**Модуль 3. Запросы SELECT, INSERT, UPDATE, DELETE (2 пары)**

**Цель занятия:** изучить работу операторов языка DML: SELECT, INSERT, UPDATE, DELETE; рассмотреть понятие транзакции.

**План занятия:**

1. Актуализация знаний.

2. Оператор SELECT:

* предложение SELECT;
* предложение FROM;
* предложение WHERE;
* предложение ORDER BY.

3. Ключевые слова IN, BETWEEN, LIKE.

4. Оператор INSERT.

5. Оператор UPDATE.

6. Оператор DELETE.

7. Понятие транзакции. Использование транзакций.

8. Домашнее задание.

1. **Задание на закрепление темы «Основы взаимодействия с MS SQL Server »**

Название SQL (Structured Query Language – язык структурированных запросов) отражает тот факт, что запросы являются наиболее часто используемым элементом SQL.

**Запрос** – это оператор, который посылает команду Системе Управления Базой Данных (СУБД) произвести манипуляцию или отобразить определенную информацию. Все запросы по выборке данных в SQL конструируются с помощью оператора SELECT. Он позволяет выполнять довольно сложные проверки и обработку данных.

Запрос может выводить данные из определенного столбца или изо всех столбцов таблицы. Чтобы создать простейших SELECT запрос, необходимо указать имя столбца и название таблицы.

**Синтаксис оператора SELECT**

**SELECT** column\_list   
**FROM** table\_name   
[**WHERE** условие]   
[**GROUP** **BY** условие]   
[**HAVING** условие]   
[**ORDER** **BY** условие]

SELECT Ключевое слово, которое сообщает базе данных о том, что оператор является запросом. Все запросы начинаются с этого слова, за ним следует пробел.

Column\_list Список столбцов таблицы, которые выбираются запросом. Столбцы, не указанные в операторе, не будут включены в результат. Если необходимо вывести данные всех столбцов, можно использовать сокращенную запись. Звездочка (\*) означает полный список столбцов.

FROM table\_name Ключевое слово, которое должно присутствовать в каждом запросе. После него через пробел указывается имя таблицы, являющейся источником данных.

Код в скобках является не обязательным в операторе SELECT. Он необходим для более точного определения запроса.

Также необходимо сказать, что SQL код является регистронезависимым. Это означает, что запись SELECT можно написать как select. СУБД не отличит эти две записи, однако советуют все операторы SQL писать прописными буквами, чтобы его легко можно было отличить от другого кода.

## Оператор SELECT.

Оператор Select служит для выборки данных. Общий синтаксис следующий:

Select имя\_поля1, имя\_поля2, имя\_поля3... имя\_поляN

From имя\_таблицы

[Where условие]

Если имя поля состоит из нескольких частей, или оно совпадает с ключевым словом языка SQL, то нужно имя этого поля заключать в двойные кавычки.

Выбираем требуемую базу данных (в нашем случае books) и пишем запрос.

Select N, Name, Price, Izd, Themes

From books

Запустив этот запрос на выполнение, вы увидите результат запроса на выборку. Первое поле N - первичный ключ, второе Name - название книги, Price - цена книги, Izd - издательство, выдавшее книги, Themes - тематика. Как видите, результат вы получаете полностью в соответствии с запросом.

Но не всегда удобно перечислять все имена полей, чтобы просмотреть все содержимое таблицы. Поэтому вместо перечисления столбцов можно указать \*, определяющая, что нужны значения всех полей.

Select \* from books

Именам полей, при вычитке, можно дать псевдоним. Например, чтоб вместо слова Name в заголовке было указано Название.

Для этого используется ключевое слово As. Общий синтаксис:

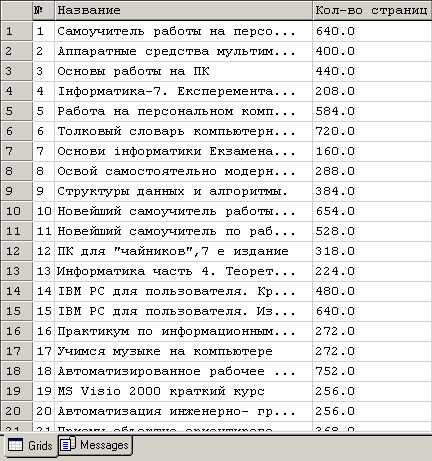
Select имя\_поля1 As 'Псевдоним1', имя\_поля2 As 'Псевдоним2'

from имя\_таблицы

Пример:

Select N as '№', Name as 'Название', Pages as 'Кол-во страниц'

From Books

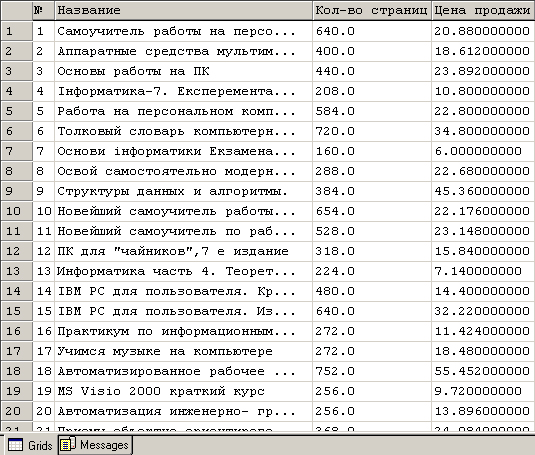


В запрос на выборку можно включать арифметические операции. Например, в базе хранится закупочная цена книг, а вам нужно показать цену, по которой вы их продаете (на 20% дороже, чем закупочная цена). Для этого достаточно выполнить такой запрос.

Select N as '№', Name as 'Название', Pages as 'Кол-во страниц', Price\*1.2 as 'Цена продажи'

From Books

При вычитке, каждое значение в поле Price будет увеличено на 20%. (Содержимое базы данных не изменится!!!).



Все арифметические операции выполняются по математическим правилам. Т.е. сначала деление, умножение, а потом сложение, вычитание. Для изменения порядка выполнения операции используются ( ).

### Добавление условий в запрос на выборку.

В любой запрос можно добавить проверку какого либо условия. Требуемое условие должно указываться поле ключевого слова Where.

Существуют следующие операторы сравнения, которые можно использовать в запросе:

* < - меньше
* > - больше
* <= - меньше либо равно
* >= - больше либо равно
* <> - проверка на неравенство
* = - проверка на равенство
* !> - не больше чем
* !< - не меньше чем
* is NULL - проверка на отсутствие записи

Например. Необходимо показать книги, у которых цена больше 50 рублей.

