ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение
профессионального высшего образования
«Уральский государственный университет им. А.М. Горького»

Математико-механический факультет

Кафедра информатики и процессов управления

Методические указания

Автоматизированные базы данных
(Основы Transact SQL и примерные практические работы)

Екатеринбург
2007

Автор (составитель, разработчик)

Стихина Татьяна Кабдешевна, к.ф.-м.н., доцент кафедры информатики и процессов управления, Уральский государственный университет им.А.М.Горького

Рекомендовано к изданию учебно-методической комиссией математико-механического факультета протокол заседания
№ _____от _____

Содержание

Введение

Методические указания составлены в соответствии с планом работ по ИОНЦ «Бизнес-информатика» в 2007 году. Данная разработка предназначена для организации самостоятельной и индивидуальной работы студентов по изучению вопросов проектирования и построения баз данных. Цель данной работы предоставить методические рекомендации по изучению дисциплины «Автоматизированные базы данных» и выработке практических навыков построения проекта и реализации его в среде СУБД MS SQL . Студентам предлагаются основные операторы языка данной СУБД и примеры их использования, а также лабораторные задания, позволяющие на заданной предметной области проверить методику проектирования и построения базы данных. Часть этих заданий являются обязательными (помечены *), другие могут использоваться для индивидуальных работ.

Основные конструкции языка Transact SQL.

ALTER DATABASE

Инструкция позволяющая модифицировать базу данных после ее создания.

ALTER DATABASE *database*
{ ADD FILE < filespec > [**,***...n*] [TO FILEGROUP *filegroup_name*]
| ADD LOG FILE < filespec > [**,***...n*]
| REMOVE FILE *logical_file_name*
| ADD FILEGROUP *filegroup_name*
| REMOVE FILEGROUP *filegroup_name*
| MODIFY FILE < filespec >
| MODIFY NAME = *new_dbname*
| MODIFY FILEGROUP *filegroup_name* {*filegroup_property* | NAME =
new_filegroup_name }
| SET < optionspec > [**,***...n*] [WITH < termination >]
| COLLATE < *collation_name* >
}

Пример. Добавление к ранее созданному файлу еще двух по 5 мб.

```
USE master
GO
ALTER DATABASE Test1
ADD FILEGROUP Test1FG1
GO

ALTER DATABASE Test1
ADD FILE
( NAME = test1dat3,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat3.ndf',
  SIZE = 5MB,
```

```
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB),
( NAME = test1dat4,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat4.ndf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB)
TO FILEGROUP Test1FG1

ALTER DATABASE Test1
MODIFY FILEGROUP Test1FG1 DEFAULT
GO
```

ALTER TABLE **инструкция модификации ранее созданной таблицы-отношения.**

```
ALTER TABLE [database.[owner].]table_name
[WITH NOCHECK]
[ADD
 {col_name column_properties column_constraints]
 |[[,]table_constraint]}
   [,{next_col_name|next_table_constraint}]...]
| DROP [CONSTRAINT]
 constraint_name[,constraint_name]
```

Модифицирует таблицу после её создания.

Пример

Добавление PRIMARY KEY CONSTRAINT:

```
  ALTER TABLE teachers
  ADD
CONSTRAINT [pk_teach] PRIMARY KEY CLUSTERED (au_id)
```

Добавление FOREIGN KEY CONSTRAINT:

```
  ALTER TABLE titles
  ADD
CONSTRAINT FK_pub_id FOREIGN KEY (pub_id) REFERENCES
publishers(pub_id)
```

Добавление UNIQUE CONSTRAINT:

```
  ALTER TABLE titles
  ADD
CONSTRAINT UNC_name_city UNIQUE NONCLUSTERED (stor_name,city)
```

Добавление DEFAULT CONSTRAINT:

```
 ALTER TABLE authors
 ADD
DEFAULT 'UNKNOWN' FOR phone
```

Будьте внимательны с default!

Добавление CHECK CONSTRAINT:

```
 ALTER TABLE authors
 ADD
CONSTRAINT CK_zip CHECK (zip LIKE '[0-9][0-9][0-9][0-9][0-9]')
```

Добавление новой колонки :

```
 ALTER TABLE publishers
 ADD
 country varchar(30)      NULL
DEFAULT('USA')
```

Batches

Batch - это набор операторов TSQL, передаваемых на выпонение и выполняющихся вместе, как одно целое. Batch компилируется целиком и оканчивается специальным символом-сигналом конца (go). Все последовательности операторов TSQL, набираемые в ISQL/w или в Enterprise Manager, Server management studio интерпретируются именно как batch'и

(интересно то, что при выделении некоторой части текста в окне выполняться будет именно она)

Пример

Несколько SELECT в одном batch'e

```
SELECT COUNT(*) FROM titles
SELECT COUNT(*) FROM authors
```

Выражения CASE

Простое CASE Expession

```
CASE expression
WHEN expression1 THEN exression1
[[WHEN expression2 THEN expression2[..]]
[ELSE expressionN]
END
```

CASE Expression с поиском

```
CASE
 WHEN Boolean_expression1 THEN expression1
 [[WHEN Boolean_expression2 THEN expression2[..]]
 [ELSE expressionN]
 END
```

Функции, полезные для CASEобразия:

```
COALESCE(expression1,expression2,...)
NULLIF(expression1,expression2)
ISNULL(expression1, expression)
```

COALESCE возвращает первое не-NULL выражение из списка, NULLIF возвращает NULL, если два выражения равны, ISNULL возвращает *expression2* в том случае, если *expression1* 'is null'

Примеры

```
SELECT Category=
  CASE type
      WHEN 'popular_comp' THEN 'Popular Computing'
      WHEN 'mod_cook' THEN 'Modern Cooking'
      WHEN 'business' THEN 'Businness'
      WHEN 'psyhology' THEN 'Psyhology'
      WHEN 'trad_cook' THEN 'Traditional Cooking'
      ELSE 'Not yet categorized'
    END,
    'Shortended Tiitle' = CONVERT(varchar(30), title),
    Price = price
FROM titles
WHERE price IS NOT NULL
ORDER BY type
COMPUTE AVG(price) BY TYPE
go
```

Результаты будут представлены в таблице:

Category	Shortedned Title	Price
Business	Cooking with Computers: Surrep	11.95
Business	Straight Talkk About Computers	19.99
Business	The Busy Executive's Database	19.99
Business	You Can Combat Computer Stress	2.99

avg
13.73

Поток управления

Порядок выполнения запросов и хранимых процедур TSQL может изменяться с помощью определенных операторов (как бы перевести Control-of-Flow Language)

Оператор	**Описание**
BEGIN...END	Определяет блок. BEGIN {*sql_statement*|*statement_block*} END
GOTO *label*	Безусловный переход к метке *label* Метки описываются незамысловато: :*label* GOTO *label*
IF...ELSE	Условный оператор. Тоже ничего неожиданного IF *Boolean_expression* *{sql_statement|statement_block}* *[ELSE [Boolean_expression]* *{sql_statement|statement_block}*
RETURN	Безусловный выход. Синтаксис бесхитростен: RETURN ([*integer_expression*])
WAITFOR	Ожидание определенного события. WINTFOR {DELAY '*time*'|TIME '*time*'}

	DELAY - определение задержки *time*, ну а TIME - ожидание до указанного времени. *time* задается аналогично значениям для **datetime**
WHILE	Цикл с предусловием. WHILE *Boolean_condition* {*sql_statement*|*statement_block*} [BREAK] {*sql_statement*|*statement_block*} [CONTINUE]
...BREAK	Выход из цикла WHILE
...CONTINUE	Продолжение цикла WHILE

Кроме вышеуказанных операторов могут оказаться полезными при написании запросов и хранимых процедур следующие возможности TSQL:

Возможность	**Описание**
CASE	Позволяет выражениям принимать значение в зависимости от условий. Между прочим, CASE является стандартной возможностью ANSI SQL-92
Комментарии	Очень даже полезно хотя бы изредка комментировать свой код. Для комментариев можно использовать две формы - первая, аналогичная комментариям в C: /*некий глубокомысленный комментарий*/, и вторая, аналогичная комментариям в Аде: -- Еще один заумный комментарий
Оператор DECLARE	Позволяет объявлять локальные переменные. К слову сказать, он может стоять не только в начале процедуры или batch'a, но и где угодно в их теле. Удобно объявлять переменные там, где они используются, а не тремястами

	строками выше. Впрочем, злоупотреблять этим тоже не стоит...
Оператор PRINT	Выдает заданное значение на экран.
Оператор RAISERROR	Устанавливает ошибочное состояние

Пример

```
DECLARE @Name varchar(40)
DECLARE CT CURSOR FOR SELECT Name from Peoples
OPEN CT
WHILE 1=1 BEGIN
 FETCH FROM CT INTO @Name
 /*Так, на мой взгляд, удобнее обходиться с курсорами*/
 IF @@fetch_status=-1
    BREAK
 IF @@fetch_status=-2
    CONTINUE
 PRINT @Name
END
DEALLOCATE CT
```

CREATE DATABASE создание базы данных

```
CREATE DATABASE database_name
[ON [DEFAULT|database_device][= size]
     [,database_device][= size]]...]
[LOG ON database_device[= size]
     [,database_device[= size]...]
[FOR LOAD]
```

Создает БД и, возможно, журнал транзакций на указанных *devices*, *size* - размер в мегабайтах. При создании новой базы данных используется как образец БД model.

