SQL (Structured Query Language).

SQL относится к классу непроцедурных языков программирования, SQL ориентирован на множества, так как в качестве исходной информации используется множество кортежей записей одной или нескольких таблиц-отношений. Результатом любой операции SQL также является таблица –отношение.

Существуют и используются две формы языка SQL: интерактивный SQL и встроенный SQL.

Интерактивный SQL используется для задания SQL – запросов пользователем и получения результата в интерактивном режиме.

Встроенный SQL состоит из команд, которые встраиваются в программы на других языках (Паскаль, C, C++, JAVA и др.). Это делает программы более мощными и эффективными. При этом используются дополнительные средства интерфейса SQL с выбранным языком программирования.

SQL подразделяется на DDL (язык определения данных) и DML (язык обработки данных).

В языке SQL имеются средства для указания типов данных , которым соответствуют отдельные атрибуты.

Определение типов данных является той частью, в которой коммерческие реализации языка не согласуются с требованиями официального стандарта.

numeric

exact numeric

integer

bigint Целые от –2^63 до 2^63-1

int Целые от –2^31 до 2^31-1

smallint Целые от –2^15 до 2^15-1

tinyint Целые от 0 до 255

bit Целые 0 или 1

decimal and numeric

decimal Десятичные числа с фиксированным numeric количеством знаков до и после запятой от –10^38+1 до 10^38-1

money and smallmoney

money Числа в денежном формате от –2^63 до 2^63-1, точность 0,0001 от денежной единицы smallmoney Числа в денежном формате от –214748,3648 до 214748,3647, точность 0,0001 от денежной единицы

approximate numeric

float Числа с плавающей точкой от –1.79E+308 до 1.79E+308

real Числа с плавающей точкой от –3.40E+38 до 3.40E+38

datetime and smalldatetime

datetime Значения даты и времени начиная с 1.01.1753 до31.12.9999

smalldatetime Значения даты и времени начиная с 1.01.1900 до 6.06.2079

character string

char Символьные значения(не Unicode) фиксированной длины максимум 8000 символов

varchar Символьные значения(не Unicode) переменной длины максимум 8000 символов

text Данные (не Unicode) переменной длины максимум до 2147483647 или 2^31 символов

Неопределенные или пропущенные данные (NULL).

Для обозначения неопределенных, пропущенных, или неизвестных значений SQL использует слово NULL. Строго говоря NULL не является значением в обычном понимании. Поэтому необходимо учитывать эту особенность при использовании значений атрибутов, которые могут находиться в состоянии NULL:

- **В агрегирующих функциях, позволяющих получать сводную информацию по множеству значений атрибута(суммарное или среднее).**
- **Условные операторы от булевой двузначной логики TRUE/FALSE расширяются до трехзначной**

- **Все операторы возвращают состояние NULL, если один из операндов NULL.**
- **Для проверки на наличие NULL используются специальные операторы IS NULL (IS NOT NULL). Знак = использовать нельзя!**
- **Функции преобразования типов при аргументе NULL возвращают NULL.**

Используемые термины и обозначения.

Ключевые слова **– зарезервированные в SQL слова.**

Команды или предложения- **это инструкции, с помощью которых SQL обращается к БД.**

Объекты- **имена таблиц, атрибутов, индексов и т.п.**

В описании синтаксиса [ ] указывают на не обязательный параметр, знак … означает повторение ранее указанного выражения, {} объединяют последовательность элементов в логическую группу, один из элементов которой должен быть использован. Вертикальна черта | указывает, что часть, следующая за этим символом, является одним из возможных вариантов. Угловые скобки < > , заключают элементы объясняемые по ходу.

База данных «Колледж»

STUDENTS				
SNUM	SFAM	SNAME	SFATH	STIP
3412	Поляков	Анатолий	Алексеевич	25.50
3413	Старова	Любовь	Михайловна	17.00
3414	Гриценко	Владимир	Николаевич	0.00
3415	Котенко	Анатолий	Николаевич	0.00
3416	Нагорный	Евгений	Васильевич	25.50

PREDMET				
PNUM	PNAME	TNUM	HOURS	COURS
2001	Физика	4001	34	1
2002	Химия	4002	68	1
2003	Математика	4003	65	1
2004	Философия	4005	17	2
2005	Экономика	4004	17	2

USP				
UNUM	SNUM	PNUM	UDATE	MARK
1001	3412	2001	10.06.02	5
1002	3413	2003	10.06.02	4
1003	3414	2005	11.06.02	3
1004	3412	2003	12.06.02	4
1005	3416	2004	12.06.02	5

TEACHERS				
TNUM	TFAM	TNAME	TFATH	TDATE
4001	Викулова	Валентина	Ивановна	01.04.94
4002	Костенко	Олег	Владимирович	01.09.97
4003	Казанцев	Виталий	Владимирович	01.09.88
4004	Поздняк	Любовь	Алексеевна	01.09.88
4005	Загоруйко	Илья	Дмитриевич	01.09.99

Простейшие SELECT- запросы.