Select \*

From Books

Where Price>50

Для объединения условий используются операторы объединения:

* and - логическое И (общее условие будет верным, только если все объединенные условия верны)
* or - логическое Или (общее верно, если хотя бы одно из объединяемых условий верно)

Вы должны помнить, что при объединении запросов каждый из них должен быть полным. Т.е. запрос типа:

Select \*

From Books

Where Price>50 and <100

выдаст ошибку, т.к. условие , стоящее после and указано не полностью. Чтобы этот запрос сработал необходимо написать:

Правильно!!!

Select \*

From Books

Where Price>50 and Price<100

### Сортировка результата запроса на выборку.

Результат очень часто хочется видеть наглядным. Например, книги отсортированные по цене, либо по имени автора.

Для этого используется директива Order by. Сортировка может производиться как по возрастанию, так и по убыванию. По умолчанию установлена сортировка по возрастанию. Чтобы указать ее явно используется ключевое слово Asc. Для сортировки по убыванию указывается ключевое слово Desc. Существует 2 варианта написания директивы сортировки:

1. Select имя\_поля1, имя\_поля2, имя\_поля3

From имя\_таблицы

Where условие

Order by имя\_поля [Asc либо Desc]

Имя поля нужно указывать из тех, которые присутствуют в Select.

1. Select имя\_поля1, имя\_поля2, имя\_поля3
2. From имя\_таблицы
3. Where условие
4. Order by номер\_поля [Asc либо Desc]

Номер поля - порядковый номер поля, выводимого запросом.

Пример.

Select Name,Price

From books

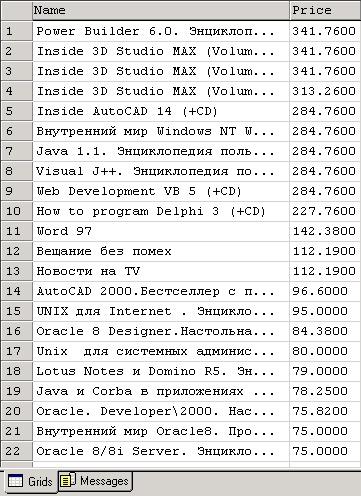
order by Price Desc

Полный аналог этого запроса.

Select Name,Price

From books

order by 2 Desc



Как видите, результат выборки отсортирован по второму столбцу (т.е. по цене) по убыванию.

## Ключевые слова IN, BETWEEN, LIKE.

### Ключевое слово In.

Ключевое слово In используется для проверки на наличие чего либо в множестве. Что имеется в виду. Задача: "Показать все книги, которые издавались издательствами Питер, Блиц, Бином и Афон". Используя изученное до этого получим запрос:

select name, izd

From books

Where Izd='Питер' or izd='Бином' or izd='Блиц' or izd='Афон'

Новый вариант этого запроса выглядит следующим образом:

select name, izd

From books

Where Izd in ('Питер', 'Бином', 'Блиц', 'Афон')

### Ключевое слово Between.

Between используется для проверки на принадлежность определенного значения указанному диапазону. Например, задача. Нужно показать все книги, у которых кол-во страниц больше 500, но меньше 650. Используя изученное до этого, получаем запрос.

Select name,pages

From books

Where pages>=500 and pages<=650

order by 2 asc

Новый вариант этого запроса:

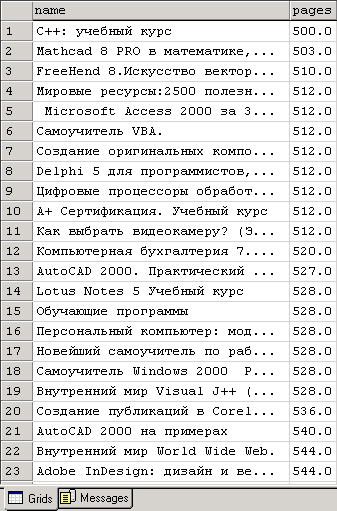
Select name,pages

From books

Where pages between 500 and 650

order by 2 asc

Результат:



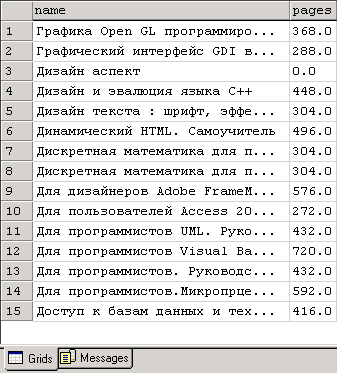
Также between можно использовать для работы с первой буквой строки. Но это будет проверка букв по ASCI кодам. Вот только при работе с цифрами верхний диапазон включается в диапазон проверки, а при работе с буквами - нет.

Select name,pages

From books

Where name between 'г' and 'е'

order by 1 asc



### Ключевое слово Like.

Like позволяет производить поиск по шаблону. Шаблон представляет из себя строку, в которой используются служебные символы. Общий синтаксис:

Select имя\_поля1, имя\_поля2, имя\_поля3

From имя\_таблицы

Where имя\_поля Like 'шаблон'

В шаблоне можно указывать такие служебные символы и комбинации символов:

1. % - заменяет любое кол-во любых символов в строке.
2. \_ - заменяет любой один символ.
3. [набор символов] - определяет, что на этом месте может стоять один из перечисленных символов.
4. [первый символ-второй символ] - определяет, что на этом месте может стоять один из символов в указанном диапазоне.
5. [^набор символов] - определяет, что на этом месте может стоять любой, кроме перечисленных символов.
6. [^первый символ-второй символ] - определяет, что на этом месте может стоять любой, кроме символов из указанного диапазона.

Пример 1. Необходимо вычитать все книги, в названии которых есть слово Словарь.

Select name,price

From books

Where name like '%словарь%'

Пример 2. Необходимо вычитать все названия книг, в названии которых первая буква либо А, либо З.

Select name,price

From books

Where name like '[АЗ]%'

### Not.

Перед каждым из этих перечисленных ключевых слов можно указать ключевое слово Not, отрицающее то, что указано после него.

Пример. Показать все книги, цена у которых не находится в диапазоне от 25 до 150.

Select name,price

From books

Where price not between 25 and 150

Order by 2 desc

Если язык DDL больше статичен, т.е. при помощи него создаются жесткие структуры (таблицы, связи и т.п.), то язык DML носит динамический характер, здесь правильные результаты вы можете получить разными путями.