Пример

```
CREATE DATABASE sales
ON DEVICE3 = 125
LOG ON DEVICE4 = 60
```

CREATE INDEX

```
CREATE [UNIQUE][CLUSTERED|NONCLUSTERED]INDEX index_name
 ON [[database.]owner.]table_name(column_name[,column_name]...)
[WITH]
 [FILLFACTOR = x]
 [[,] IGNORE_DUP_KEY]
 [[,] {SORTED_DATA | SORTED_DATA_REORG}]
 [[,] {IGNORE_DUP_ROW | ALLOW_DUP_ROW}]]
[ON segment_name]
```

Создает индекс для перечисленных колонок на указанном сегоменте.

CLUSTERED - создавать кластеризованный индекс, т.е. такой индекс, при котором в листьях B-дерева, образующего индекс, находятся не ссылки на данные, а собственно страницы данных.

FILLFACTOR - позволяет управлять заполнением страниц B-дерева индекса, задается в процентах, 100% - полное заполнение.

Пример

```
CREATE UNIQUE CLUSTERED INDEX au_id_ind
   ON authors (au_id)
```

Типы данных

Типы данных определяют представление колонок таблиц, параметров процедур и переменных. В SQL Server помимо предопределенных системных типов данных можно создавать и пользовательские типы, основывающиеся на системных. Более подробно о создании пользовательских типов данных можно ознакомиться в разделе, посвященном хранимой процедуре sp_addtype. В SQL Server все типы данных регистро-независимые, в силу чего недопустимо использование различных пользовательских типов данных, отличающихся только регистром.

Существуют следующие системные типы данных:

Вид данных	Системное представление
Двоичные	*binary*[(n)] *varbinary*[(n)]
Символьные	*char*[(n)] *varchar*[(n)]
Дата и время	*datetime* *smalldatetime*
Точное представление чисел	*decimal*[(p[, s])] *numeric*[(p[, s])]
Представление чисел с плавающей точкой	*float*[(n)] *real*
Целочисленные типы	*int* - 4 байта *smallint* - 2 байта *tinyint* -1 байт
Денежные типы	*money* *smallmoney*

Специальные	*bit* *timestamp* *типы*, определяемые пользователем
Текст и картинки	*text* *image*
Синонимы	*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*

Типы данных даты и времени

Данные и время представляются алфавитно-цифровыми данными, в виде строки. По умолчанию для показа даты используется формат Mon dd yyyy hh:mmAM, например, "Apr 15 1996 10:23AM". При вводе данных следует обращать внимание на порядок лет, месяцев, дней и т.п.

При вводе данных используйте один из нескольких форматов, заключая значение в одиночные кавычки - " ' ". Если требуется получить секунды или миллисекунды - для этого применяется функция CONVERT. Существуют следующие типы даты и времени:

datetime

Этот тип данных имеет размер в 8 байт, т.е. два четырехбайтных целых - 4 байта на количество дней, прошедших или еще не наступивших с 1 января 1900, и 4 байта на число миллисекунд, прошедших с полуночи.

datetime может содержать даты с 1 января 1753 года и по 31 декабря 9999

года, с точностью в три тысячных секунды. По умолчанию datetime имеет значение 1 января 1900 года, полдень.

smalldatetime

Тип данных, во многом аналогичный datetime, но менее точный. Размер его - 4 байта, два байта на число дней, прошедших с 1 января 1900 года, и два байта на число минут с полуночи. Даты могут быть представлены в диапазоне с 1 января 1900 года и по 6 июня 2079 года, с точностью в минуту Для ввода дат и времени можно применять следующие форматы:

Apr[il] [15][,] 1996
Apr[il] 15[,] [19]96
Apr[il] 1996 [15]
Apr[il] [19]96 15
[15] Apr[il][,] 1996
15 Apr[il][,][19]96
15 [19]96 apr[il]
[15] 1996 apr[il]
1996 APR[IL] [15]
[19]96 APR[IL] 15
1996 [15] APR[IL]

[0]4/15/[19]96 (mdy)
[0]4-15-[19]96 (mdy)
[0]4.15.[19]96 (mdy)
[04]/[19]96/15 (myd)

15/[0]4/[19]96 (dmy)
15/[19]96/[0]4 (dym)
[19]96/15/[0]4 (ydm)
[19]96/[04]/15 (ymd)

Но, наверное, самым удобным и безопасным является формат [19]960415 - строка из шести или восьми цифр, в формате ггггммдд или ггммдд. Строка из четырех цифр будет интерпретирована как год.

Денежные типы данных

Типы данных *money* и *smallmoney* предназначены в первую очередь для представления денег.

money

Тип данных *money* в состоянии представлять числа в диапазоне от -922,337,203,685,477.5808 до +922,337,203,685,477.5807 с точностью в одну десятитысячную, имеет размер в 8 байт.

smallmoney

Тип данных *smallmoney* в состоянии представлять числа в диапазоне от -214,748.3648 до +214,748.3647, размер 4 байта.

CREATE PROCEDURE создание процедуры

CREATE PROCedure [*owner.*]*procedure_name*[;*number*]
  [(*parameter1* [, *parameter2*]...[*parameter255*])]
[{FOR REPLICATION} | {WITH RECOMPILE}
  [{[WITH] | [,]} ENCRYPTION]]
AS *sql_statements*

parameter=
 @parameter_name datatype[=default][OUTPUT]

Создает процедуру с указанным именем. Процедура может быть создана только в текущей базе данных, за исключением временных процедур, которые создаются в *tempdb*. Для создания временных процедур следует начинать ее имя с '#' или '##'. Длина имени хранимой процедуры вместе с ## не может превышать 20 символов. Одна процедура может вызывать другую процедуру, уровень вложенности не может превышать 16, текущий уровень вложенности можно узнать из глобальной переменной @@NESTLEVEL

Пользователь может создавать свои системные процедуры; они начинаются с символов **sp_**. При попытке выполнения такой процедуры она сначала ищется в текущей базе данных, в случае же неудачи - в базе данных **master**. Таблицы, используемые в системной процедуре, определяемой пользователем, также сначала отыскиваются в текущей базе данных, и если это не удалось - в базе данных **master**.

Пример

```
CREATE PROCEDURE au_info_all
AS
SELECT au_lname, au_fname, title, pub_name
   FROM authors a, titles t, publishers p, titleauthor ta
      WHERE a.au_id = ta.au_id
      AND t.title_id = ta.title_id
      AND t.pub_id = p.pub_id
```

CREATE RULE

CREATE RULE [*owner*.]*rule_name*
AS *condition_expression*

Создает правило - некоторый объект, который впоследствие может быть привязан к какой-либо колонке или типу, определяемому пользователем.

Вы можете привязать правило к колонке, к которой до этого уже было привязано какое-либо другое правило; после этого уже будет использоваться вновь привязанное правило. Более того, привязка правила имеет больший приоритет, чем тип данных, определяемый пользователем.

Пример

```
CREATE RULE list_rule
AS
@list IN ('1389', '0736', '0877')
```

CREATE TABLE

CREATE TABLE [*database*.[*owner*].]*table_name*

(

{*col_name column_properties* [*constraint* [*constraint* [*...constraint*]]]

| [[,] *constraint*]}

[[,] {*next_col_name* | *next_constraint*}*...*]

)

[ON *segment_name*]

Создает таблицу

Кроме обычных таблиц можно также создавать и временные таблицы - такими таблицами являются те, чье имя начинается с #(локальные временные таблицы) или ##(глобальные временные таблицы). Временные таблицы существуют только на время клиентской сессии и после ее окончания автоматически уничтожаются. Кроме того, временные таблицы, созданные в хранимой процедуре автоматически уничтожаются после ее окончания. Временные таблицы создаются в базе данных tempdb, и вносятся в таблицу tempdb..sysobjects под указанным именен + некоторая строка, генереруемая сервером. При создании временных таблиц нельзя использовать ограничение FOREIGN KEY и ON *segment_name*

column_properties =

datatype [NULL | NOT NULL | IDENTITY[(seed, increment)]]

datatype

Опрелеляет тип создаваемой колонки - как системный, так и определяемый пользователем.

IDENTITY[(seed, increment)]

Для колонки с таким свойством сервером автоматически генерируется возрастающая последовательность, начиная с seed и приращением increment.