Оператор SELECT (выбрать) языка SQL является самым важным и самым часто используемым оператором. В обобщенной форме его синтаксис :

SELECT [DISTINCT] <список атрибутов>

FROM <список таблиц>

[WHERE <условие выборки>]

[ORDER BY <список атрибутов>]

[GROUP BY <список атрибутов>]

[HAVING <условие>]

[UNION <выражение с оператором SELECT>];

ПРИМЕРЫ:

1) SELECT * from USP

WHERE SNUM = 3412;

unum	snum	pnum	udate	mark
1001	3412	2001	10.06.200	5
1004	3412	2003	12.06.200	4

2)SELECT SNUM,SFAM,SNAME

FROM STUDENTS

WHERE STIP>0;

SNUM	SFAM	SNAME
3412	Поляков	Анатолий
3413	Старова	Любовь
3416	Нагорный	Евгений

Построение запросов с условием отбора.

Наибольший интерес представляют такие запросы, в которых выполняется выборка данных в соответствии с поставленными условиями. В записи условия отбора используются логические выражения. Порядок действий регламентируется скобками, логическими операциями и связками.

Пример 1

Показать номера студенческих билетов, фамилии и имена тех лиц, чьи имена начинаются с буквы «А».

```
SELECT SNUM, SFAM,SNAME
FROM STUDENTS
WHERE SNAME)<"Б";
```

SNUM	SFAM	SNAME
3412	Поляков	Анатолий
3415	Котенко	Анатолий

Пример 2

Показать предметы, которые изучаются на 1 курсе и на них отводится более 30 часов.

SELECT PREDMET.PNUM, PREDMET.PNAME, PREDMET.HOURS, PREDMET.COURS

FROM PREDMET

WHERE (((PREDMET.HOURS)>30) AND ((PREDMET.COURS)=1));

PNUM	PNAME	HOURS	COURS
2001	Физика	34	1
2002	Химия	68	1
2003	Математика	65	1

В записи логических условий могут быть использованы операторы IN, BETWEEN, LIKE, IS NULL.

Операторы IN (равен любому из списка) и NOT IN (не равен любому из списка) используются для сравнения проверяемого значения поля с заданным списком. Список значений указывается справа от оператора и заключается в скобки.

IN (3412; 3413; 3414; 3415; 3416)

Пример 3

Получить сведения о студентах, получивших оценки только 4 и 5.

```
SELECT USP.SNUM, USP.UDATE, USP.MARK, STUDENTS.SFAM
FROM USP, STUDENTS
WHERE (((USP.SNUM)=[STUDENTS].[SNUM]) AND ((USP.MARK) IN (4,5)));
```

SNUM	UDATE	MARK	SFAM
3412	10.06.2002	5	Поляков
3413	10.06.2002	4	Старова
3412	12.06.2002	4	Поляков
3416	12.06.2002	5	Нагорный
3412	13.06.2002	4	Поляков

Пример 4

Получить сведения о студентах, не получивших оценок 4 и 5.

```
SELECT USP.SNUM, STUDENTS.SFAM, USP.UDATE, USP.MARK

FROM USP, STUDENTS

WHERE (((USP.SNUM)=[STUDENTS].[SNUM]) AND ((USP.MARK) NOT IN (4,5)));
```

SNUM	SFAM	UDATE	MARK
3414	Гриценко	11.06.2002	3
3414	Гриценко	12.06.2002	2

Оператор BETWEEN используется для проверки условия вхождения значения поля в заданный интервал, т.е. задаются вместо списка границы. BETWEEN 20 AND 30 . Типы полей как числовые, так и символьные.

Пример 5 Показать список тех, кто получает стипендию в указанном диапазоне.

```
SELECT STUDENTS.SNUM, STUDENTS.SFAM, STUDENTS.STIP
FROM STUDENTS
WHERE (((STUDENTS.STIP) BETWEEN 20 AND 30));
```

SNUM	SFAM	STIP
3412	Поляков	25,50
3416	Нагорный	25,50

Оператор LIKE применим только символьным полям типа CHAR или VARCHAR. Этот оператор просматривает строковые значения полей и определяет входит ли образец поиска в символьную строку-значение поля. В образце может использоваться шаблон:

- **Символ подчеркивания «_» определяет наличие 1 любого символа.**
- **Символ % или * допускает наличие любых символов произвольной длины.**

Пример 7

Показать списки студентов с отчеством на «Ни*».

```
SELECT STUDENTS.SNUM, STUDENTS.SFAM, STUDENTS.SNAME, STUDENTS.SFATH
FROM STUDENTS
WHERE (((STUDENTS.SFATH) LIKE "Ни*"));
```

SNUM	SFAM	SNAME	SFATH
3414	Гриценко	Владимир	Николаевич
3415	Котенко	Анатолий	Николаевич

Если внутри образца содержатся знаки _ | % | *|, то применяют escape – символы.

Например, в выражении LIKE “_%_” ESCAPE “%” знак % будет восприниматься не как управляющий символ, а как процент.

Все рассмотренные ранее операторы нельзя рассматривать для работы с NULL .

Для этого используют IS NULL (является пустым) или IS NOT NULL (не является пустым).

Пример 8

Составить список изучаемых предметов.

SELECT PREDMET.PNUM as код, PREDMET.PNAME as название, PREDMET.HOURS as количество_часов

FROM PREDMET;

код	название	количество_часов
2001	Физика	34
2002	Химия	68
2003	Математика	65
2004	Философия	17
2005	Экономика	17

Пример. Создать таблицу STUDENTS.