Для тех, кто не создавал БД в первой части (т.к. не всех может интересовать язык DDL), может воспользоваться следующим скриптом:

Скрипт создания БД Test

*-- создание БД*

**CREATE** **DATABASE** **Test**

**GO**

*-- сделать БД Test текущей*

**USE** **Test**

**GO**

*-- создаем таблицы справочники*

**CREATE** **TABLE** Positions(

**ID** int **IDENTITY**(1,1) **NOT** NULL **CONSTRAINT** PK\_Positions PRIMARY **KEY**,

**Name** **nvarchar**(30) **NOT** NULL

)

**CREATE** **TABLE** Departments(

**ID** int **IDENTITY**(1,1) **NOT** NULL **CONSTRAINT** PK\_Departments PRIMARY **KEY**,

**Name** **nvarchar**(30) **NOT** NULL

)

**GO**

*-- заполняем таблицы справочники данными*

**SET** IDENTITY\_INSERT Positions **ON**

**INSERT** Positions(**ID**,**Name**)**VALUES**

(1,**N**'Бухгалтер'),

(2,**N**'Директор'),

(3,**N**'Программист'),

(4,**N**'Старший программист')

**SET** IDENTITY\_INSERT Positions **OFF**

**GO**

**SET** IDENTITY\_INSERT Departments **ON**

**INSERT** Departments(**ID**,**Name**)**VALUES**

(1,**N**'Администрация'),

(2,**N**'Бухгалтерия'),

(3,**N**'ИТ')

**SET** IDENTITY\_INSERT Departments **OFF**

**GO**

*-- создаем таблицу с сотрудниками*

**CREATE** **TABLE** Employees(

**ID** int **NOT** NULL,

**Name** **nvarchar**(30),

Birthday date,

Email **nvarchar**(30),

PositionID int,

DepartmentID int,

HireDate date **NOT** NULL **CONSTRAINT** DF\_Employees\_HireDate **DEFAULT** SYSDATETIME(),

ManagerID int,

**CONSTRAINT** PK\_Employees PRIMARY **KEY** (**ID**),

**CONSTRAINT** FK\_Employees\_DepartmentID FOREIGN **KEY**(DepartmentID) **REFERENCES** Departments(**ID**),

**CONSTRAINT** FK\_Employees\_PositionID FOREIGN **KEY**(PositionID) **REFERENCES** Positions(**ID**),

**CONSTRAINT** FK\_Employees\_ManagerID FOREIGN **KEY** (ManagerID) **REFERENCES** Employees(**ID**),

**CONSTRAINT** UQ\_Employees\_Email **UNIQUE**(Email),

**CONSTRAINT** CK\_Employees\_ID **CHECK**(**ID** **BETWEEN** 1000 **AND** 1999),

**INDEX** IDX\_Employees\_Name(**Name**)

)

**GO**

*-- заполняем ее данными*

**INSERT** Employees (**ID**,**Name**,Birthday,Email,PositionID,DepartmentID,ManagerID)**VALUES**

(1000,**N**'Иванов И.И.','19550219','i.ivanov@test.tt',2,1,NULL),

(1001,**N**'Петров П.П.','19831203','p.petrov@test.tt',3,3,1003),

(1002,**N**'Сидоров С.С.','19760607','s.sidorov@test.tt',1,2,1000),

(1003,**N**'Андреев А.А.','19820417','a.andreev@test.tt',4,3,1000)

## SELECT – оператор выборки данных

Первым делом, для активного редактора запроса, сделаем текущей БД Test, выбрав ее в выпадающем списке или же командой «USE Test». Начнем с самой элементарной формы SELECT:

**SELECT** \*

**FROM** Employees

В данном запросе мы просим вернуть все столбцы (на это указывает «\*») из таблицы Employees – можно прочесть это как «ВЫБЕРИ все\_поля ИЗ таблицы\_сотрудники». В случае наличия кластерного индекса, возвращенные данные, скорее всего будут отсортированы по нему, в данном случае по колонке ID (но это не суть важно, т.к. в большинстве случаев сортировку мы будем указывать в явном виде сами при помощи ORDER BY …):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Birthday** | **Email** | **PositionID** | **DepartmentID** | **HireDate** | **ManagerID** |
| 1000 | Иванов И.И. | 1955-02-19 | i.ivanov@test.tt | 2 | 1 | 2015-04-08 | NULL |
| 1001 | Петров П.П. | 1983-12-03 | p.petrov@test.tt | 3 | 3 | 2015-04-08 | 1003 |
| 1002 | Сидоров С.С. | 1976-06-07 | s.sidorov@test.tt | 1 | 2 | 2015-04-08 | 1000 |
| 1003 | Андреев А.А. | 1982-04-17 | a.andreev@test.tt | 4 | 3 | 2015-04-08 | 1000 |

Вообще стоит сказать, что в диалекте MS SQL самая простая форма запроса SELECT может не содержать блока FROM, в этом случае вы можете использовать ее, для получения каких-то значений:

**SELECT**

5550/100\*15,

SYSDATETIME(), *-- получение системной даты БД*

**SIN**(0)+**COS**(0)

|  |  |  |
| --- | --- | --- |
| **(No column name)** | **(No column name)** | **(No column name)** |
| 825 | 2015-04-11 12:12:36.0406743 | 1 |

Обратите внимание, что выражение (5550/100\*15) дало результат 825, хотя если мы посчитаем на калькуляторе получится значение (832.5). Результат 825 получился по той причине, что в нашем выражении все числа целые, поэтому и результат целое число, т.е. (5550/100) дает нам 55, а не (55.5).

Запомните следующее, что в MS SQL работает следующая логика:

Целое / Целое = Целое (т.е. в данном случае происходит целочисленное деление)

Вещественное / Целое = Вещественное

Целое / Вещественное = Вещественное

Т.е. результат преобразуется к большему типу, поэтому в 2-х последних случаях мы получаем вещественное число (рассуждайте как в математике – диапазон вещественных чисел больше диапазона целых, поэтому и результат преобразуется к нему):

**SELECT**

123/10, *-- 12*

123./10, *-- 12.3*

123/10. *-- 12.3*

Здесь (123.) = (123.0), просто в данном случае 0 можно отбросить и оставить только точку.

При других арифметических операциях действует та же самая логика, просто в случае деления этот нюанс более актуален.

Поэтому обращайте внимание на тип данных числовых столбцов. В том случае если он целый, а результат вам нужно получить вещественный, то используйте преобразование, либо просто ставьте точку после числа указанного в виде константы (123.).

Для преобразования полей можно использовать функцию CAST или CONVERT. Для примера воспользуемся полем ID, оно у нас типа int:

**SELECT**

**ID**,

**ID**/100, *-- здесь произойдет целочисленное деление*

**CAST**(**ID** **AS** float)/100, *-- используем функцию CAST для преобразования в тип float*

**CONVERT**(float,**ID**)/100, *-- используем функцию CONVERT для преобразования в тип float*

**ID**/100. *-- используем преобразование за счет указания что знаменатель вещественное число*

**FROM** Employees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **(No column name)** | **(No column name)** | **(No column name)** | **(No column name)** |
| 1000 | 10 | 10 | 10 | 10.000000 |
| 1001 | 10 | 10.01 | 10.01 | 10.010000 |
| 1002 | 10 | 10.02 | 10.02 | 10.020000 |
| 1003 | 10 | 10.03 | 10.03 | 10.030000 |

Так же не забываем, что в тексте запроса мы можем использовать как однострочные «-- …», так и многострочные «/\* … \*/» комментарии. Если запрос большой и сложный, то комментарии могут очень помочь, вам или кому-то другому, через некоторое время, вспомнить или разобраться в его структуре.