Может включать в себя ограничение как для столбца, так и для всей таблицы. Всего на таблицу может быть не более одного PRIMARY KEY, не более чем 249 UNIQUE, не более чем 31 FOREIGN KEY (каждый из которых может ссылаться не более чем на 16 колонок), не более одного DEFAULT на колонку, и неограниченное число CHECK. Все эти ограничения могут находиться в одном операторе CREATE TABLE. Синтаксис этих ограничений таков:

PRIMARY KEY:

[CONSTRAINT *constraint_name*]
  PRIMARY KEY [CLUSTERED | NONCLUSTERED]
    (*col_name* [, *col_name2* [..., *col_name16*]])
    [ON *segment_name*]

UNIQUE:

[CONSTRAINT *constraint_name*]
  UNIQUE [CLUSTERED | NONCLUSTERED]
    (*col_name* [, *col_name2* [..., *col_name16*]])
    [ON *segment_name*]

FOREIGN KEY:

[CONSTRAINT *constraint_name*]
  [FOREIGN KEY (*col_name* [, *col_name2* [..., *col_name16*]])]
    REFERENCES [*owner*.]*ref_table* [(*ref_col* [, *ref_col2*
      [..., *ref_col16*]])]

DEFAULT:

[CONSTRAINT *constraint_name*]
  DEFAULT {*constant_expression* | *niladic-function* | NULL}
    [FOR *col_name*]

CHECK:
[CONSTRAINT *constraint_name*]
CHECK [NOT FOR REPLICATION] (*expression*)
PRIMARY KEY [CLUSTERED | NONCLUSTERED]
Определяет первичный ключ таблицы и тип индекса, который будет для него построен.

UNIQUE [CLUSTERED | NONCLUSTERED]
Определяет ограничение уникальности для колонки, и указывает тип индекса, создаваемого для этого. Хотя для UNIQUE и можно использовать поля, допускающие значения типа NULL, все же рекомендуется этого не делать.
По умолчанию создается некластеризованный индекс.
[FOREIGN KEY (*col_name* [, *col_name2* [*...*, *col_name16*]])]
REFERENCES [*owner*.]*ref_table* [(*ref_col*
[, *ref_col2* [*...*, *ref_col16*]])]
Создает ограничение FOREIGN KEY для таблицы. Число полей и их тип должны совпадать. FOREIGN KEY не может ссылаться на таблицу, находящуюся в другой базе данных.

DEFAULT
Указывает значение, используемое по умолчанию. Это должно быть либо константное выражение, в котором допустимо использование т.н. *niladic* функций. Это
USER
CURRENT_USER
SESSION_USER
SYSTEM_USER
CURRENT_TIMESTAMP

DEFAULT может использоваться для колонок любых типов, кроме timestamp или тех, для которых указано IDENTITY.

CHECK(*expression*)

Задает условие проверки

Указанное *expression* должно вычисляться в булево выражение, и если оно принимает значение FALSE - попытка добавления или изменения данных отвергается. При создании таблицы, возможно, указать только один CHECK, в дальнейшем же их число можно увеличить.

ON *segment_name*

Указывает сегмент, который будет использоваться для таблицы.

Пример

```
CREATE TABLE employee
(
   emp_id  empid
      CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED
      CONSTRAINT CK_emp_id CHECK (emp_id LIKE
         '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]' or
         emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]'),
      /* Each employee ID consists of three characters that
      represent the employee's initials, followed by a five
      digit number ranging from 10000 to 99999 and then the
      employee's gender (M or F). A - (hyphen) is acceptable
      for the middle initial. */
   fname   varchar(20)     NOT NULL,
   minit   char(1) NULL,
   lname   varchar(30)     NOT NULL,
   job_id  smallint        NOT NULL
      DEFAULT 1
      /* Entry job_id for new hires. */
```

```
    REFERENCES jobs(job_id),
  job_lvl tinyint
    DEFAULT 10,
    /* Entry job_lvl for new hires. */
  pub_id  char(4) NOT NULL
    DEFAULT ('9952')
    REFERENCES publishers(pub_id),
    /* By default the Parent Company Publisher is the company
    to whom each employee reports. */
  hire_date      datetime       NOT NULL
    DEFAULT (getdate())
    /* By default the current system date will be entered. */
)
```

CREATE TRIGGER

```
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR {INSERT, UPDATE, DELETE}
[WITH ENCRYPTION]
AS sql_statements
```

Или используя предложение IF UPDATE

```
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR {INSERT, UPDATE}
[WITH ENCRYPTION]
AS
IF UPDATE (column_name)
[{AND | OR} UPDATE (column_name)...] sql_statements
```

Этот оператор создает триггер - специальную разновидность хранимой процедуры, которая выполняется в тех случаях, когда пользователь пытается

добавить, удалить или модифицировать данные. Триггеры часто используются для реализации бизнес-логики и проверки целостности данных.

Таблицы INSERTED и DELETED

Когда триггер срабатывает и начинает выполняться, во время его выполнения существуют две специальные таблицы - INSERTED и DELETED. В них находятся записи, соответсвенно добавляемые или удаляемые. Глобальная переменная @@ROWCOUNT указывает на число записей, участвующих в операциях с данными.

Невозможно создать триггер на VIEW.

Пример

```
CREATE TRIGGER employee_insupd
ON employee
FOR INSERT, UPDATE
AS
/* Get the range of level for this job type from the jobs table. */
DECLARE @min_lvl tinyint,
   @max_lvl tinyint,
   @emp_lvl tinyint,
   @job_id smallint
SELECT @min_lvl = min_lvl,
   @max_lvl = max_lvl,
   @emp_lvl = i.job_lvl,
   @job_id = i.job_id
FROM employee e, jobs j, inserted i
WHERE e.emp_id = i.emp_id AND i.job_id = j.job_id
IF (@job_id = 1) and (@emp_lvl <> 10)
BEGIN
```

```
   RAISERROR ('Job id 1 expects the default level of 10.',16,-1)
   ROLLBACK TRANSACTION
END
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
BEGIN
   RAISERROR ('The level for job_id:%d should be between %d and %d.',
      16, -1, @job_id, @min_lvl, @max_lvl)
   ROLLBACK TRANSACTION
END
```

CREATE VIEW

CREATE VIEW [*owner*.]*view_name*
[(*column_name* [, *column_name*]...)]
[WITH ENCRYPTION]
AS *select_statement* [WITH CHECK OPTION]

Создает виртуальную таблицу(представление), воспроизводящую данные из одной или более реальных таблиц. Для создания VIEW невозможно использовать оператор UNION, не разрешаются операции вставки в представление, если во VIEW существуют вычислимые поля.

Пример

```
CREATE VIEW titles_view
AS
SELECT title, type, price, pubdate
   FROM titles
```

Курсоры

Курсоры позволяют обрабатывать данные для каждой возвращаемой строки отдельно, не пользуясь множественными, традиционными для SQL

операциями. Эти курсоры - серверные (server-based), и их не следует путать с курсорами, предоставляемыми DB-Library или ODBC. Для работы с курсорами используются следующие операторы:

Оператор	**Описание**
DECLARE CURSOR	Создает курсор
OPEN	Открывает курсор
FETCH	Выбирает данные
CLOSE	Закрывает ранее открытый курсор
DEALLOCATE	Уничтожает ранее созданный курсор

Для обновления текущей сроки используется следующая форма оператора UPDATE:

UPDATE *table_name*

SET *column_name1* =

{*expression1* | NULL | (*select_statement*)}

[, *column_name2* =

{*expression2* | NULL | (*select_statement*)}...]

WHERE CURRENT OF *cursor_name*

а для удаления - следующая:

DELETE FROM *table_name*

WHERE CURRENT OF *cursor_name*

Курсор объявляется оператором DECLARE:

DECLARE *cursor_name* [INSENSITIVE] [SCROLL] CURSOR

FOR *select_statement*

[FOR {READ ONLY | UPDATE [OF *column_list*]}]

Оператор OPEN служит для открытия курсора. Он имеет следующий синтаксис: OPEN *cursor_name* После открытия курсора в переменной @@CURSOR_ROW может быть одно из следующих значений:

-*m*

В этом случае данные для курсора выгребаются асинхронно. Значение *m* представляет число уже выбранных записей.

n

Все данные получены, *n* - число записей

0

Курсор не открыт

Оператор FETCH служит для получения данных. Он имеет следующий синтаксис:

FETCH [[NEXT | PRIOR | FIRST | LAST | ABSOLUTE *n* | RELATIVE *n*] FROM] *cursor_name*

[INTO *@variable_name1*, *@variable_name2*, ...]

NEXT

Получает следующую запись

PRIOR

Получает предыдущую запись

FIRST

Получает первую запись

LAST

Получает последнюю запись

ABSOLUTE

Получает *n*-ю запись

RELATIVE

Получает *n*-ю запись относительно текущей

Для того, чтобы опции PRIOR, LAST, FIRST, ABSOLUTE и RELATIVE были доступны, необходимо, чтобы курсор был создан с опцией SCROLL.