- CREATE TABLE STUDENTS
  (SNUM INTEGER, SFAM CHAR (20),
  SNAME CHAR (15), SFATH CHAR (15),
  STIP DECIMAL)

1. **В этой команде порядок полей определяется их местом в списке.**
2. **После того, как таблица создана, её можно изменить.**

Для удаления таблицы необходимо:

1) быть ее создателем или иметь на это право;

2) перед удалением необходимо ее очистить от данных, это позволяет избежать случайной потери информации.

- DROP TABLE <name of table>;

Пример. Удалить все сведения и таблицу PREDMET.

1. DELETE FROM PREDMET;
2. DROP TABLE PREDMET;

Использование выражений :

- унарный оператор « - » (знак минус) меняет знак выражения на противоположный;
- бинарные операторы « + », « - », « * », « / » предоставляют возможность выполнения арифметических действий;
- операция конкатенации строк + (||) позволяет «склеивать» значения двух и более строк.

Пример 9

1) **Увеличить размер стипендии «учащимся без троек»(оператор *)**

```
SELECT DISTINCT STUDENTS.SNUM, STUDENTS.SFAM,
    STUDENTS.STIP*1.25 AS STIP

FROM STUDENTS, USP

WHERE (((STUDENTS.SNUM)=[USP].[SNUM]) AND
    ((USP.MARK)>3));
```

SNUM	SFAM	STIP
3412	Поляков	31,875
3413	Старова	21,25
3416	Нагорный	31,875

Функции преобразования символов в строке:

- LOWER <строка> – перевод в строчные символы(нижний регистр)
- UPPER <строка> – перевод в прописные символы(верхний регистр)
- INITCAP <строка> – перевод первой буквы каждого слова в прописную(верхний регистр)
- LPAD(<строка>,<длина>[,<подстрока>]) – дополнение строки слева элементами подстроки, по умолчанию пробелами; если <длина> меньше длины <строки>, то исходная строка усекается слева до заданной длины.
- RPAD (<строка>, <длина> [,<подстрока>]) – дополнение строки справа элементами подстроки, по умолчанию пробелами; если <длина> меньше длины <строки>, то исходная строка усекается справа до заданной длины.

- **LTRIM (<строка [,<подстрока>]) удаление левых граничных символов**
- **RTRIM (<строка> [,<подстрока>]) удаление правых граничных символов**
- **SUBSTR (<строка>, <начало> [,<количество>]) выделение подстроки**
- **INSTR (<строка>, < подстрока > [,<начало поиска>]) поиск подстроки**
- **LENGH (<строка>) длина строки**

ЧИСЛОВЫЕ ФУНКЦИИ:

- ABS – абсолютное значение
- FLOOR –урезанное целое
- CELL-самое малое целое >=заданного
- ROUND - округленное
- TRUNC - усеченное с точностью
- COS, SIN, TAN - тригонометрические
- COSH, SINH, TANH -гиперболические
- EXP, LOG – экспонента, логарифм
- POWER, SQRT – степень, корень
- SIGN- знак.

Агрегирование и групповые функции

Агрегирующие функции позволяют получать из таблицы сводную (агрегированную) информацию, выполняя операции над группой строк таблицы.

Для задания в SELECT-запросе агрегирующих операций используются следующие ключевые слова:

- COUNT определяет количество строк или значений поля, выбранных посредством запроса включая NULL-значения;

Для подсчета общего количества строк в таблице следует использовать функцию COUNT .

COUNT **(** { [ALL | DISTINCT] *expression*] | ***** } **)**

SELECT COUNT(*) FROM USP;

Аргументы DISTINCT и ALL позволяют, соответственно, исключать и включать дубликаты обрабатываемых функцией COUNT значений, ALL работает по умолчанию .

```
SELECT COUNT(DISTINCT SNUM)
FROM USP;
```

DDL – язык определения данных.

- В SQL существует ряд операторов, позволяющих изменять структуру данных .
- Операторы DDL позволяют не вникать в детали хранения информации в БД на физическом уровне, используя стандартные понятия поля и таблицы.

Это операции:

1) создание новой БД;
2) определение новой структуры и создание таблицы;
3) удаление таблицы;
4) изменение структуры существующей таблицы;
5) обеспечение условий безопасности;
6) создание индексов для доступа к таблице;
7) управление размещением данных на устройствах.

Основу DDL составляют три команды:

1) CREATE - создать;
2) DROP – удалить;
3) ALTER – изменить.

Создание базы данных.

В системе MS SQL эти действия выполняются оператором:

CREATE DATABASE

<name of database> ON < name 1>, <…>, < name n>;

```
CREATE DATABASE database_name
[ ON
   [ < filespec > [ ,...n ] ]
   [ , < filegroup > [ ,...n ] ]
]
[ LOG ON { < filespec > [ ,...n ] } ]
[ COLLATE collation_name ]
[ FOR LOAD | FOR ATTACH ]
< filespec > ::=
[ PRIMARY ]
( [ NAME = logical_file_name , ]
   FILENAME = 'os_file_name'
   [ , SIZE = size ]
   [ , MAXSIZE = { max_size | UNLIMITED } ]
   [ , FILEGROWTH = growth_increment ] ) [ ,...n ]
< filegroup > ::=
FILEGROUP filegroup_name < filespec > [ ,...n ]
```

