
```

2. Команда добавления данных в существующие таблицы и представления INSERT.

```
INSERT INTO {ИмяТаблицы WITH УровеньБлокировки} | ИмяПредставления
        |{СписокКолонок)
        | VALUES {(DEFAULT | NULL | Выражение1, … )}
              | SQL_Select
              | SQL_Execute }
    |DEFAULT VALUES
```

Здесь:

ИмяТаблицы и ИмяПредставления – имя таблицы или представления, в которую производится вставка строк.

УровеньБлокировки – определяет уровень блокировки (хинт) таблицы. Если уровень блокировки для таблицы не указан, то он

определяется сервером автоматически. Для представлений уровень блокировки не указывается и всегда определяется автоматически.

СписокКолонок – определяет список столбцов, в который будет производится вставка данных. Аргумент необязательный, и, если он опущен, то вставка данных производится последовательно, в порядке определения столбцов при создании таблицы.

VALUES – ключевое слово, задающее значения данных. Список значений определяется одним из следующих вариантов:

1) указанием значения для каждого столбца в виде: DEFAULT – значения по-умолчанию; NULL – значения NULL; Выражение – выражения с использованием констант, переменных и функций.

2) SQL_Select – подзапросом с помощью оператора SELECT

3) SQL_Execute – результатом выполнения хранимой процедуры, т.е. выполнения команды:

EXEC ИмяХранимойПроцедуры Параметры.

DEFAULT VALUES – указание вставки для всех столбцов новой строки значений, определенных по-умолчанию. Данный аргумент является альтернативой конструкции VALUES

3. Команда создания таблиц и одновременного добавления в них данных SELECT INTO.

```
SELECT ИмяКолонки1 AS Алиас1, …
     INTO ИмяНовойТаблицы
     FROM SQL_Select
```

Здесь:

ИмяКолонки – задает имя колонки таблицы, которая будет включена в результат,

AS Алиас – переопределение имени колонки в создаваемой таблице. Аргумент необходим в случае, если данные выбираются из нескольких таблиц, имеющих столбцы с одинаковыми именами.

ИмяНовойТаблицы – имя создаваемой таблицы. Структура и данные таблицы будут определяться типом данных и данными столбцов, выбираемых оператором SELECT.

SQL_Select – оператор SELECT.

Выполнение команды SELECT INTO по умолчанию запрещено. Для включения или выключения возможности ее выполнения используется хранимая процедура с соответствующим параметром.

```
EXEC SP_DBOPTION 'SELECT INTO/BULKCOPY', 'ON' | 'OFF'
```

4. Обновление данных UPDATE.

```
UPDATE {ИмяТаблицы WITH УровеньБлокировки} | ИмяПредставления
      SET ИмяКолонки1 = DEFAULT | NULL | Выражение1 , …
      WHERE Условие
```

Здесь:

`SET` – ключевое слово, определяющее список изменяемых столбцов и соответствующее каждому столбцу новое значение (по-умолчанию, NULL или определяемое выражением).

`WHERE` – раздел, в котором задается `Условие` – логическое выражение, определяющее фильтр для строк, в которых будут изменены данные. Если WHERE отсутствует, то обновляются все строки.

5. Команда удаления данных DELETE.

```
DELETE FROM {ИмяТаблицы WITH УровеньБлокировки} | ИмяПредставления
      WHERE Условие
```

Команда удаляет строки из указанной таблицы или представления в соответствии с выбранным в предложении WHERE условием. Если WHERE отсутствует, то удаляются все строки.

4.6 Основы Transact-SQL Server

4.7 Курсоры SQL Server

Курсоры СУБД SQL Server представляют собой механизм обмена данными между клиентом и сервером в тех случаях, когда результатом выборки по запросу клиента является значительный объем данных, который может чрезмерно загрузить сетевой трафик. Курсоры позволяют клиентским приложениям работать не с полным набором данных, а с одной или несколькими строками, постепенно расширяя выборку по мере необходимости.

SQL Server 2003 поддерживает четыре типа курсоров: статические, динамические, последовательные и ключевые

(основанные на наборе ключей). Тип курсора определяется на стадии его создания и в последующем изменен быть не может.

При работе с курсорами в SQL Server выделяют следующие шесть операций.

1) Объявление курсора.

В SQL Server 2003 объявление (создание) курсора возможно с помощью команды в стандарте SQL-92 и на «родном» языке Transact-SQL.

Объявление курсора тем или иным способом аналогично объявлению переменной в языках Pascal, C. При объявлении курсора создается соответствующий объект в системной базе данных TempDB сервера.

1.1) в стандарте SQL – 92:

```
DECLARE ИмяКурсора [INSENSITIVE] [SCROLL] CURSOR
FOR SQL_Select [FOR READ ONLY]|[UPDATE [OF ИмяКолонки1, …]]
```

INSENSITIVE – создается статический курсор, в котором изменения данных не разрешаются, и изменения, сделанные другими пользователями не отображаются. Если слово INSENSITIVE опущено, то создается динамический курсор, в котором разрешено изменение данных (если не указано FOR READ ONLY), и изменения, сделанные другими пользователями отражаются при чтении соответствующей строки.

SCROLL – создается прокручивающийся курсор, позволяющий использовать любые команды выборки (чтения) курсора. Если слово SCROLL опущено – создается последовательный курсор, позволяющий читать данные только последовательно, строка за строкой.

UPDATE OF ИмяКолонки1 [, …] – перечисление колонок, в которых возможно изменение данных при создании динамического курсора. Если OF ИмяКолонки1 [, …] опущено изменению доступны все колонки.

SQL_Select – оператор SELECT языка T-SQL.

1.2) на языке T-SQL:

```
DECLARE ИмяКурсора CURSOR
[LOCAL|GLOBAL]
[FORWARD_ONLY|SCROLL]
[STATIC|KEYSET|DYNAMIC|FAST_FORWARD]
```

```
[READ_ONLY|OPTIMISTIC]
[TYPE_WARNING]
FOR SQL_Select [FOR UPDATE [OF Column1, …]]
```

LOCAL|GLOBAL – При объявлении курсора с помощью T-SQL может быть объявлен как локальный курсор (LOCAL), т.е. видимый в пределах создавшей его хранимой процедуры, триггера или функции и неявно уничтожающийся при выходе из процедуры, так и глобальный (GLOBAL), существующий до закрытия текущего соединения с сервером. При работе с глобальным курсором, ключевое слово GLOBAL должно присутствовать во всех командах.

FORWARD_ONLY|SCROLL – создается, соответственно, последовательный или прокручивающийся курсор.

STATIC – создание статического курсора.

KEYSET – создание ключевого курсора. Ключевые курсоры строятся на наборе ключей, идентифицирующих строки результирующей выборки. Такие курсоры отражают все изменения, сделанные другими пользователями, поскольку они хранят информацию только о ключевых столбах.

DYNAMIC – создание динамического курсора.

FAST_FORWARD – создание последовательного курсора.

READ_ONLY – курсор только-для-чтения.

OPTIMISTIC – в курсоре, созданном с указанием этого ключевого слова, запрещается изменение или удаление строк, измененных другими пользователями после открытия курсора.

TYPE_WARNING – если тип курсора, определяемый командой DECLARE CURSOR, не совместим с оператором SQL_Select, то тип курсора будет автоматически преобразован. Наличие ключевого слова TYPE_WARNING предписывает в этом случае выдачу соответствующего предупреждения.

2) Открытие курсора:

```
OPEN [GLOBAL] ИмяКурсора
```

Открытие курсора соответствует его созданию и заполнению данными в системной базе данных TempDB.

3) Чтение данных в курсоре:

```
FETCH [NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|@n | RELATIVE n|@n]
FROM [GLOBAL] ИмяКурсора [INTO @Var1 [, …]]
```

Команда обеспечивает изменение текущей строки и чтение данных в следующей, предыдущей, первой или последней строке, или в строке с номером, определяемом значением n (значением переменной @n), или в строке, расположенной относительно текущей на n|@n позиций. Причем, если в последнем случае n положительное, то текущая строка изменяется в сторону увеличения номеров строк, если n отрицательное – то в сторону уменьшения. По-умолчанию, производится чтение следующей строки.

INTO @Var1 [, …] – имена переменных, в которые помещаются значения колонок. Если предложение опущено, то данные выводятся на экран.

4) Обновление данных в курсоре:

Если созданный курсор является динамическим, то с его помощью можно изменять данные в таблицах операторами UPDATE и DELETE. И в том и в другом случае обрабатывается строка таблицы (ИмяТаблицы), определяемая текущей строкой курсора (WHERE CURRENT OF ИмяКурсора).

4.1) Изменение строки в таблице:

```
UPDATE ИмяТаблицы
SET ИмяКолонки1 = Default|Null|выражение [, …]
WHERE CURRENT OF ИмяКурсора
```

4.2) Удаление строк в таблице с помощью курсора:

```
DELETE ИмяТаблицы WHERE CURRENT OF ИмяКурсора
```

5) Закрытие курсора:

```
CLOSE GLOBAL ИмяКурсора
```

Закрытие курсора удаляет записи, созданные данным курсором в системной базе данных TempDB сервера. Закрытый курсор может быть открыт повторно командой OPEN.

6) Освобождение курсора:

```
DEALLOCATE GLOBAL ИмяКурсора
```

Освобождение курсора удаляет его как объект из системной базы данных.

4.8 Синтаксические конструкции языка программирования Transact-SQL

4.9 Технологии доступа к данным в клиент-серверных приложениях

4.9.1 DB-Library

Интерфейс DB-Library (DB-Lib) представляет собой библиотеку с набором API-функций, осуществляющих прямой доступ к функциям SQL-Server из различных программных сред. Технология DB-Lib в настоящее время не поддерживается фирмой Microsoft, но используется в огромном количестве разработанных ранее систем.

При использовании DB-Lib в состав проекта необходимо включать несколько дополнительных файлов, например для C – SQLDB.H, SQLFRONT.H; для Delphi – BLDBLIB.LIB, BMDBLIB.LIB; для Visual Basic – VBSQL.OCX, VBSQL.BAS.

В среде VB для подключения к серверу используются функции:

```
Login% = SqlLogin%()
Result% = SqlSetLUser%(Login%, "loginid")
Result% = SqlSetLPwd%(Login%, "passwd")
Result% = SqlSetLApp%(Login%, "myapp")
SqlConn% = SqlOpen%(Login%, "server")
```

Если на сервере используется интегрированная схема защиты, заполнение полей loginid и passwd является обязательным, однако сервер при уста новке соединения игнорирует их. В этом случае сервер использует данные поль зователя, указанные им при регистрации в системе Windows. В подобной си туации, если заранее известно, что в SQL-Server используется интегрированная схема защиты, в соответствующие поля можно помещать произвольную инфор мацию. Описательное имя приложения myapp не является обязательным, однако использовать его настоятельно рекомендуется. Назначение описательного имени приложения состоит в следующем. Если работа системы SQL-Server контролируется системным администратором, то при анализе им списка открытых соединений, в этот список будет помещено описательное имя приложения. Если это

имя будет достаточно информативным, системный администратор сможет в любой момент точно установить, кто рабо тает с сервером.

При отправке команд серверу, прежде всего, необходимо поместить их в буфер команд SQL. Помещение команды в этот буфер осуществляется посредством вызова процедуры SqlCmd() с параметрами, содержащими данные, которые следует поместить в буфер:

Status% = SqlCmd(Sqlconn%, “Команда”)

Каждая помещаемая в буфер команда дописывается в конец предыдущей, которая была туда помешена при вызове функции SqlCmd . Это позволяет создавать команды, размер которых больше одной строки:

Status% = SqlCmd(Sqlconn%, “SELECT * FROM pubs”)

Status% = SqlCmd(Sqlconn%, “ WHERE autor LIKE ‘A’”)

После помещения требуемой команды в буфер она отсылается серверу на выполнение функцией SqlExec(), синтаксис которой:

Status% = SqlExec(Sqlconn%)

Поскольку текст команды был уже создан, и она была связана с конкретным соединением, функция отправляет находящуюся в буфере SQL-соединения команду на сервер, который компилирует и выполняет ее.

Если необходимо **вызвать** хранимую процедуру, в создаваемой команде указывается фраза EXECUTE, например:

Status% = SqlCmd(Sqlconn%, “EXECUTE GetAuthors ‘A’”)

В данном случае вызывается на выполнение хранимая процедура GetAuthors, которой передается параметр 'A', используемый для определения критерия отбора.

После того как в среду SQL-Server был отправлен запрос, необходимо обработать результат его выполнения. Для этого используются две следующие константы, предназначенные для контроля за ходом обработки полученного набора данных:

SUCCEED

NOMOREROWS

Эти константы определяются в файлах с расширениями .bas, которые необходимы для работы с библиотекой DB-Library. При обработке набора данных, возвращенного сервером как результат выполнения запроса, следует последовательно обрабатывать одну строку данных за другой, пока не будет получено значение NOMOREROWS, указывающее, что все найденные строки уже

обработаны приложением. Текущее состояние набора данных можно определить с помощью функции Status% = SqlResults%(Sqlconn%)

Если выполнение запроса завершится успешно и программе будет возвращена запрашиваемая информация, можно организовать цикл ее обработки с использованием функции

SqlNextRow(Sqlconn%)

предназначенной для последовательной построчной обработки полученных данных. Результат запроса помешается в рабочий буфер, в котором данные становятся доступными для обработки. Когда при очередном вызове функции SqlNextRow () оказывается. что достигнут конец буфера, эта функция возвращает значение NOMOREROWS. По этому сигналу приложение должно завершить цикл обработки полученного набора данных.

Последним этапом работы с полученной информацией является выборка ее из буфера. Для этого предназначена функция SqlData$(Sqlconn%, Column), где Column – номер колонки в результирующем наборе (возвращаемые функциями DB-Lib результаты не именованные, а перечисляемые, т.е. колонки определяются не именами, а номерами).

Например:

```
…
Status% = SqlCmd(Sqlconn%, “SELECT * FROM pubs”)
Status% = SqlExec(Sqlconn%)
While SqlNextRow(Sqlconn%) <> NOMOREROWS
Print SqlData$(1)
Wend
…
```

После завершения всей требуемой обработки данных, доставляемых через установленное соединение, последнее обязательно следует закрыть. Это необходимо для возврата системе использованной памяти и закрытия соединения в среде SQL-Server. Для закрытия соединения предназначена функция SqlClose% (Sqlconn%)

Необходимо закрывать каждое соединение, открытое в приложении для получения досту па к серверу. В качестве альтернативы можно применить функцию SqlExit(), которая автоматически закрывает все открытые на данный момент соединения с сервером.

4.8.2 ODBC

Технология ODBC разработана с целью обеспечения возможности установки соединений с (и между) различными СУБД. Эта технология предусматривает создание дополнительного уровня абстракции между приложением и используемой СУБД. Данный уровень позволяет создать одну команду Select и использовать ее для выборки данных из разных поддерживаемых типов баз данных, включая и те случаи, когда СУБД вообще не под держивает язык SQL.

Службы ODBC образуют особый уровень доступа к файлам баз данных. ODBC берет на себя ответственность за получение от приложения запросов на выборку информации и перевода их на язык, используемый ядром адресуемой базы данных для доступа к размешенной в ней информации. Технология ODBC обеспечивает приложению универсальный интерфейс доступа к информации. На его основе разрабатывается общий набор необходимых запросов и правил обработки полученных результатов, без учета специфики используемых баз данных.

Сеанс обработки данных ODBC

При установке средств ODBC. устанавливается не только общая подсистема ODBC. но н опреде ляются конкретные пары "драйвер-база данных". Этим парам присваиваются имена, исполь зуемые впоследствии в запросах на установку соединения с соответствующей базой данных. Определения подобных пар, состоящих из драйвера и базы данных, называют *поименованными источниками данных* (DSN).

В большинстве языков программирования в команде установки ODBC-соединения имеет ся несколько параметров. Прежде всего, это имя того соединения, которое следует установить. Если оно не будет указано в команде, система выведет диалоговое окно с предложением указать требуемое соединение, выбрать базу данных и указать логин и пароль пользователя:

Set db = OpenDatabase(“”,,,”odbc;”)

В следующем примере в приложении указывается строка подключения, содержащая все сведения необходимые для установления соединения без участия пользователя:

```
strCn = "ODBC;DSN=Access_SQL_ORDB;"
strCn = strCn & "DATABASE=Pubs;Network=DBMSRPCN;"
strCn = strCn & "Trusted_Connection=Yes;"
strCn = strCn & "Regional = Yes;UID=Логин;PWD=Пароль;"
```

```
Set db = OpenDatabase("", False, False, strCn)
```

После установления соединения с базой данных можно работать средствами DAO. Пример:

```
Sub ArchiveSa1es()
Dim dbs As DATABASE, qdf As QueryDef
Dim strCn As String, rst As Recordset
Dim strSQL As String
On Error GoTo ErrorHandter
Set dbs = CurrentDb
strCn = "ODBC;DSN=Pubs;UID=sa;PWD=;DATABASE=Pubs"
Set qdf = dbs.CreateQueryDefC"")
qdf.Connect = strCn
strSQL = "CREATE TABLE dbo.sa1es_archive (stor_id char (4) "
strSQL = strSQL & "NOT NULL, ord_num varchar (20) NOT NULL, "
strSQL = strSQL & "ord_date datefime NOT NULL, "
strSQL = strSQL & "qty small1nt NOT NULL, "
strSQL = strSQL & "payterms varchar (12) NOT NULL,"
strSQL = strSQL & " tit1e_id tid NOT NULL)"
qdf.ReturnsRecords = False
qdf.SQL = strSQL
qdf.Execute
strSQL = "INSERT INTO sales_archive SELECT * "
strSQL = strSQL & "FROM sales WHERE "
strSQL = strSQL & "ord_date < ‘1/1/93’"
qdf.SQL = strSQL
qdf.Execute
strSQL = "DELETE FROM sales "
strSQL = strSQL & "WHERE ord_date < '1/1/93'"
qdf.SQL = strSQL
qdf.Execute
Exit Sub
ErrorHandler:
MsgBox "Error " & Err & ": " & Error Exit Sub
End Sub
```

Вызов хранимых процедур с помощью DAO сходен с созданием запросов к серверу. Вместо строки SQL для свойства SQL используется название существующей хранимой процедуры. Если хранимая процедура возвращает записи, необходимо открыть

выборку на основе QueryDef, указывающего на сохраненную процедуру. Для хранимых процедур, которые не возвращают значения, можно использовать метод Execute. Пример:

```
Private Sub cmbNew_Click()
On Error GoTo Err_cmbNew_Click

Dim dbs As Database, rst As Recordset, qdf As QueryDef
Dim strSQL As String
Dim Kod As String, Street As String

Set dbs = CurrentDb
Set qdf = dbs.CreateQueryDef("")
qdf.Connect = strCn

Kod = "0001"
Kod = InputBox("Код улицы", "Ввод значения", Kod)
If IsEmpty(Kod) = False Then
Street = "Новая улица"
Street = InputBox("Наименование улицы", " Ввод значения", Street)
If IsEmpty(Street) = False Then
strSQL = "exec dbo.insert_Улицы " & Kod & "," & Street
qdf.ReturnsRecords = False
qdf.SQL = strSQL
qdf.Execute
End If
End If

Exit_cmbNew_Click:
Exit Sub
Err_cmbNew_Click:
MsgBox Err.Description
Resume Exit_cmbNew_Click
End Sub
```

Здесь переменная strCn – глобальная переменная (объявленная в модуле) строки подключения.