Глобальная переменная @@FETCH_STATUS возвращает результат последнего fetch'a -

0

Все нормально, данные получены

-1

Курсор кончился

-2

Текущая строка не является более членом требуемого множества записей, заданного в операторе select при объявлении курсора.

Оператор CLOSE закрывает ранее созданный курсор, однако структуры данных, необходимых для его функционирования остаются нетронутыми; таким образом, курсор позднее может быть открыт вновь.

Оператор DEALLOCATE закрывает курсор и уничтожает все связанные с ним структуры данных.

Example

```
DECLARE @Name varchar(40)
DECLARE CT CURSOR FOR SELECT Name from Peoples
OPEN CT
WHILE 1=1 BEGIN
 FETCH FROM CT INTO @Name
 /*Так, на мой взгляд, удобнее обходиться с курсорами*/
 IF @@fetch_status=-1
    BREAK
 IF @@fetch_status=-2
    CONTINUE
 PRINT @Name
END
DEALLOCATE CT
```

DECLARE

DECLARE *@variable_name datatype*

[, *@variable_name datatype*...]

Создает локальные переменные для batch'a или процедуры. Также служит для объявления курсоров.

Пример

DECLARE @I INT

DELETE

DELETE [FROM] {*table_name* | *view_name*}

[WHERE *clause*]

Где *clause* - {*search_conditions* | CURRENT OF *cursor_name*}

Ключевое слово IDENTITYCOL может быть использовано вместо имени колонки, имеющей свойство IDENTITY. Оператор TRUNCATE TABLE работает быстрее, чем DELETE, хотя и делает то же самое. Дело в том, что если DELETE удаляет по одной строке, занося данные об этом в журнал транзакций, то TRUNCATE TABLE заносит в журнал только данные об удалении страниц.

В дополнение к стадартному синтаксису Transact-SQL имеет расширение:

DELETE [FROM] {*table_name* | *view_name*}

[FROM {*table_name* | *view_name*}

[, {*table_name* | *view_name*}]...]

[..., {*table_name16* | *view_name16*}]]

[WHERE *clause*]

что позволяет использовать при удалении более одной таблицы в WHERE

Пример

```
DELETE FROM titleauthor
   FROM authors a, titles t
      WHERE a.au_id = titleauthor.au_id
      AND titleauthor.title_id = t.title_id
      AND t.title LIKE '%computers%'
```

DROP DATABASE

DROP DATABASE *database_name* [, *database_name*...]

Уничтожает базу данных - то есть уничтожает все объекты, относящиеся к ней, освобождает распределенное для нее место на диске и уничтожает все ссылки на нее из master'a. Невозможно уничтожить базу данных, кем-либо используемую в данный момент.

Пример

DROP DATABASE pubs, newpubs

DROP TABLE

DROP TABLE [[*database*.]*owner*.]*table_name*

[, [[*database*.]*owner*.]*table_name*...]

Уничтожает таблицу. Невозможно уничтожить таблицу, на которую ссылаются другие таблицы с помощью FOREIGN KEY, кроме того, невозможно уничтожить системную таблицу. Не следует путать операторы DROP TABLE, DELETE и TRUNCATE TABLE - в то время как первый из них уничтожает таблицу, то второй и третий просто удаляют из нее данные.

Пример

DROP TABLE titles1

DROP TRIGGER

DROP TRIGGER [*owner*.]*trigger_name* [, [*owner*.]*trigger_name*...]

Уничтожает указанный триггер

Пример

DROP TRIGGER employee_insupd

DROP VIEW

DROP VIEW [*owner*.]*view_name* [, [*owner*.]*view_name*...]

Уничтожает указанный VIEW

Пример

DROP VIEW titles_view

EXECUTE выполнение процедуры

EXEC[ute]

{[*@return_status* =]

{[[[*server*.]*database*.]*owner*.]*procedure_name*[;*number*] |

@procedure_name_var}

[[*@parameter_name* =] {*value* | *@variable* [OUTPUT]]

[, [*@parameter_name* =] {*value* | *@variable* [OUTPUT]}]...]

[WITH RECOMPILE]]

EXEC[ute] ({*@str_var* | '*tsql_string*'} [{*@str_var* | '*tsql_string*'}...)}

Выполняет указанную хранимую процедуру с указанными параметрами, возвращая код возврата. Кроме процедур также можно ивыполнять и строку симоволов

Пример

Выражения

constant | *column_name* | *function* | (*subquery*)}

[{*operator* | AND | OR | NOT}

{*constant* | *column_name* | *function* | (*subquery*)}...]

Используются вместе с константами, переменными и именами колонок в большинстве операторов TSQL. Строковые константы представляются как

имеющие тип *varchar*, все выражения, имеющие тип *varchar* при сравнении с выражениями типа char преобразуются в него же, т.е. к ним добавляются дополнительные пробелы

Функции

Аггрегативные функции

Аггрегативные функции возвращают суммарные значения. Вот список этих функций:

AVG	COUNT(*)	MIN
COUNT	MAX	SUM

Функции манипуляции датой и временем:

DATEADD	DATENAME	GETDATE
DATEDIFF	DATEPART	

Математические функции

ABS	DEGREES	RAND
ACOS	EXP	ROUND
ASIN	FLOOR	SIGN
ATAN	LOG	SIN
ATN2	LOG10	SQRT
CEILING	PI	TAN
COS	POWER	
COT	RADIANS	

Niladic-функции

Эти функции возвращают различные системные значения.

CURRENT_TIMESTAMP	SYSTEM_USER
CURRENT_USER	USER

SESSION_USER

Функции для манипуляции со строками

LTRIM	SOUNDEX	
ASCII	PATINDEX	SPACE
CHAR	REPLICATE	STR
CHARINDEX	REVERSE	STUFF
DIFFERENCE	RIGHT	SUBSTRING
LOWER	RTRIM	UPPER

Системные функции

COALESCE	HOST_NAME	OBJECT_NAME
COL_LENGTH	IDENT_INCR	STATS_DATE
COL_NAME	IDENT_SEED	SUSER_ID
DATALENGTH	INDEX_COL	SUSER_NAME
DB_ID	ISNULL	USER_ID
DB_NAME	NULLIF	USER_NAME
GETANSINULL	HOST_ID	OBJECT_ID

CONVERT

Функция CONVERT служит для преобразования различных типов данных.

GRANT

Полномочия на выполнение операторов:

GRANT {ALL | *statement_list*}

TO {PUBLIC | *name_list*}

Полномочия на объекты:

GRANT {ALL | *permission_list*}

ON {*table_name* [(*column_list*)] | *view_name* [(*column_list*)] |

stored_procedure_name | *extended_stored_procedure_name*}

TO {PUBLIC | *name_list*}

Назначает полномочия пользователям.

По умолчанию полномочия даются системному администратору, владельцу базы данных, владельцу объекта или группе PUBLUC. Некторые из них они могут передать другим пользователям. По умолчанию системный администратор имеет все полномочия, он может использовать хранимую процедуру **sp_changedbowner** для изменения владельца базы данных. Владелец базы данных или системный администратор могут передавать полномочия на сам оператор GRANT.

Пользователи могут входить только в одну группу, исключая группу PUBLIC.

Для изменения прав кроме оператора GRANT существует оператор REVOKE, имеющий следующий синтаксис:

Полномочия на выполнение операторов:

REVOKE {ALL | *statement_list*}

FROM {PUBLIC | *name_list*}

Полномочия на объекты:

REVOKE {ALL | *permission_list*}

ON {*table_name* [(*column_list*)] | *view_name* [(*column_list*)] |

stored_procedure_name | *extended_stored_procedure_name*}

FROM {PUBLIC | *name_list*

Пример

```
GRANT SELECT
ON authors
TO public
go
GRANT INSERT, UPDATE, DELETE
ON authors
TO Mary, John, Tom
```

Идентификаторы

Идентификаторы в TSQL должны состоять из символов латинского алфавита, цифр или из символов _ ,@ , #. Дополнительно существуют следующие правила:

Все идентификаторы, начинающиеся с @, почитаются за локальные переменные.

Все идентификаторы, начинающиеся с #, считаются именами временный объектов.

В противном случае идентификаторы должны начинаться с символа латинского алфавита.

По умолчанию в имени объектов не могут встречаться пробелы, но, используя режим "Quoted identifiers", это можно обойти. Для более подробного ознакомления с режимимом "Quoted identifiers" см. Transact-SQL Reference.