Создание БД со спецификациями данных и журнала.

```
USE master
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
   FILENAME = 'c:\program files\microsoft sql server\mssql\data\saledat.mdf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'c:\program files\microsoft sql server\mssql\data\salelog.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

```
USE master
GO
CREATE DATABASE Archive
ON
PRIMARY ( NAME = Arch1,
   FILENAME = 'c:\program files\microsoft sql
     SIZE = 100MB,
server\mssql\data\archdat1.mdf',
     MAXSIZE = 200,
     FILEGROWTH = 20),
( NAME = Arch2,
  FILENAME = 'c:\program files\microsoft sql
  SIZE = 100MB,
server\mssql\data\archdat2.ndf',
  MAXSIZE = 200,
  FILEGROWTH = 20),
( NAME = Arch3,
  FILENAME = 'c:\program files\microsoft sql
  SIZE = 100MB,
server\mssql\data\archdat3.ndf',
  MAXSIZE = 200,
  FILEGROWTH = 20)
LOG ON
( NAME = Archlog1,
  FILENAME = 'c:\program files\microsoft sql
  SIZE = 100MB,
server\mssql\data\archlog1.ldf',
  MAXSIZE = 200,
  FILEGROWTH = 20),
( NAME = Archlog2,
  FILENAME = 'c:\program files\microsoft sql
  SIZE = 100MB,
server\mssql\data\archlog2.ldf',
  MAXSIZE = 200,
  FILEGROWTH = 20)
GO
```

```
USE master
GO
CREATE DATABASE Employees
ON
( NAME = Empl_dat,
   FILENAME = 'f:',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'g:',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

После создания пустой базы можно создавать таблицы. Эти действия относятся к структуре, а не к данным.

```
CREATE TABLE
   [ database_name.[ owner ] . | owner. ] table_name
   ( { < column_definition >
      | column_name AS computed_column_expression
      | < table_constraint > ::= [ CONSTRAINT
constraint_name ] }
         | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
   )

[ ON { filegroup | DEFAULT } ]
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]

< column_definition > ::= { column_name data_type }
   [ COLLATE < collation_name > ]
   [ [ DEFAULT constant_expression ]
      | [ IDENTITY [ ( seed , increment ) [ NOT FOR
REPLICATION ] ] ]
   ]
   [ ROWGUIDCOL]
   [ < column_constraint > ] [ ...n ]

< column_constraint > ::= [ CONSTRAINT
constraint_name ]
   { [ NULL | NOT NULL ]
      | [ { PRIMARY KEY | UNIQUE }
         [ CLUSTERED | NONCLUSTERED ]
         [ WITH FILLFACTOR = fillfactor ]
         [ON {filegroup | DEFAULT} ] ]
      ]
      | [ [ FOREIGN KEY ]
         REFERENCES ref_table [ ( ref_column ) ]
         [ ON DELETE { CASCADE | NO ACTION } ]
         [ ON UPDATE { CASCADE | NO ACTION }
]
         [ NOT FOR REPLICATION ]
      ]
      | CHECK [ NOT FOR REPLICATION ]
      ( logical_expression )
   }
< table_constraint > ::= [ CONSTRAINT
constraint_name ]
   { [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      { ( column [ ASC | DESC ] [ ,...n ] ) }
      [ WITH FILLFACTOR = fillfactor ]
      [ ON { filegroup | DEFAULT } ]
   ]
   | FOREIGN KEY
      [ ( column [ ,...n ] ) ]
      REFERENCES ref_table [ ( ref_column [ ,...n
] ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ]
      ( search_conditions )
   }
```

1) Для разделения элементов команды используются пробелы, пробел не может быть частью имени (*MY*_Table).
2) Значение аргумента размерность [<size of attribution>] – зависит от типа данных и может заноситься по умолчанию самой СУБД. Это удобно для числовых полей. Тип CHAR требует обязательного указания размера – количества символов в поле. По умолчанию размер равен одному **символу.**
3) Имена таблиц должны отличаться, но могут использоваться одинаковые имена полей (STUDENTS.SNUM USP.SNUM).
4) Пользователи не владельцы таблиц должны к ним обращаться по имени владельца. (SA.STUDENTS).

Пример. Создать таблицу STUDENTS.

- CREATE TABLE STUDENTS (SNUM INTEGER, SFAM CHAR (20), SNAME CHAR (15), SFATH CHAR (15), STIP DECIMAL)

1. **В этой команде порядок полей определяется их местом в списке.**
2. **После того, как таблица создана, её можно изменить.**

Добавление новых полей выполняется командой:

ALTER TABLE <name of table> ADD <Name of attribution1> <type of attribution1> [(<size of attribution1>)], …

<Name of attribution n> <type of attribution n> [(<size of attribution n>)]);

Добавляемые поля автоматически получают значения NULL.

Пример. Предположим мы решили добавить номер курса и специальность.

```
ALTER TABLE STUDENTS
ADD
COURS INTEGER,
SPEC CHAR (20);
```

Для удаления таблицы необходимо:
1) быть ее создателем или иметь на это право;
2) перед удалением необходимо ее очистить от данных, это позволяет избежать случайной потери информации.

- DROP TABLE <name of table>;

Пример. Удалить все сведения и таблицу PREDMET.