Если столбцов в таблице очень много, а особенно, если в таблице еще очень много строк, плюс к тому если мы делаем запросы к БД по сети, то предпочтительней будет выборка с непосредственным перечислением необходимых вам полей через запятую:

**SELECT** **ID**,**Name**

**FROM** Employees

Т.е. здесь мы говорим, что нам из таблицы нужно вернуть только поля ID и Name. Результат будет следующим (кстати оптимизатор здесь решил воспользоваться индексом, созданным по полю Name):

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1003 | Андреев А.А. |
| 1000 | Иванов И.И. |
| 1001 | Петров П.П. |
| 1002 | Сидоров С.С. |

## Задание псевдонимов для таблиц

При перечислении колонок их можно предварять именем таблицы, находящейся в блоке FROM:

**SELECT** Employees.**ID**,Employees.**Name**

**FROM** Employees

Но такой синтаксис обычно использовать неудобно, т.к. имя таблицы может быть длинным. Для этих целей обычно задаются и применяются более короткие имена – псевдонимы (alias):

**SELECT** emp.**ID**,emp.**Name**

**FROM** Employees **AS** emp

или  
**SELECT** emp.**ID**,emp.**Name**

**FROM** Employees emp *-- ключевое слово AS можно отпустить (я предпочитаю такой вариант)*

Здесь emp – псевдоним для таблицы Employees, который можно будет использоваться в контексте данного оператора SELECT. Т.е. можно сказать, что в контексте этого оператора SELECT мы задаем таблице новое имя.

Конечно, в данном случае результаты запросов будут точно такими же как и для «SELECT ID,Name FROM Employees». Для чего это нужно будет понятно дальше (даже не в этой части), пока просто запоминаем, что имя колонки можно предварять (уточнять) либо непосредственно именем таблицы, либо при помощи псевдонима. Здесь можно использовать одно из двух, т.е. если вы задали псевдоним, то и пользоваться нужно будет им, а использовать имя таблицы уже нельзя.

## DISTINCT – отброс строк дубликатов

Ключевое слово DISTINCT используется для того чтобы отбросить из результата запроса строки дубликаты. Грубо говоря представьте, что сначала выполняется запрос без опции DISTINCT, а затем из результата выбрасываются все дубликаты. Продемонстрируем это для большей наглядности на примере:

*-- создадим для демонстрации временную таблицу*

**CREATE** **TABLE** #Trash(

**ID** int **NOT** NULL PRIMARY **KEY**,

Col1 varchar(10),

Col2 varchar(10),

Col3 varchar(10)

)

*-- наполним данную таблицу всяким мусором*

**INSERT** #Trash(**ID**,Col1,Col2,Col3)**VALUES**

(1,'A','A','A'), (2,'A','B','C'), (3,'C','A','B'), (4,'A','A','B'),

(5,'B','B','B'), (6,'A','A','B'), (7,'A','A','A'), (8,'C','A','B'),

(9,'C','A','B'), (10,'A','A','B'), (11,'A',NULL,'B'), (12,'A',NULL,'B')

*-- посмотрим что возвращает запрос без опции DISTINCT*

**SELECT** Col1,Col2,Col3

**FROM** #Trash

*-- посмотрим что возвращает запрос с опцией DISTINCT*

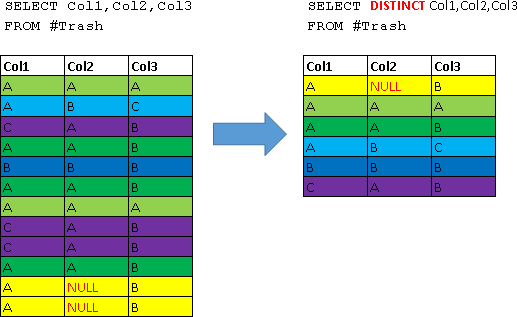
**SELECT** **DISTINCT** Col1,Col2,Col3

**FROM** #Trash

*-- удалим временную таблицу*

**DROP** **TABLE** #Trash

Наглядно это будет выглядеть следующим образом (все дубликаты помечены одним цветом):



Теперь давайте рассмотрим где это можно применить, на более практичном примере – вернем из таблицы Employees только уникальные идентификаторы отделов (т.е. узнаем ID отделов в которых числятся сотрудники):

**SELECT** **DISTINCT** DepartmentID

**FROM** Employees

|  |
| --- |
| **DepartmentID** |
| 1 |
| 2 |
| 3 |

Здесь мы получили три строки, т.к. 2 сотрудника у нас числятся в одном отделе (ИТ).

Теперь узнаем в каких отделах, какие должности фигурируют:

**SELECT** **DISTINCT** DepartmentID,PositionID

**FROM** Employees

|  |  |
| --- | --- |
| **DepartmentID** | **PositionID** |
| 1 | 2 |
| 2 | 1 |
| 3 | 3 |
| 3 | 4 |

Здесь мы получили 4 строчки, т.к. повторяющихся комбинаций (DepartmentID, PositionID) в нашей таблице нет.

## Ненадолго вернемся к DDL

Так как данных для демонстрационных примеров начинает не хватать, а рассказать хочется более обширно и понятно, то давайте чуть расширим нашу таблицу Employess. К тому же немного вспомним DDL, как говорится «повторение – мать учения», и плюс снова немного забежим вперед и применим оператор UPDATE:

*-- создаем новые колонки*

**ALTER** **TABLE** Employees **ADD**

LastName **nvarchar**(30), *-- фамилия*

FirstName **nvarchar**(30), *-- имя*

MiddleName **nvarchar**(30), *-- отчество*

Salary float, *-- и конечно же ЗП в каких-то УЕ*

BonusPercent float *-- процент для вычисления бонуса от оклада*

**GO**

*-- наполняем их данными (некоторые данные намерено пропущены)*

**UPDATE** Employees

**SET**

LastName=**N**'Иванов',FirstName=**N**'Иван',MiddleName=**N**'Иванович',

Salary=5000,BonusPercent= 50

**WHERE** **ID**=1000 *-- Иванов И.И.*

**UPDATE** Employees

**SET**

LastName=**N**'Петров',FirstName=**N**'Петр',MiddleName=**N**'Петрович',

Salary=1500,BonusPercent= 15

**WHERE** **ID**=1001 *-- Петров П.П.*

**UPDATE** Employees

**SET**

LastName=**N**'Сидоров',FirstName=**N**'Сидор',MiddleName=NULL,

Salary=2500,BonusPercent=NULL

**WHERE** **ID**=1002 *-- Сидоров С.С.*

**UPDATE** Employees

**SET**

LastName=**N**'Андреев',FirstName=**N**'Андрей',MiddleName=NULL,

Salary=2000,BonusPercent= 30

**WHERE** **ID**=1003 *-- Андреев А.А.*

Убедимся, что данные обновились успешно:

**SELECT** \*

**FROM** Employees

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **…** | **LastName** | **FirstName** | **MiddleName** | **Salary** | **BonusPercent** |
| 1000 | Иванов И.И. |  | Иванов | Иван | Иванович | 5000 | 50 |
| 1001 | Петров П.П. |  | Петров | Петр | Петрович | 1500 | 15 |
| 1002 | Сидоров С.С. |  | Сидоров | Сидор | NULL | 2500 | NULL |
| 1003 | Андреев А.А. |  | Андреев | Андрей | NULL | 2000 | 30 |

## Задание псевдонимов для столбцов запроса

Думаю, здесь будет проще показать, чем написать:

**SELECT**

*-- даем имя вычисляемому столбцу*

LastName+' '+FirstName+' '+MiddleName **AS** ФИО,

*-- использование двойных кавычек, т.к. используется пробел*

HireDate **AS** "Дата приема",

*-- использование квадратных скобок, т.к. используется пробел*

Birthday **AS** [Дата рождения],

*-- слово AS не обязательно*

Salary ZP

**FROM** Employees

|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО** | **Дата приема** | **Дата рождения** | **ZP** |
| Иванов Иван Иванович | 2015-04-08 | 1955-02-19 | 5000 |
| Петров Петр Петрович | 2015-04-08 | 1983-12-03 | 1500 |
| NULL | 2015-04-08 | 1976-06-07 | 2500 |
| NULL | 2015-04-08 | 1982-04-17 | 2000 |

Как видим заданные нами псевдонимы столбцов, отразились в заголовке результирующей таблицы. Собственно, это и есть основное предназначение псевдонимов столбцов.

Обратите внимание, т.к. у последних 2-х сотрудников не указано отчество (NULL значение), то результат выражения «LastName+' '+FirstName+' '+MiddleName» так же вернул нам NULL.

Для соединения (сложения, конкатенации) строк в MS SQL используется символ «+».

Запомним, что все выражения в которых участвует NULL (например, деление на NULL, сложение с NULL) будут возвращать NULL.

Для того чтобы не городить конструкцию с использованием функции ISNULL, в MS SQL мы можем применить функцию CONCAT. Рассмотрим и сравним 3 варианта:

**SELECT**

LastName+' '+FirstName+' '+MiddleName FullName1,

*-- 2 варианта для замены NULL пустыми строками '' (получаем поведение как и в ORACLE)*

**ISNULL**(LastName,'')+' '+**ISNULL**(FirstName,'')+' '+**ISNULL**(MiddleName,'') FullName2,

**CONCAT**(LastName,' ',FirstName,' ',MiddleName) FullName3

**FROM** Employees

|  |  |  |
| --- | --- | --- |
| **FullName1** | **FullName2** | **FullName3** |
| Иванов Иван Иванович | Иванов Иван Иванович | Иванов Иван Иванович |
| Петров Петр Петрович | Петров Петр Петрович | Петров Петр Петрович |
| NULL | Сидоров Сидор | Сидоров Сидор |
| NULL | Андреев Андрей | Андреев Андрей |

В MS SQL псевдонимы еще можно задавать при помощи знака равенства:

**SELECT**

'Дата приема'=HireDate, *-- помимо "…" и […] можно использовать '…'*

[Дата рождения]=Birthday,

ZP=Salary

**FROM** Employees

Использовать для задания псевдонима ключевое слово AS или же знак равенства, наверное, больше дело вкуса. Но при разборе чужих запросов, данные знания могут пригодиться.

Напоследок скажу, что для псевдонимов имена лучше задавать, используя только символы латиницы и цифры, избегая применения '…', "…" и […], то есть использовать те же правила, что мы использовали при наименовании таблиц. Дальше, в примерах я буду использовать только такие наименования и никаких '…', "…" и […].

# Основные арифметические операторы SQL

|  |  |
| --- | --- |
| **Оператор** | **Действие** |
| + | Сложение (x+y) или унарный плюс (+x) |
| - | Вычитание (x-y) или унарный минус (-x) |
| \* | Умножение (x\*y) |
| / | Деление (x/y) |
| % | Остаток от деления (x%y). Для примера 15%10 даст 5 |

Приоритет выполнения арифметических операторов такой же, как и в математике. Если необходимо, то порядок применения операторов можно изменить используя круглые скобки — (a+b)\*(x/(y-z)).

И еще раз повторюсь, что любая операция с NULL дает NULL, например: 10+NULL, NULL\*15/3, 100/NULL – все это даст в результате NULL. Т.е. говоря просто неопределенное значение не может дать определенный результат. Учитывайте это при составлении запроса и при необходимости делайте обработку NULL значений функциями ISNULL (Заменяет значение NULL указанным замещающим значением), COALESCE (Вычисляет аргументы в указанном порядке и возвращает текущее значение первого выражения, которое изначально не дает значение NULL.):

**SELECT**

**ID**,**Name**,

Salary/100\*BonusPercent **AS** Result1, *-- без обработки NULL значений*

Salary/100\***ISNULL**(BonusPercent,0) **AS** Result2, *-- используем функцию ISNULL*

Salary/100\***COALESCE**(BonusPercent,0) **AS** Result3 *-- используем функцию COALESCE*

**FROM** Employees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Name** | **Result1** | **Result2** | **Result3** |
| 1000 | Иванов И.И. | 2500 | 2500 | 2500 |
| 1001 | Петров П.П. | 225 | 225 | 225 |
| 1002 | Сидоров С.С. | NULL | 0 | 0 |
| 1003 | Андреев А.А. | 600 | 600 | 600 |
| 1004 | Николаев Н.Н. | NULL | 0 | 0 |
| 1005 | Александров А.А. | NULL | 0 | 0 |

Немного расскажу о функции COALESCE:

COALESCE (expr1, expr2, ..., exprn) - Возвращает первое не NULL значение из списка значений.