Для имен объектов необязательно быть уникальными в базе данных, например, имена колонок и индексов должны быть уникальными только в пределах таблицы или представления(view), все же имена других объектов должны быть уникальными в пределах базы данных для каждого владельца. Любую колонку или таблицу можно уникально идентифицировать следующим составным именем - имя базы данных, имя владельца, имя таблицы или представления. Промежуточные значения - имя владельца может быть опущено, если это не приводит к конфликтам имен. В случае удаленных хранимых процедур ее имя задается следующим образом: *server.database.owner.procedure*

Если вы указываете имя объекта не целиком, то сервер сначала пытается найти его среди объектов, которыми владеете вы, после этого производится попытка найти указанный объект как *database.dbowner.name*. Для определеня видимости хранимых процедур, начинающихся с символов **sp_** см. раздел, посвященный оператору CREATE PROCEDURE

INSERT

INSERT [INTO]

{*table_name* | *view_name*} [(*column_list*)]

{DEFAULT VALUES | *values_list* | *select_statement*}

Добавляет запись в таблицу.

При указании значений конкретных полей вместо использования каких-либо значений можно использовать ключевое слово DEFAULT

Вставка пустой строки приводит к добавлению пробела ' ', а не значения NULL

Пример

```
INSERT titles
   VALUES('BU2222', 'Faster!', 'business', '1389',
      NULL, NULL, NULL, NULL, 'ok', '06/17/87')

INSERT titles(title_id, title, type, pub_id, notes, pubdate)
   VALUES ('BU1237', 'Get Going!', 'business', '1389',
      'great', '06/18/86')



INSERT INTO newauthors
   SELECT *
      FROM authors
         WHERE city = 'San Francisco'
```

Ключевые слова

Следующие слова являются зарезервированными и не могут быть использованы для именования объектов в базе данных, за исключением режима Quoted Identifiers.

ADD	ALL	ALTER
AND	ANY	AS
ASC	AVG	BEGIN
BETWEEN	BREAK	BROWSE
BULK	BY	CASE
CHECK	CHECKPOINT	CLOSE
CLUSTERED	COALESCE	COMMIT
COMMITTED	COMPUTE	CONFIRM
CONSTRAINT	CONTINUE	CONTROLROW
CONVERT	COUNT	CREATE
CURRENT	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DBCC	DEALLOCATE
DECLARE	DEFAULT	DELETE
DESC	DISK	DISTINCT
DOUBLE	DROP	DUMMY
DUMP	ELSE	END
ERRLVL	ERROREXIT	EXCEPT
EXEC	EXECUTE	EXISTS
EXIT	FETCH	FILLFACTOR
FLOPPY	FOR	FOREIGN
FROM	GOTO	GRANT
GROUP	HAVING	HOLDLOCK
IDENTITY	IDENTITY_INSERT	IDENTITYCOL
IF	IN	INDEX

INSENSITIVE	INSERT	INTERSECT
INTO	IS	ISOLATION
KEY	KILL	LEVEL
LIKE	LINENO	LOAD
MAX	MIN	MIRROREXIT
NOCHECK	NONCLUSTERED	NOT
NULL	NULLIF	OF
OFF	OFFSETS	ON
ONCE	ONLY	OPEN
OPTION	OR	ORDER
OVER	PERM	PERMANENT
PIPE	PLAN	PRECISION
PREPARE	PRIMARY	PRINT
PROC	PROCEDURE	PROCESSEXIT
PUBLIC	RAISERROR	READ
RECONFIGURE	REFERENCES	REPEATABLE
REPLICATION	RETURN	REVOKE
ROLLBACK	ROWCOUNT	RULE
SAVE	SCROLL	SELECT
SERIALIZABLE	SESSION_USER	SET
SETUSER	SHUTDOWN	SOME
STATISTICS	SUM	SYSTEM_USER
TABLE	TAPE	TEMP
TEMPORARY	TEXTSIZE	THEN
TO	TRAN	TRANSACTION

TRIGGER	TRUNCATE	TSEQUAL
UNCOMMITTED	UNION	UNIQUE
UPDATE	UPDATETEXT	USE
USER	VALUES	VARYING
VIEW	WAITFOR	WHEN
WHERE	WHILE	WITH
WRITETEXT		

NULL

Значения типа NULL служат для представления пустых значений, не следует путать их с пустой строкой или с 0 в числовых полях, NULL подразумевает, что значение неизвестно или не определено.

Следует обратить внимание на то, что если вы создаете таблицу с колонками, допускающими NULL, то значение будут внутрисерверно преобразовываться в соответствии со следующей таблицей:

Тип данных	**Преобразуется в**
binary	varbinary
char	varchar
datetime	datetim
decimal	decimal
float	float
int	int
money	money
numeric	numeric
real	float
smalldatetime	datetim

smallint	int
smallmoney	money
tinyint	int

Аггрегативные функции игнорируют значения NULL, за исключением функции count(*). Значения NULL считаются дублирующимися при использовании в GROUP BY, ORDER BY, или при использовании DISTINCT.

Операции

SQL Server допускает следующие операции

Сложение	+
Вычитание	-
Умножение	*
Деление	/
Побитное И	&
Побитное ИЛИ	|
Побитное НЕ	~
Исключающее ИЛИ	^
Равно	=
Больше	>
Меньше	<
Больше или равно	>=
Меньше или равно	<=
Неравно	<> или != (нестандартно!)
Не больше	!>
Не меньше	!<

LEFT OUTER JOIN	*=
RIGHT OUTER JOIN	=*
Конкатенация строк	+

PRINT

PRINT {'*any ASCII text*' | *@local_variable* | *@@global_variable*}

Возвращает сообщение, определяемое пользователем, пользовательскому обработчику сообщений. Длина строки с сообщением не должна превышать 255 символов. Глобальной переменной в настоящее время можеть быть только переменная @@VERSION, так как только она является глобальной и одновременно символьной переменной.

Пример

```
IF EXISTS (SELECT zip
           FROM authors
              WHERE zip = '94705')
PRINT 'Berkeley author'
```

RAISERROR

RAISERROR ({*msg_id* | *msg_str*}, *severity*, *state*
[, *argument1* [, *argument2*]])
[WITH LOG]

Возвращает пользовательское сообщение об ошибке и устанавливает системный флаг, указывающий на то, что произошла ошибка. *severity* определяет важность ошибки, которые можно посмотреть в System Administration Companion. Только системный администратор может использовать RAISERROR с *severity* от 19 до 25

Пример

```
REATE TRIGGER employee_insupd
ON employee
FOR INSERT, UPDATE
AS
/* Get the range of level for this job type from the jobs table. */
DECLARE @min_lvl tinyint,
   @max_lvl tinyint,
   @emp_lvl tinyint,
   @job_id smallint
SELECT @min_lvl = min_lvl,
   @max_lvl = max_lvl,
   @emp_lvl = i.job_lvl,
   @job_id = i.job_id
FROM employee e, jobs j, inserted i
WHERE e.emp_id = i.emp_id AND i.job_id = j.job_id
IF (@job_id = 1) and (@emp_lvl <> 10)
BEGIN
   RAISERROR ('Job id 1 expects the default level of 10.',16,-1)
   ROLLBACK TRANSACTION
END
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
BEGIN
   RAISERROR ('The level for job_id:%d should be between %d and %d.',
      16, -1, @job_id, @min_lvl, @max_lvl)
   ROLLBACK TRANSACTION
END
```

REVOKE

Полномочия на выполнение операторов:

REVOKE {ALL | *statement_list*}

FROM {PUBLIC | *name_list*}

Полномочия на объекты базы данных:

REVOKE {ALL | permission_list}

ON {table_name [(column_list)] | view_name [(column_list)] |

stored_procedure_name | extended_stored_procedure_name}

FROM {PUBLIC | name_list}

Оператор REVOKE служит для отъятия различных прав у пользователей.

При определении прав они сохраняются в системной таблице **sysprotecs**.

Более подробно см. GRANT

Пример

REVOKE CREATE TABLE, CREATE DEFAULT

FROM Mary, John

Условия поиска

{WHERE | HAVING} [NOT] *Boolean_expression*

{WHERE | HAVING} [NOT] *Boolean_expression* {AND | OR}

Boolean_expression

{WHERE | HAVING} [NOT] *column_name* IS [NOT] NULL

{WHERE | HAVING} [NOT] *column_name join_operator column_name*

{WHERE | HAVING} [NOT] *column_name* [NOT] LIKE '*match_string*'

{WHERE | HAVING} [NOT] EXISTS (*subquery*)

{WHERE | HAVING} [NOT] *expression comparison_operator expression*

{WHERE | HAVING} [NOT] *expression* [NOT] BETWEEN *expression* AND

expression

{WHERE | HAVING} [NOT] *expression* [NOT] IN (*value_list* | *subquery*)

{WHERE | HAVING} [NOT] *expression comparison_operator* {ANY | ALL}

(*subquery*)

{WHERE CURRENT OF cursor_name}

Определяет условия поиска, используемые для получения или модификации данных. Если вы используете более одного условия, соединяйте их логическими операторами AND и OR. При использовании LIKE с датами будьте внимательны: SQL Server преобразует даты к стандартному виду для дат, и поэтому дата 01.01.1997 будет представлена в виде Jan 1, 1997 12:00AM. Так что придется искать с % по обеим сторонам шаблона.