1. DELETE FROM PREDMET;
2. DROP TABLE PREDMET;

```
INSERT [ INTO]
   { table_name WITH ( < table_hint_limited > [ ...n ] )
      | view_name
      | rowset_function_limited
   }

   {    [ ( column_list ) ]
      { VALUES
         ( { DEFAULT | NULL | expression } [ ,...n] )
         | derived_table
         | execute_statement
      }
   }
```

Пример

Создать для пользователя копию таблицы PREDMET,
добавить в нее поля: лабораторные работы, их количество.

- CREATE TABLE PREDMET_NEW (

  PNUM INTEGER,

  PNAME CHAR (30),

  COURS INTEGER,

  HOURS INTEGER,

  LAB CHAR (30),

  NUM INTEGER);

- INSERT INTO PREDMET_NEW SELECT * FROM PREDMET;
- Новые поля заполнятся значениями по умолчанию или значениями NULL.

PNUM	PNAME	TNUM	COURS	HOURS	LAB	NUM
2001	Физика	4001	1	34		0
2002	Химия	4002	1	68		0
2003	Математика	4003	1	65		0
2004	Философия	4005	2	17		0
2005	Экономика	4004	3	17		0

Индексы, ограничения, синонимы.

- **Индексом принято называть упорядоченный список полей таблицы или групп полей в таблице. В таблице с большим количеством полей при отсутствии упорядоченности поиск может занимать длительное время.**
- **Индексный адрес – это специальный метод объединения всех значений в группы(из 1 или более записей), которые отличаются друг от друга, т.к. уникальность записей часто необходима.**

- Когда создаётся индекс, в поле БД запоминается порядок всех значений этого поля в области памяти.
- При наличии индекса система могла бы найти искомый номер прямо в этом упорядоченном массиве и указать, какую искомую строку следует найти. У индексов есть и недостатки :

1) наличие индексов замедляет операции модификации INSERT, DELETE;
2) сам индекс занимает тоже место в памяти.

- Индексы могут состоять из нескольких полей, при этом первое поле считается главным, второе поле упорядоченным внутри первого и т. д.
- Создаются индексы командой:
- CREATE INDEX <Name of index> on <Name of table>(< Name of attribution1 >,[< Name of attribution2 >,…]);
- Разумеется, таблица должна быть создана ранее, и иметь имена полей указанных в команде. Имя индекса является уникальным и не может быть использовано в других целях. SQL сам определяет, когда индекс необходим и использует его автоматически.

- Пример В таблице STUDENT наиболее часто употребимо поле SFAM, создать индекс по этому полю.
- **CREATE INDEX SFAMIDX on STUDENTS(SFAM);**
- При создании индекса ему не приписана уникальность. Это делается с помощью специального ключевого слова UNIQUE.
- **CREATE UNIQUE INDEX SNUMIDX ON STUDENTS(SNUM);**
- Однако эта команда не будет выполнена, если среди значений этого поля есть не уникальные значения. Поэтому рекомендуем создавать индексы сразу после создания её структуры, до ввода в неё значений.

- Для удаления используется команда:

DROP INDEX <Name of index>;

- Например:

DROP INDEX SFAMIDX;

- Удаление индексов не влияет на данные.

Ограничения данных.

- Ограничения данных – это часть определений таблицы, описывающих условия ввода данных. В качестве ограничений мы рассмотрим тип, размер вводимых данных, т.е. их совместимость с полями, в которые вводятся данные. Ограничения дают возможность оговорить их значения по умолчанию.

Существуют ограничения двух типов:

1) **ограничения поля – применимые только к указанному полю;**
2) **ограничения таблицы – применимые к указанным группам полей.**

Ограничения поля (атрибута) – помещается в конец фрагмента команды, объявляющего его имя после типа данных.

Ограничения таблицы (отношения) – помещаются в конец объявления имени таблицы после последнего имени поля.

- CREATE TABLE <name of table>

  (<Name of attribution1> <type of attribution1> [(<size of attribution1>)] <limit1>,

  <Name of attribution2> <type of attribution2> [(<size of attribution2>] <limit2>, … ,

<Name of attribution n> <type of attribution n> [(<size of attribution n>)], <limit n>,

<limit of table>);

- Часто описание ограничений используют для ограждения от так называемых NULL значений, для этих целей используют предложения NOT NULL, которое может быть указано как ограничение поля.

Ограничения по уникальности.

- **Уникальные индексы** – один из самых простых и наиболее эффективных методов. Однако имеется возможность установить уникальность для отдельных столбцов (полей) таблицы, если существует уверенность, что все значения должны отличаться. **При создании таблицы в конкретном поле указывается слово UNIQUE, при этом СУБД будет контролировать процесс ввода и отклонит попытку ввести имевшееся ранее значение.** Это ограничение может применяться **к полям с ограничением NOT NULL.**

Пример 11.3
Устраним повторяющиеся значения в поле SNUM.

- **CREATE TABLE STUDENTS**

(SNUM INTEGER NOT NULL UNIQUE,

SFAM CHAR (20) NOT NULL,

SNAME CHAR (15),

SFATH CHAR (15),

STIP DECIMAL);

- Напоминаем, что поля являющиеся уникальными являются кандидатами-ключами, или уникальными ключами.