Пример:

**SELECT** **COALESCE**(f1, f1\*f2, f2\*f3) val *-- в данном случае вернется третье значение*

**FROM** (**SELECT** null f1, 2 f2, 3 f3) q

Для примера рассмотрим, как можно воспользоваться остатком от деления (%). Данный оператор очень полезен, когда требуется разбить записи на группы. Например, вытащим всех сотрудников, у которых четные табельные номера (ID), т.е. те ID, которые делятся на 2:

**SELECT** **ID**,**Name**

**FROM** Employees

**WHERE** **ID**%2=0 *-- остаток от деления на 2 равен 0*

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1000 | Иванов И.И. |
| 1004 | Николаев Н.Н. |
| 1002 | Сидоров С.С. |

## ORDER BY – сортировка результата запроса

Предложение ORDER BY используется для сортировки результата запроса.

**SELECT**

LastName,

FirstName,

Salary

**FROM** Employees

**ORDER** **BY** LastName,FirstName *-- упорядочить результат по 2-м столбцам – по Фамилии, и после по Имени*

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Salary** |
| Андреев | Андрей | 2000 |
| Иванов | Иван | 5000 |
| Петров | Петр | 1500 |
| Сидоров | Сидор | 2500 |

После имя поля в предложении ORDER BY можно задать опцию DESC, которая служит для сортировки этого поля в порядке убывания:

**SELECT** LastName,FirstName,Salary

**FROM** Employees

**ORDER** **BY** *-- упорядочить в порядке*

Salary **DESC**, *-- 1. убывания Заработной Платы*

LastName, *-- 2. по Фамилии*

FirstName *-- 3. по Имени*

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Salary** |
| Иванов | Иван | 5000 |
| Сидоров | Сидор | 2500 |
| Андреев | Андрей | 2000 |
| Петров | Петр | 1500 |

**Для заметки.** Для сортировки по возрастанию есть ключевое слово ASC, но так как сортировка по возрастанию применяется по умолчанию, то про эту опцию можно забыть (я не помню случая, чтобы я когда-то использовал эту опцию).

Сортировать можно так же используя разные выражения в предложении ORDER BY:

**SELECT** LastName,FirstName

**FROM** Employees

**ORDER** **BY** **CONCAT**(LastName,' ',FirstName) *-- используем выражение*

Так же в ORDER BY можно использовать псевдонимы заданные для колонок:

**SELECT** **CONCAT**(LastName,' ',FirstName) fi

**FROM** Employees

**ORDER** **BY** fi *-- используем псевдоним*

**Примечание 2.**В MS SQL при сортировке по возрастанию NULL значения будут отображаться первыми.

**SELECT** BonusPercent **FROM** Employees **ORDER** **BY** BonusPercent

Соответственно при использовании DESC они будут в конце

**SELECT** BonusPercent **FROM** Employees **ORDER** **BY** BonusPercent **DESC**

## TOP – возврат указанного числа записей

**Вырезка из MSDN.** **TOP** – ограничивает число строк, возвращаемых в результирующем наборе запроса до заданного числа или процентного значения. Если предложение TOP используется совместно с предложением ORDER BY, то результирующий набор ограничен первыми N строками отсортированного результата. В противном случае возвращаются первые N строк в неопределенном порядке.

Обычно данное выражение используется с предложением ORDER BY и мы уже смотрели примеры, когда нужно было вернуть N-первых строк из результирующего набора.

Без ORDER BY обычно данное предложение применяется, когда нужно просто посмотреть на неизвестную нам таблицу, в которой может быть очень много записей, в этом случае мы можем, для примера, попросить вернуть нам только первые 10 строк, но для наглядности мы скажем только 2:

**SELECT** TOP 2

\*

**FROM** Employees

Так же можно указать слово PERCENT, для того чтобы вернулось соответствуй процент строк из результирующего набора:

**SELECT** TOP 25 **PERCENT**

\*

**FROM** Employees

На моей практике чаше применяется именно выборка по количеству строк.

Так же с TOP можно использовать опцию WITH TIES, которая поможет вернуть все строки в случае неоднозначной сортировки, т.е. это предложение вернет все строки, которые равны по составу строкам, которые попадают в выборку TOP N, в итоге строк может быть выбрано больше чем N. Давайте для демонстрации добавим еще одного «Программиста» с окладом 1500:

**INSERT** Employees(**ID**,**Name**,Email,PositionID,DepartmentID,ManagerID,Salary)

**VALUES**(1004,**N**'Николаев Н.Н.','n.nikolayev@test.tt',3,3,1003,1500)

и введем еще одного сотрудника без указания должности и отдела с окладом 2000:

**INSERT** Employees(**ID**,**Name**,Email,PositionID,DepartmentID,ManagerID,Salary)

**VALUES**(1005,**N**'Александров А.А.','a.alexandrov@test.tt',NULL,NULL,1000,2000)

А что же будет если применить одновременно предложения DISTINCT и TOP? На такие вопросы легко ответить, проводя эксперименты. В общем, не бойтесь и не ленитесь экспериментировать, т.к. большая часть познается именно на практике. Порядок слов в операторе SELECT следующий, первым идет DISTINCT, а после него идет TOP, т.е. если рассуждать логически и читать слева-направо, то первым применится отброс дубликатов, а потом уже по этому набору будет сделан TOP. Что-ж проверим и убедимся, что так и есть:

**SELECT** **DISTINCT** TOP 2

Salary

**FROM** Employees

**ORDER** **BY** Salary

|  |
| --- |
| **Salary** |
| 1500 |
| 2000 |

Т.е. в результате мы получили 2 самые маленькие зарплаты из всех. Конечно может быть случай что ЗП для каких-то сотрудников может быть не указанной (NULL), т.к. схема нам это позволяет. Поэтому в зависимости от задачи принимаем решение либо обработать NULL значения в предложении ORDER BY, либо просто отбросить все записи, у которых Salary равна NULL, а для этого переходим к изучению предложения WHERE.

## WHERE – условие выборки строк

Данное предложение служит для фильтрации записей по заданному условию. Например, выберем всех сотрудников работающих в «ИТ» отделе (его ID=3):

**SELECT** **ID**,LastName,FirstName,Salary

**FROM** Employees

**WHERE** DepartmentID=3 *-- ИТ*

**ORDER** **BY** LastName,FirstName

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Salary** |
| 1004 | NULL | NULL | 1500 |
| 1003 | Андреев | Андрей | 2000 |
| 1001 | Петров | Петр | 1500 |

Предложение WHERE пишется до команды ORDER BY.