SELECT

SELECT [ALL | DISTINCT] *select_list*
[INTO [*new_table_name*]]
[FROM {*table_name* | *view_name*}[(*optimizer_hints*)]
[[, {*table_name2* | *view_name2*}[(*optimizer_hints*)]
[..., {*table_name16* | *view_name16*}[(*optimizer_hints*)]]]
[WHERE *clause*]
[GROUP BY *clause*]
[HAVING *clause*]
[ORDER BY *clause*]
[COMPUTE *clause*]
[FOR BROWSE]

Производит выборку данных из БД или присваивает значения переменным, причем выборка данных и присваивание значений не могут производиться одновременно. При установленной опции **select into/bulkcopy** сервера, после ключевого слова INTO может идти имя новой таблицы. Она будет создана и в нее будут помещены данные, выбираемые этим оператором; если же эта опция не установлена, то в качестве таблицы-получателя может выступать только временная таблица.

Для более точной настройки пути выборки данных можно указывать т.н. optimizer hints - подсказки оптимизатору. Вот они:

INDEX = {*index_name* | *index_id*}

Указывает индекс, который будет использован для выборки данных из таблицы. Если указать вместо имени или идентификатора индекса 0, то будет произведено сканирование таблицы.

NOLOCK

Включает режим "грязного чтения",

HOLDLOCK

Указывает, что блокировки следует сохранять до конца транзакции.

UPDLOCK

Блокирует "блокировками обновления", а не разделяемыми блокировками, сохраняя их до конца транзакции.

TABLOCK

Блокирует таблицу в разделяющем режиме. Если используется вместе с HOLDLOCK, то блокировки сохраняются до конца транзакции.

PAGLOCK

Блокирует разделяемой блокировкой на страничном уровне в тех случаях, когда блокировка была бы на всю таблицу.

TABLOCKX

Блокирует таблицу в исключающем режиме до конца BATCH или до конца транзакции.

FASTFIRSTROW

Указывает оптимизатору, что необходимо использовать некластеризованный индекс (конечно, если существует подходящий) для выборки данных в порядке, задаваемом ORDER BY даже в том случае, если оптимизатор, сравнив затраты на выборку с помощью индекса и произвольной выборкой и последующей сортировкой находит более выгодым использование сортировки.

В одном операторе SELECT не может участвовать более 16 таблиц, включая подзапросы.

SET

SET {

{{{ANSI_NULL_DFLT_OFF | ANSI_NULL_DFLT_ON}

| ARITHABORT

| ARITHIGNORE

| FMTONLY

| FORCEPLAN

| IDENTITY_INSERT [*database*.[*owner*.]]*tablename*

| NOCOUNT

| NOEXEC

| OFFSETS {keyword_list}

| PARSEONLY

| PROCID

| QUOTED_IDENTIFIER

| SHOWPLAN

| STATISTICS IO

| STATISTICS TIME}

{ON | OFF}}

| DATEFIRST number

| DATEFORMAT format

| DEADLOCKPRIORITY {LOW | NORMAL}

| LANGUAGE language

| ROWCOUNT number

| TEXTSIZE number

| TRANSACTION ISOLATION LEVEL {READ COMMITTED | READ

UNCOMMITTED | REPEATABLE READ | SERIALIZABLE}}

Позволяет изменить различные установки сервера.

ANSI_NULL_DFLT_OFF

Устанавливает режим обработки значений NULL, несовместимый с ANSI SQL. Колонки во вновь создаваемых таблицах создаются как NOT NULL.

ANSI_NULL_DFLT_ON

Отменяет режим, устанавливаемый ANSI_NULL_DFLT_OFF

ARITHABORT

Останавливает запрос, если во время его выполнения встретилась арифметическая ошибка, такая как деление на ноль или переполнение

ARITHIGNORE

Указывает игнорировать возникающие при выполнении запроса арифметические ошибки. Если не установлена ни эта, ни предыдущая опция - сервер возвращает NULL вместо значений, при выполнении которых возникла ошибка.

FMTONLY

Возвращает только описание колонок, получающихся при выполнении запроса.

FORCEPLAN

Подавляет оптимизатор и указывает ему использовать порядок соединения таблиц, указываемый в запросе.

IDENTITY_INSERT [*database*.[*owner*.]]*tablename*

Разрешает вставку данных в колонку со свойством IDENTITY в указанной таблице.

NOCOUNT

Подавляет выдачу сообщений о количестве строк таблицы, получающейся в качестве запроса. Глобальная переменная @@ROWCOUNT обновляется даже когда установлена опция NOCOUNT.

NOEXEC

Указывает только скомпилировать запрос, но не выполнять его.

OFFSETS

Возвращает смещение keyword_list. Используется только вместе с DB-Library

PARSEONLY

Отменяет компиляцию и выполнение запроса, только проверяет синтаксис.

PROCID

Возвращает ID хранимой процедуры клиентской части запроса. Используется вместе с DB-Library.

QUOTED_IDENTIFIER

Указывает, что в текущей сессии ' и " различны. Строки в двойных кавычках (") считаются ключевыми словами или именами объектов.

SHOWPLAN

Возвращает вместе с данными план выполнения запроса.

STATISTICS IO

Сообщает статистику по вводу-выводу.

STATISTICS TIME

Сообщает статистику по времени.

DATEFIRST number

Устанавливает первый день недели. По умолчанию стоит американский формат - неделя начинается с воскресенья.

DATEFORMAT format

Устанавливает формат даты. Формат может быть mmddyy, ddmmyy, yymmdd, yyddmm, mmyydd, and ddyymm. По умолчанию стоит американский формат - mmddyy.

DEADLOCKPRIORITY {LOW | NORMAL}

Устанавливает режим остановки запросов в текущей сессии при взаимоблокировках. При LOW процесс считается наилучшей жертвой при их разрешении.

LANGUAGE language

Устанавливает язык сообщений об ошибках. По умолчанию, само собой, английский.

ROWCOUNT number

Ограничивает число строк таблиц, участвующих в запросе. Обратите внимание: - это относится ко всем таблицам, участвующим в запросе!

TEXTSIZE number

Устанавливает размер возвращаемый размер данных типа text. По умолчанию - 4 килобайта.

TRANSACTION ISOLATION LEVEL

Устанавливает уровень изолированности транзакций Допустимые значения:

READ COMMITED

READ UNCOMMITED

REPEATABLE READ | SERIAZABLE

Если вы используете оператор SET в хранимой процедуре, триггере или в BATCH'e, то все параметры, изменяемые им, изменяются им только в пределах процедуры/триггера или BATCH'a.

Подзапросы

expression comparison_operator [ANY | ALL | SOME] (subquery)expression [NOT] IN (*subquery*)[NOT] EXISTS (*subquery*)

(SELECT [ALL | DISTINCT] subquery_select_list

[FROM {*table_name* | *view_name*}[*optimizer_hints*]

[[, {*table_name2* | *view_name2*}[*optimizer_hints*]

[..., {*table_name16* | *view_name16*}[*optimizer_hints*]]]

[WHERE clause]

[GROUP BY clause]

[HAVING clause])

Подзапросы могут быть использованы помимо всего прочего и как выражения. **Пример**

```
declare @a varchar(80)
select @a=(select first_name from employee ...)+
        ' получает '+
       (select convert(sum(pay)) from pays...)
```

TRUNCATE TABLE

TRUNCATE TABLE [[database.]owner.]table_name

Уничтожает данные в таблице, но оставляет ее структуру и индексы нетронутыми, делает то же самое, что и DELETE FROM, только быстрее. Да, не забудьте сказать UPDATE STATISTICS после TRUNCATE TABLE.

Пример

TRUNCATE TABLE authors

UNION

SELECT select_list [INTO *clause*]
[FROM *clause*]
[WHERE *clause*]
[GROUP BY *clause*]
[HAVING *clause*]
[UNION [ALL]
SELECT *select_list*
[FROM *clause*]
[WHERE *clause*]
[GROUP BY *clause*]
[HAVING *clause*]...]
[ORDER BY *clause*]
[COMPUTE *clause*]

Объединяет результаты нескольких запросов в один. Имена колонок берутся из первого запроса. Колонки объединяются по порядку; они должны иметь одинаковый тип или приводится к одному типу. **Пример**

```
/* CORRECT */
SELECT cities = city FROM storeseast
UNION
```

SELECT city FROM stores

ORDER BY cities

UPDATE

UPDATE {*table_name* | *view_name*}

SET [{*table_name* | *view_name*}]

{*column_list*

| *variable_list*

| *variable_and_column_list*}

[, {*column_list2*

| *variable_list2*

| *variable_and_column_list2*}

... [, {*column_listN*

| *variable_listN*

| *variable_and_column_listN*}]]

[WHERE clause]

Этот оператор служит для обновления данных. Обратите внимание на то, что в если удается произвести т.н. update in-place, то производительность обновлений заметно возрастает. Это происходит при выполнении следующих условий:

Изменяемые колонки не входят в кластеризованный индекс

У таблицы не существует триггера на обновление

Для обновления одной строки:

Размер обновленной записи должен совпадать с размером не обновленной.