- **Подобное ограничение в поле SFAM запретило бы иметь однофамильцев в таблице STUDENTS!**
- **Объявление уникальности возможно и для группы полей, с помощью ограничения к таблице.**
- **Между уникальностью поля и таблицы существуют различия:**

1) **уникальные поля - дают единственную запись-строку;**
2) **уникальные группы – уникальная комбинация значений полей из этой группы, при этом не требуется уникальность каждого отдельного поля.**

- **С другой стороны, если хотя бы одно поле в группе уникальное, то и значение всей группы уникальное.**

Транзакции

- Транзакция это последовательность операций, объединенных в единый логический рабочий модуль.
- Механизм транзакций позволяет контролировать выполнение операций в этом логическом модуле и производить откаты (отмену уже сделанной операции), если этого требует логика приложения.
- Рабочий модуль должен соответствовать основным требованиям к транзакциям, сокращенно называемые ACID (Atomicity, Consistency, Isolation, Durability)

- Atomicity (атомарность)Логика приложения должна предполагать, что должны быть проделаны либо все изменения данных, входящие в транзакцию либо ни одного;
- Consistensy(постоянство) После завершения транзакции не должна быть нарушена целостность данных,система не может оказаться в неком промежуточном состоянии;
- Isolation (изолированность)Изменения, производимые в рамках одной транзакции, изолируются от других (конкурирующих) транзакций;(4-уровня изоляции:0-двум процессам запрещается изменять одни и те же данные; 1-запрещено считывание пока идут изменения; 2-в промежутках чтения в одной TRAN не допускаются изменения в другой; 3-запрещаются в это время вставки и удаления)
- Durability (устойчивость)После завершения транзакции все сделанные изменения будут сделаны в любом случае, даже если во время этого процесса произошел сбой системы или потеря связи – после восстановления работоспособности SQL сервер обращается к журналу транзакций и производит изменения.

О соответствии транзакции ACID заботится разработчик.

Запуск транзакции

SQL сервер позволяет запустить явную, автоматически совершаемую или неявную транзакцию

- Explicit (явная) транзакция предваряется выражением BEGIN TRANSACTION
- Autocommit (автоматически совершаемая) транзакция – режим, в котором работает SQL сервер по умолчанию, каждая отдельная инструкция T-SQL совершается (изменения в данные вносятся физически) после отрабатывания инструкции. Не нужно указывать никаких ключевых слов, чтобы начать такую транзакцию
- Implicit (неявная) транзакция. Такой режим транзакции устанавливается инструкцией SET IMPLICIT_TRANSACTIONS ON, следующая за этой инструкцией конструкция T-SQL автоматически начинает новую транзакцию. Когда эта транзакция завершается, следующее выражение начинает новую транзакцию.

Завершение транзакции.

- Для завершения транзакции используется конструкция COMMIT
- Если все прошло успешно, конструкция COMMIT гарантирует, что все изменения будут сделаны на физическом уровне.
- Если же во время выполнения транзакции произошла ошибка, используется конструкция ROLLBACK – данные возвращаются к первоначальному состоянию, или к некоторой точке сохранения, системные ресурсы освобождаются.

Синтаксис

- SAVE TRAN [SACTION] { *savepoint_name* | *@savepoint_variable* } – объявить savepoint
- BEGIN TRAN [SACTION] [*transaction_name* | *@tran_name_variable*
  [WITH MARK ['*description*']]]
- ROLLBACK [TRAN [SACTION]
  [*transaction_name* | *@tran_name_variable*
  | *savepoint_name* | *@savepoint_variable*]]
- COMMIT [TRAN [SACTION] [*transaction_name* | **@***tran_name_variable*]]

Триггеры

Триггер - особая 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. В триггере определяется тип запроса (INSERT, DELETE или UPDATE) и таблица, с которыми он связан.