Порядок применения команд к исходному набору Employees следующий:

WHERE – если указано, то первым делом из всего набора Employees идет отбор только удовлетворяющих условию записей

1. DISTINCT – если указано, то отбрасываются все дубликаты
2. ORDER BY – если указано, то делается сортировка результата
3. TOP – если указано, то из отсортированного результата возвращается только указанное число записей

Рассмотрим для наглядности пример:

**SELECT** **DISTINCT** TOP 1

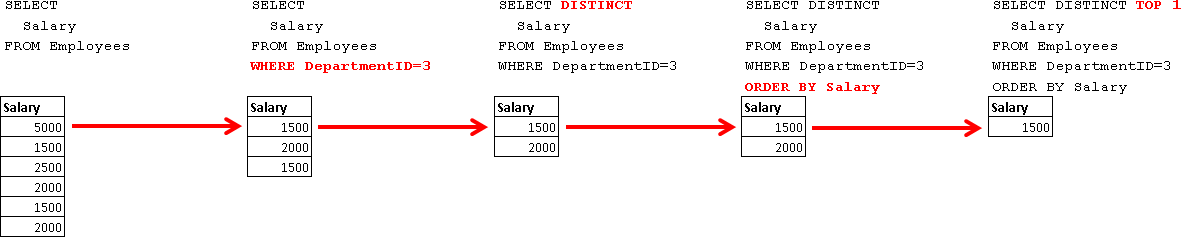
Salary

**FROM** Employees

**WHERE** DepartmentID=3

**ORDER** **BY** Salary

Наглядно это будет выглядеть следующим образом:



Стоит отметить, что проверка на NULL делается не знаком равенства, а при помощи операторов IS NULL и IS NOT NULL. Просто запомните, что на NULL при помощи оператора «=» (знак равенства) сравнивать нельзя, т.к. результат выражения будет так же равен NULL.

Например, выберем всех сотрудников, у которых не указан отдел (т.е. DepartmentID IS NULL):

**SELECT** **ID**,**Name**

**FROM** Employees

**WHERE** DepartmentID **IS** NULL

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1005 | Александров А.А. |

Теперь для примера посчитаем бонус для всех сотрудников у которых указано значение BonusPercent (т.е. BonusPercent IS NOT NULL):

**SELECT** **ID**,**Name**,Salary/100\*BonusPercent **AS** Bonus

**FROM** Employees

**WHERE** BonusPercent **IS** **NOT** NULL

Да, кстати, если подумать, то значение BonusPercent может равняться нулю (0), а так же значение может быть внесено со знаком минус, ведь мы не накладывали на данное поле никаких ограничений.

Хорошо, рассказав о проблеме, нам пока сказали считать, что если (BonusPercent<=0 или BonusPercent IS NULL), то это означает что у сотрудника так же нет бонуса. Для начала, как нам сказали, так и сделаем, реализуем это при помощи логического оператора OR и NOT:

**SELECT** **ID**,**Name**,Salary/100\*BonusPercent **AS** Bonus

**FROM** Employees

**WHERE** **NOT**(BonusPercent<=0 **OR** BonusPercent **IS** NULL)

Т.е. здесь мы начали изучать булевы операторы. Выражение в скобках «(BonusPercent<=0 OR BonusPercent IS NULL)» проверяет на то что у сотрудника нет бонуса, а NOT инвертирует это значение, т.е. говорит «верни всех сотрудников которые не сотрудники у которых нет бонуса».

Так же данное выражение можно переписать и сразу сказав сразу «верни всех сотрудников, у которых есть бонус» выразив это выражением (BonusPercent>0 и BonusPercent IS NOT NULL):

**SELECT** **ID**,**Name**,Salary/100\*BonusPercent **AS** Bonus

**FROM** Employees

**WHERE** BonusPercent>0 **AND** BonusPercent **IS** **NOT** NULL

Также в блоке WHERE можно делать проверку разного рода выражений с применением арифметических операторов и функций. Например, аналогичную проверку можно сделать, использовав выражение с функцией ISNULL:

**SELECT** **ID**,**Name**,Salary/100\*BonusPercent **AS** Bonus

**FROM** Employees

**WHERE** **ISNULL**(BonusPercent,0)>0

## Булевы операторы и простые операторы сравнения

Да, без математики здесь не обойтись, поэтому сделаем небольшой экскурс по булевым и простым операторам сравнения. Булевых операторов в языке SQL всего 3 – AND, OR и NOT:

|  |  |
| --- | --- |
| **AND** | логическое И. Ставится между двумя условиями (условие1 AND условие2). Чтобы выражение вернуло True, нужно, чтобы истинными были оба условия |
| **OR** | логическое ИЛИ. Ставится между двумя условиями (условие1 OR условие2). Чтобы выражение вернуло True, достаточно, чтобы истинным было только одно условие |
| **NOT** | инвертирует условие/логическое\_выражение. Накладывается на другое выражение (NOT логическое\_выражение) и возвращает True, если логическое\_выражение = False и возвращает False, если логическое\_выражение = True |

Для каждого булева оператора можно привести таблицы истинности где дополнительно показано какой будет результат, когда условия могут быть равны NULL:

Есть следующие простые операторы сравнения, которые используются для формирования условий:

|  |  |
| --- | --- |
| **Условие** | **Значение** |
| = | Равно |
| < | Меньше |
| > | Больше |
| <= | Меньше или равно |
| >= | Больше или равно |
| <> != | Не равно |

Плюс имеются 2 оператора для проверки значения/выражения на NULL:

|  |  |
| --- | --- |
| **IS NULL** | Проверка на равенство NULL |
| **IS NOT NULL** | Проверка на неравенство NULL |

Приоритет: 1) Все операторы сравнения; 2) NOT; 3) AND; 4) OR.

При построении сложных логических выражений используются круглые скобки:

((условие1 AND условие2) OR NOT(условие3 AND условие4 AND условие5)) OR (…)

Так же при помощи использования круглых скобок, можно изменить стандартную последовательность вычислений.

## BETWEEN – проверка на вхождение в диапазон

Этот оператор имеет следующий вид:

проверяемое\_значение [NOT] BETWEEN начальное\_ значение AND конечное\_ значение

В роли значений могут выступать выражения.