Если на изменяемые колонки построен индекс, то он должен быть неуникальным некластеризованным и колонки имеют фиксированную длину.

Обновляемые колонки могут находиться в некластеризованном уникальном индексе в том случае, если в условии WHERE было заданно точное соответствие, и для поиска обновляемой строки был использован именно этот индекс.

Transact-SQL имеет следующее расширение стандартного синтаксиса **UPDATE**:

UPDATE {*table_name* | *view_name*}
SET [{*table_name* | *view_name*}]
{*column_list*
| *variable_list*
| *variable_and_column_list*}
[, {*column_list2*
| *variable_list2*
| *variable_and_column_list2*}
... [, {*column_listN*
| *variable_listN*
| *variable_and_column_listN*}]]
[FROM {*table_name* | *view_name*}
[, {*table_name* | *view_name*}]...]
[..., {*table_name16* | *view_name16*}]]
[WHERE clause]

что позволяет несколько более изящно проделывать различные операции.

Ранее было невозможно одновременно получить старое значение колонки и присвоить ему новое, для этого приходилось использовать что-то вроде

BEGIN TRANSACTION
SELECT *variable_name = column_name1*
FROM *table_name*
WHERE *column_name2 = expression*
HOLDLOCK
UPDATE *table_name*
SET *column_name1 = expression*
WHERE *column_name2 = expression*
COMMIT TRANSACTION

теперь же можно поступать следующим образом:

UPDATE table_name

SET *column_name1 = expression*, *variable_name = column_name1*

WHERE *column_name2 = expression*

Пример

```
UPDATE titles
   SET ytd_sales =
      (select sum(qty)
         FROM sales
            WHERE sales.title_id = titles.title_id
               AND sales.date IN (SELECT MAX(date) FROM sales))
   FROM titles, sales
```

UPDATE STATISTICS

UPDATE STATISTICS [[database.]owner.]table_name [index_name]

Обновляет статистические данные о распределении ключей в индексе. Эти данные используются оптимизатором для поиска наиболее выгодного плана, поэтому весьма важно не забывать периодически обновлять статистику; впрочем, для автоматического обновления можно воспользоваться SQL Executive, а в Transact-SQL Reference приведен пример полезной хранимой процедуры, обновляющей статистические данные для всех пользовательских таблиц. Эти данные автоматически обновляются при перестроении индекса. Утилита SQLMAINT.EXE предоставляет интерфейс командной строки для того же.

USE

USE *database_name*

Изменяет текущую для данной сессии базу данных.

Пример

USE pubs

Рассмотрим темы возможных практических работ по материалам электронного курса лекций-презентаций.

Тема: Проектирование базы данных и приведение к нормальной форме.

Построение проекта базы данных - это один из определяющих моментов всей разработки. Анализ предметной области и задач, составляющих суть деятельности в этой области, позволяет вычленить цели решения задачи, основные объекты рассмотренной задачи, перечень лиц участвующих в решении задачи, задания, связанные с объектами и лицами, связи, возникающие в этих процессах. Построение проекта начинается с приведения всех описанных данных и условий к модели. Модели делятся на иерархические, сетевые, реляционные, смешанные, мультимодельные. Мы рассматриваем варианты реляционной модели. По способу организации базы делятся на локальные (персональные), общие(интегрированные, централизованные), распределенные. Также выделяют три уровня моделей: инфологический (концептуальный, формально описывающий предметную область), даталогический (уточняющий предыдущее проектирование), физический (описывающий устройства и способы хранения данных в среде – схему хранения).

Одним из самых распространенных способов представления структур баз данных является ER-модель (Entity – relationship model) сущность – связь. Диаграмма строится из трех основных типов: множеств сущностей, атрибутов, связей. Сущность в отличие от *объекта* отражает статически данные, поэтому строить предположения о содержании в ней каких-либо *методов,* нет оснований. Множеству сущностей соответствует набор атрибутов, являющихся свойствами сущностей. Атрибуты принято относить к атомарным, либо «struct», как в С, или кортеж с фиксированным числом атомарных компонент, либо множеству значений одного типа: либо

атомарному, либо «struct». Связи – это соединения между двумя или большим числом множеств сущностей. Диаграмма – графическое представление множеств сущностей, их атрибутов, связей. Элементы – вершины графа. Обозначения в диаграмме для элементов:
прямоугольник- сущность, овал- атрибуты, ромб – связи. У множества сущностей могут быть сущности специального свойства отличные от других, для них создаются подклассы, соединения с ними носят характер «is a» - есть и обозначаются треугольником (вершина – база, противоположная сторона - подкласс). Функциональные зависимости (FD) – определение «функции» сопоставляющей набору параметров – значениям атрибутов $A_1A_2…A_n$ строго определенное значение атрибута B, либо функция не возвращающая ничего. Нормальные формы: BCNF (Бойса Кодда нормальная форма)- форма, исключающая аномалии (избыточность, аномалии изменения, аномалии удаления) методом декомпозиции (разбиение на 2 и более отношений - сущностей). 1NF - каждый компонент каждого кортежа – атомарен; 2NF - допускает наличие транзитивных связей FD, запрещающих нетривиальные FD с подмножеством ключа, 3NF – не удовлетворяет BCNF, но декомпозиция не целесообразна.

Рассмотрим построение проекта базы данных на задаче автоматизации деятельности учебного заведения – колледжа, о котором нам будет доступна следующая информация: информация о сотрудниках и их паспортных данных, должностях, контрактах, нагрузке, учащихся и их паспортных данных, № студ. билета, курса, адреса, учебных планах по специальностям, наборам предметов, объёмы, формы занятий, аттестация, курс, семестр, расписание занятий по каким предметам, у кого, когда, где, кто ведет, журналам аттестации студентов, показывающим предмет, преподавателя, вид работы, результат, дату. Кроме этого известно, что преподаватели являются сотрудниками кафедр, а группы формируются из студентов одного курса и специальности.

Представить в виде ER-диаграммы следующие структуры баз данных (см. образцы построения ER- схем лекция №2):

1. База учебного заведения с сотрудниками, учащимися, учебными планами, журналами, мероприятиями.
2. Футбольная база с информацией об игроках, командах, тренерах, болельщиках, матчах, клубах.
3. База фильмов, студий, режиссеров, актеров, продюссеров, студий, проката.
4. База лечебного учреждения со специалистами и сотрудниками, пациентами, характеристиками и видами услуг, платежами.
5. База транспортной компании с видами транспорта, работниками, владельцами, заказчиками, заказами, платежами.
6. База торговой сети с предприятиями, сотрудниками, клиентами, поставщиками, товарами, услугами, платежами.
7. База метеорологических наблюдений со станциями, оборудованием, видами наблюдений, результатами, сотрудниками, журналами.
8. База банковского учреждения с сотрудниками, клиентами, услугами, платежами, валютами.
9. База производственного предприятия с материалами, оборудованием, технологиями, произведенной продукцией, работниками.
10. База природного парка с территориями, флорой, фауной, климатом, сотрудниками, охраняемыми объектами.
11. База археологических данных с местом хранения, названиями экспонатов, экспедиций, мест экспедиций, сотрудников, исторических характеристик.
12. База данных почтовой службы с сотрудниками, видами услуг, адресатами, адресами, стоимостью, контролем.
13. База данных спортивного комплекса с тренерами, посетителями, видами услуг, расписанием, платежами и контролем.

14. База данных ГАИ с сотрудниками, владельцами автотранспорта, автотранспортом, учетом, нарушениями и штрафами, платежами.
15. База данных библиотеки с сотрудниками, читателями, видами услуг, фондами, регистрацией заказов, контролем.
16. База данных адресное бюро с жителями, улицами, домами, регистрацией проживания, сотрудниками, запросами, заказчиками.

Тема: Создание БД и основных объектов БД (таблицы, атрибуты, ограничения, ключи, индексы. См. команды CREATE).

Используя стандартного клиента SMS(Server management studio), создать для задания 1 лабораторная работа №1 базу данных и основные объекты с помощью мастера. Построить диаграмму БД со связями. Построить инструкции T-SQL на создание объектов методом скриптования.

Написать аналогичные инструкции самостоятельно в режиме запросов, с учетом всех ограничений, ключей и связей в одной инструкции CREATE. Построить индексы, повышающие эффективность работы с объектами. Сохранить все инструкции и построить сценарий в одном файле. Проверить его работу, результаты предъявить преподавателю.

Тема: Управление данными (ввод данных, модификация, удаление. Команды INSERT, UPDATE, DELETE)

Используя стандартного клиента SMS(Server management studio), для задания 1 лаб.раб.№1 удалить основные объекты из дерева объектов и саму базу данных. Воспроизвести сценарий на создание из работы №1. Написать инструкции на заполнение всех объектов данными. Добавить в БД информацию о формах аттестации и видах оценок. Обновить соответствующие таблицы. Создать сценарий на удаление, заполнение, модификацию данных в новом файле. Проверить его работу, результаты предъявить.