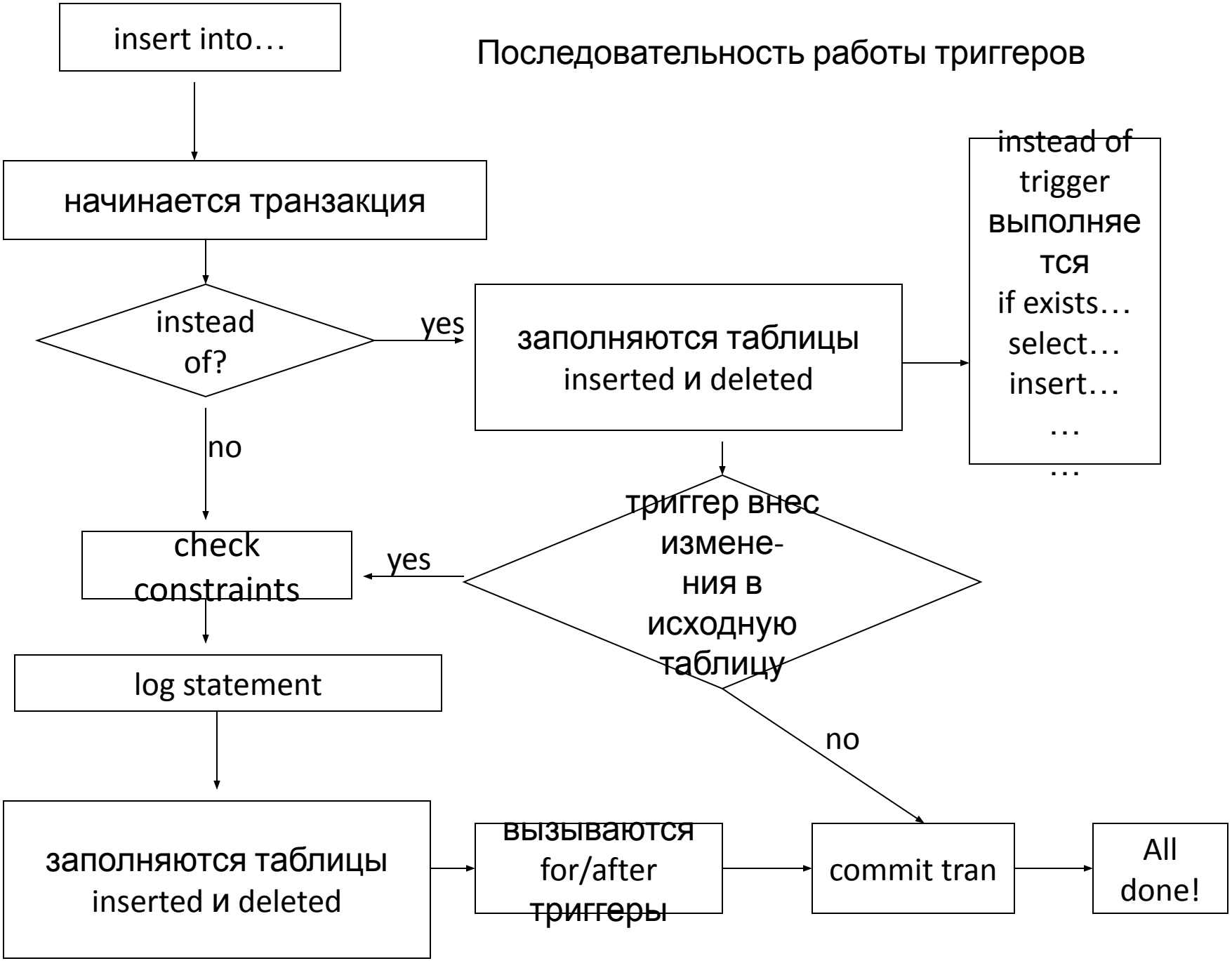
Во время выполнения триггера создаются две специальные таблицы - INSERTED и DELETED. В них находятся записи, соответственно добавляемые или удаляемые запросами в таблице, для которой создан триггер.

Синтаксис

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
   { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
      [ WITH APPEND ]
      [ NOT FOR REPLICATION ]
      AS
      [ { IF UPDATE ( column )
         [ { AND | OR } UPDATE ( column ) ]
            [ ...n ]
      | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
            { comparison_operator } column_bitmask [ ...n ]
      } ]
      sql_statement [ ...n ]
   }
}
```

- FOR (или AFTER) и INSTEAD OF устанавливают тип триггера. FOR(AFTER) – все операции в триггере выполняются **после** того, как отработал запрос, на который наложен триггер. INSTEAD OF – триггер выполняется **вместо** запроса (таблицы deleted и inserted создаются и заполняются, однако модификация данных в первичной таблице не производится).
- WITH APPEND – для совместимости с предыдущими версиями, доступна только если MS SQL сервер работает в режиме совместимости с предыдущими версиями (в предыдущих версиях нельзя было в явном виде создать несколько однотипных триггеров на одной и той же таблице)
- IF UPDATE(COLUMN NAME) – true, если колонка была

Последовательность работы триггеров



Пример 1