Тема: Работа с объектами (построение запросов, представлений, временных таблиц. Команды SELECT, CREATE VIEW, CREATE table #Name_table)

Используя стандартного клиента SMS(Server management studio), для задания 1 лаб.раб.№1, написать запросы:

1. Показать список всех учителей с указанием должности и предмета.
2. Показать список всех учеников с указанием класса.
3. Показать списки классных руководителей и классов.
4. Показать список по каждой параллели всех общих дисциплин.
5. Показать журнал заданного класса по предметам.
6. Выдать статистику по параллели и классам о пропусках занятий и средних баллах по двум наиболее общим предметам.
7. Создать временную таблицу «Итоги недели» по классу.
8. Создать представление для просмотра таких итогов по всем классам.
9. Заполнить временную таблицу данными из представления.
10. Показать работу с индексом и без индекса.

Тема: Создание, выполнение, изменение и удаление хранимых процедур и триггеров(CREATE PROCEDURE, DROP , CREATE TRIGGER).

1. Создать процедуру для заполнения временной таблицы из пункта 7. лаб. Раб №4 данными по указанному классу. Просмотреть результат.
2. Создать процедуру показывающую результаты конкретного ученика за неделю.
3. Написать процедуру исправления оценки по заданному ученику и предмету.
4. Создать триггер, контролирующий вид и форму аттестации при занесении оценки в журнал с соответствующими сообщениями.
5. Удалить триггер и процедуры.

Тема: Использование транзакций и блокировок.

Модифицировать сценарии на создание и заполнение данными базы с использованием явной и неявной транзакций. Предусмотреть обработку ошибок. Просмотреть журнал. Предусмотреть блокировки операций с таблицами при их модификации, использовать разные уровни изоляции.

Тема: Индивидуальный проект.

По базам данных из лабораторной работы №1(пункты 2-16) построить сценарий, создающий основные объекты и процедуры их заполнения.

Написать триггеры, контролирующие данные в связующих отношениях.

Построить запросы на объединение, соединение сведений из разных объектов, запросы, выполняющие статистику на группах, на основе представлений по этим запросам создать временные таблицы и заполнить их данными. Написать процедуры, позволяющие по заданным параметрам, выбирать из них информацию.

Рекомендуемая литература (основная):

Диго С.М. Бзы данных.Проектирование и использование. Учебник- М.: Финансы и статистика.2005.

Дейт К.Дж. Введение в системы баз данных.- М: Вильямс, 2005, 8-ое изд.- 1328 с.

Хомоненко А.Д. Базы данных: Учебник для вузов, СПб., Корона принт, 2006.[7]

Советов Б.Я., Цехановский В.В., Чертовский В.Д. Базы данных. Теория и практика. Учебник для вузов. М.: Высшая школа., 2005, 463с.

Гектор Гарсиа-Молина, Джеффри.Д.Ульман, Дженнифер Уидом Системы баз данных. Полный курс. М.: Вильямс., 2003, 1082с.

Глушаков С.В., Ломотько Д.В. Базы данных: Учебный курс. – Харьков: Фолио; М.: ООО «Издательство АСТ», 2000.

Ревунков Г.И., Базы и банки данных знаний. Учебник для вузов, М., Высш. шк., 1992. [6]

Чери С., Готлоб Г, Танка Л. Логическое программирование и базы данных.- М.: Мир, 1992.- 352 с. [1]

Рекомендуемая литература (дополнительная):

Архипенков С. Я. Хранилища данных: От концепции до внедрения / С. Я. Архипенков, Д. В. Голубев, О. Б. Максименко ; Под ред. С. Архипенкова. - М. : Диалог-МИФИ, 2002. - 528 с. : ил. - Библиогр.: с. 525 (19 назв.). - ISBN 5-86404-167-X . [1]

Михеев Р.Н. MS SQL Server 2005 для администраторов. СПб.: БВХ-Петербург, 2006.- 544с.

Мамаев Е. Microsoft SQL Server 2000. Спб. 2003.[1]

Ребекка М. Риордан «Программирование в SQL Server 2000».[4]

Малкольм Г. Программирование для Microsoft SQL Server 2000 с использованием XML./Пер. с англ.- М.: Издательско-торговый дом «Русская редакция», 2002, 320 с.:илл.[1]

Мак-Маус Д.П., Гольдштейн Д., Прайс К.Т. Обработка баз данных на Visual Basic.Net, М., 2003[3]

Гофман В.Э., Хомоненко А.Д. Работа с базами данных в DELPHI – 2-е изд. –СПб.: БХВ-Петербург, 2003.- 624 с.: илл..

Архипенков С.Я. Аналитические системы на базе Oracle Express Olap. М., Диалог-Мифи, 2000.

Лугачев М.И. и др. Экономическая информатика: Введение в экономический анализ. Учебник. –М.: ИНФРА-М, 2005.- 958с.

СУБД ORACLE и ее сетевые приложения. М., Россия, 1993.

Архипенков С. Я. От переработки данных к анализу//Банковские технологии» 1998.

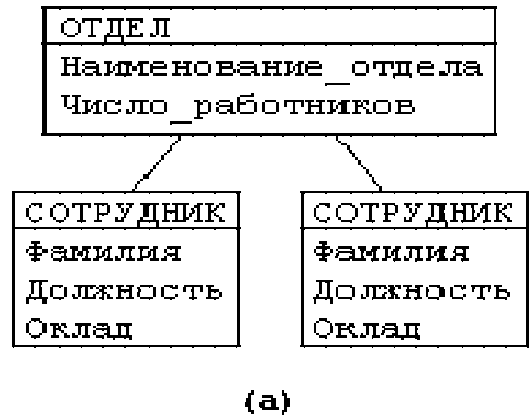
Марков А.С. Информационная технология по стандартам ИСО.- М.: Финансы и статистика, 2000-382 с.

Заморин А.П., Марков А.С. Толковый словарь по вычислительной технике и программированию (Ред. Шура-Бура М.Р.). -М.: Русский язык, 1988.- 221 с.

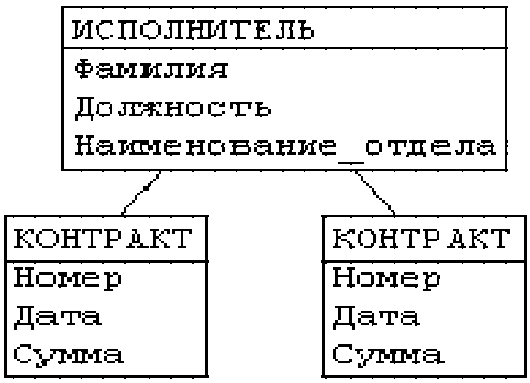
Першиков В.И., Марков А.С., Савинков В.М. Русско-английский толковый словарь по информатике (3-е издание).- М.: Финансы и статистика, 1999.363 с.

Клайн К. SQL . Справочник 2-ое изд.

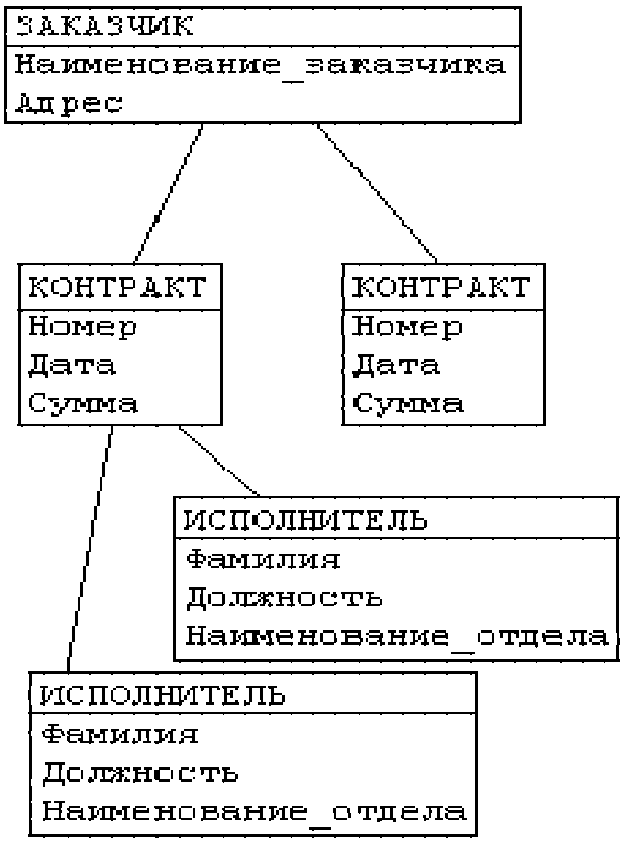
Приложение 1. Иерархическая структура.



(a)



(c)



(b)

Приложение 2. Реляционная структура.