```
CREATE TABLE my_table (a int NULL, b int NULL)
GO
ALTER TRIGGER my_trig ON my_table
FOR INSERT
AS
PRINT '1'
GO
ALTER TRIGGER my_trig1 ON my_table
FOR INSERT
AS
PRINT '2'
--ROLLBACK TRAN
GO
insert into my_table(a) values(1)
insert into my_table(a,b) values(1,2)
```

Хранимые процедуры

- *"трехзвенная архитектура"* - имеется хранилище данных (1-е звено), имеется сервер приложений (2-е звено), который выбирает из этого хранилища данные и определенным образом эти данные обрабатывает и после обработки конечный результат уже посылает на терминал клиента (3-е звено).

- *"клиент-сервер"* - имеется хранилище данных (сервер) и клиент, который с этого сервера выбирает данные с помощью определенного языка запросов (SQL) (Устаревший взгляд, возвращаюший нас во времена СУБД типа FoxPRO со встроенной поддержкой sql-запросов).

Репликации , дублирование, восстановление

Репликация, дублирование и восстановление.

- **Репликация** - это процесс, посредством которого данные копируются между базами данных, находящимися на том же самом сервере или на других серверах, связанных через LAN, WAN или Internet
- **Репликация** Microsoft SQL Server использует метафоры (способы передачи данных между БД по сети):

  publisher

  distributor

  subscriber.

- Publisher - сервер или база данных, которая посылает данные на другой сервер или в другую базу данных.

- Subscriber - сервер или база данных, которая получает данные от другого сервера или другой базы данных.

- Distributor - сервер, который управляет потоком данных через систему **репликации**. Этот сервер содержит специализированную базу данных: Distribution database.

- Publisher содержит публикацию/публикации. Публикация - это совокупность одной или более статей, которые посылаются серверу подписчику (subscriber) или базе данных.

- Статья (Article) - основной модуль **репликации** и это может быть таблица или подмножество таблицы.

- Подписка (subscriptions) - это группа данных, которые сервер или база данных получает.

- Существуют виды подписки:

push и pull subscriptions

- Push subscription - это подписка, при которой сервер издатель периодически помещает транзакции на подписавшиеся сервера или базы данных.
- Pull subscription - это подписка, при которой подписавшийся сервер будет периодически соединяться с тиражируемой информацией и перемещать её из Distribution database.

- Distribution database - это системная база данных, которая хранится на дистрибуторе (distributor) и не содержит никаких пользовательских таблиц. Эта база данных используется для хранения снимков заданий и всех транзакций, ожидающих распределения подписчикам.

Топология **репликации**

Microsoft SQL Server поддерживает следующие топологии **репликации**

- Центральный publisher

- Центральный subscriber

- Центральный publisher с отдаленным distributor

- Центральный distributor

- Издающий subscriber

Центральный publisher

Это одна из наиболее используемых топологий **репликации**. В этом сценарии, один сервер исполняет роли publisher и distributor, а другой сервер/серверы определяется, как подписчик/подписчики.

Центральный subscriber

Это обычная топология складирования данных. Несколько серверов или баз данных копируют свои данные на центральный сервер в одну или более базы данных

Центральный publisher с отдаленным distributor

В этой топологии база Distribution постоянно находится на сервере, отличном от сервера, где располагается publisher. Эта топология используется для повышения эффективности, когда объём **репликации** увеличивается, а также, если сервер или сетевые ресурсы ограничены. Это уменьшает загрузку publisher, но увеличивает сетевой трафик. Эта топология требует отдельных инсталляций Microsoft SQL Server для publisher и для distributor.

Центральный distributor

- В этой топологии, несколько издателей используют только один distributor, который постоянно находится на отличном от издателей сервере. Это одна из наиболее редко используемой топологии **репликации**, потому что имеет уязвимую точку (на сервере с центральным distributor), и если сервер distributor потерпит неудачу, сценарий **репликации** будет разрушен полностью.

Издающий subscriber

Это топология двойственной роли. В ней, два сервера издают те же самые данные. Сервер издатель посылает данные на subscriber, и затем subscriber издает данные на любое число подписчиков. Это полезно когда publisher должен послать данные подписчикам по медленной или дорогой линии связи.

Типы **репликации**

Microsoft SQL Server 7.0/2000 поддерживает следующие виды **репликации:**

- Snapshot
- Transactional
- Merge

Snapshot репликация (снимок)

Является самой простой. При этом, все копируемые данные (точная копия) будут копироваться из базы данных publisher в базу (ы) данных subscriber/subscribers на периодической основе. Snapshot **репликация** является лучшим методом копирования данных, которые нечасто изменяются и когда размер копируемых данных не очень большой.

Transactional репликация

SQL Server фиксирует (делает моментальные снимки) все изменения, которые были сделаны в статье, и сохраняет, как: INSERT, UPDATE и DELETE инструкции в базе Distribution. Эти изменения посылаются подписчикам от Distribution и применяются к расположенным в них данным.

Transactional **репликации** лучше использовать, когда копируемые данные часто изменяются или когда размер копируемых данных достаточно велик и нет необходимости поддерживать автономные изменения реплицируемых данных относительно publisher и относительно subscriber.

Merge **репликация**

Является наиболее трудным типом **репликации**. Она предоставляет возможность автономных изменений реплицируемых данных и на publisher и на subscriber. При Merge **репликации**, SQL Server фиксирует все накопившиеся изменения не только в источнике данных, но и целевых базах данных, и урегулирует конфликты согласно правилам, которые Вы предварительно конфигурируете, или посредством определённого Вами блока принятия решений - resolver-ра.

Merge **репликацию** лучше использовать, когда Вы хотите обеспечить поддержку автономных изменений реплицируемых данных относительно publisher и относительно subscriber.

Агенты **Репликации**

Microsoft SQL Server 7.0/2000 поддерживает следующих агентов **репликации:**

- Snapshot Agent
- Log Reader Agent
- Distribution Agent
- Merge Agent

Snapshot Agent

Агент **репликации**, который создаёт файлы снимков, хранит снимки на distributor и производит запись информации о состоянии синхронизации в Distribution database. Snapshot Agent используется во всех типах **репликации** (Snapshot, Transactional и Merge) и может управляться из SQL Server Enterprise Manager.

Log Reader Agent

Агент **репликации**, который перемещает транзакции, отмеченные для **репликации** из transaction log, находящегося на publisher, в Distribution database. Этот агент **репликации** не используется в Snapshot **репликации**.

Distribution Agent

Агент **репликации**, который перемещает обрабатывающие снимки задания из Distribution database к подписчикам и перемещает все транзакции, ожидающие распределения на подписчиков. Distribution Agent используется в Snapshot и Transactional **репликациях** и может управляться с помощью SQL Server Enterprise Manager.

Merge Agent

Агент **репликации**, который применяет первоначальные, обрабатывающие снимки задания по таблицам базы данных publication на подписчиках, и потом объединяет возможные последующие изменения данных, которые произошли после создания первоначального снимка. Merge Agent используется только в Merge **репликации**

Резервное копирование

MS SQL поддерживает 3 типа backup’а данных

- Full backup
- Differential backup
- Transaction-log backup