Разберем на примере:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** Salary **BETWEEN** 2000 **AND** 3000 *-- у кого ЗП в диапазоне 2000-3000*

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Salary** |
| 1002 | Сидоров С.С. | 2500 |
| 1003 | Андреев А.А. | 2000 |
| 1005 | Александров А.А. | 2000 |

Собственно, BETWEEN это упрощенная запись вида:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** Salary>=2000 **AND** Salary<=3000 *-- все у кого ЗП в диапозоне 2000-3000*

Перед словом BETWEEN может использоваться слово NOT, которое будет осуществлять проверку значения на не вхождение в указанный диапазон:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** Salary **NOT** **BETWEEN** 2000 **AND** 3000 *-- аналогично выражению NOT(Salary>=2000 AND Salary<=3000)*

Соответственно, в случае использования BETWEEN, IN, LIKE вы можете так же объединять их с другими условиями при помощи AND и OR:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** Salary **BETWEEN** 2000 **AND** 3000 *-- у кого ЗП в диапазоне 2000-3000*

**AND** DepartmentID=3 *-- учитывать сотрудников только отдела 3*

## IN – проверка на вхождение в перечень значений

Этот оператор имеет следующий вид:

проверяемое\_значение [NOT] IN (значение1, значение2, …)

Думаю, проще показать на примере:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** PositionID **IN**(3,4) *-- у кого должность равна 3 или 4*

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Salary** |
| 1001 | Петров П.П. | 1500 |
| 1003 | Андреев А.А. | 2000 |
| 1004 | Николаев Н.Н. | 1500 |

Т.е. по сути это аналогично следующему выражению:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** PositionID=3 **OR** PositionID=4 *-- у кого должность равна 3 или 4*

В случае NOT это будет аналогично (получим всех кроме тех, кто из отдела 3 и 4):

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** PositionID **NOT** **IN**(3,4) *-- аналогично выражению NOT(PositionID=3 OR PositionID=4)*

Так же запрос с NOT IN можно выразить и через AND:

**SELECT** **ID**,**Name**,Salary

**FROM** Employees

**WHERE** PositionID<>3 **AND** PositionID<>4 *-- равносильно PositionID NOT IN(3,4)*

Учтите, что искать NULL значения при помощи конструкции IN не получится, т.к. проверка NULL=NULL вернет так же NULL, а не True:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** DepartmentID **IN**(1,2,NULL) *-- NULL записи не войдут в результат*

В этом случае разбивайте проверку на несколько условий:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** DepartmentID **IN**(1,2) *-- 1 или 2*

**OR** DepartmentID **IS** NULL *-- или NULL*

Или же можно написать что-то вроде:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** **ISNULL**(DepartmentID,-1) **IN**(1,2,-1) *-- если вы уверены, что в нет и не будет департамента с ID=-1*

Думаю, первый вариант, в данном случае будет более правильным и надежным. Ну ладно, это всего лишь пример, для демонстрации того какие еще конструкции можно строить.

Так же стоит упомянуть еще более коварную ошибку, связанную с NULL, которую можно допустить при использовании конструкции NOT IN. Для примера, давайте попробуем выбрать всех сотрудников, кроме тех, у которых отдел равен 1 или у которых отдел вообще не указан, т.е. равен NULL. В качестве решения напрашивается вариант:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** DepartmentID **NOT** **IN**(1,NULL)

Но выполнив запрос, мы не получим ни одной строки, хотя мы ожидали увидеть следующее:

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **DepartmentID** |
| 1001 | Петров П.П. | 3 |
| 1002 | Сидоров С.С. | 2 |
| 1003 | Андреев А.А. | 3 |
| 1004 | Николаев Н.Н. | 3 |

Опять же шутку здесь сыграло NULL указанное в списке значений.

Разберем почему в данном случае возникла логическая ошибка. Разложим запрос при помощи AND:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** DepartmentID<>1

**AND** DepartmentID<>NULL *-- проблема из-за этой проверки на NULL - это условие всегда вернет NULL*

Правое условие (DepartmentID<>NULL) нам всегда здесь даст неопределенность, т.е. NULL. Теперь вспомним таблицу истинности для оператора AND, где (TRUE AND NULL) дает NULL. Т.е. при выполнении левого условия (DepartmentID<>1) из-за неопределенного правого условия в результате мы получим неопределенное значение всего выражения (DepartmentID<>1 AND DepartmentID<>NULL), поэтому строка не войдет в результат.

Переписать условие правильно можно следующим образом:

**SELECT** **ID**,**Name**,DepartmentID

**FROM** Employees

**WHERE** DepartmentID **NOT** **IN**(1) *-- или в данном случае просто DepartmentID<>1*

**AND** DepartmentID **IS** **NOT** NULL *-- и отдельно проверяем на NOT NULL*

## LIKE – проверка строки по шаблону

Про данный оператор я расскажу только в самом простом виде, который является стандартом и поддерживается большинством диалектов языка SQL. Даже в таком виде при помощи него можно решить много задач, которые требуют выполнить проверку по содержимому строки.

Этот оператор имеет следующий вид:

проверяемая\_строка [NOT] LIKE строка\_шаблон [ESCAPE отменяющий\_символ]

В «строке\_шаблон» могут применятся следующие специальные символы:

1. Знак подчеркивания «\_» — говорит, что на его месте может стоять любой единичный символ
2. Знак процента «%» — говорит, что на его месте может стоять сколько угодно символов, в том числе и ни одного

Рассмотрим примеры с символом «%» (на практике, кстати он чаще применяется):

**SELECT** **ID**,**Name**

**FROM** Employees

**WHERE** **Name** **LIKE** 'Пет%' *-- у кого имя начинается с букв "Пет"*

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** LastName **LIKE** '%ов' *-- у кого фамилия оканчивается на "ов"*

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** LastName **LIKE** '%ре%' *-- у кого фамилия содержит сочетание "ре"*

Рассмотрим примеры с символом «\_»:

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** LastName **LIKE** '\_етров' *-- у кого фамилия состоит из любого первого символа и последующих букв "етров"*

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** LastName **LIKE** '\_\_\_\_ов' *-- у кого фамилия состоит из четырех любых символов и последующих букв "ов"*

При помощи ESCAPE можно задать отменяющий символ, который отменяет проверяющее действие специальных символов «\_» и «%». Данное предложение используется, когда в строке нужно непосредственно проверить наличие знака процента или знака подчеркивания.

Для демонстрации ESCAPE давайте занесем в одну запись мусор:

**UPDATE** Employees

**SET**

FirstName='Это\_мусор, содержащий %'

**WHERE** **ID**=1005

И посмотрим, что вернут следующие запросы:

**SELECT** \*

**FROM** Employees

**WHERE** FirstName **LIKE** '%!%%' ESCAPE '!' *-- строка содержит знак "%"*

**SELECT** \*

**FROM** Employees

**WHERE** FirstName **LIKE** '%!\_%' ESCAPE '!' *-- строка содержит знак "\_"*

В случае, если требуется проверить строку на полное совпадение, то вместо LIKE лучше использовать просто знак «=»:

**SELECT** \*

**FROM** Employees

**WHERE** FirstName='Петр'

## Немного о строках

В случае проверки строки на наличие Unicode символов, нужно будет ставить перед кавычками символ N, т.е. N'…'. Но так как у нас в таблице все символьные поля в формате Unicode (тип nvarchar), то для этих полей можно всегда использовать такой формат. Пример:

**SELECT** **ID**,**Name**

**FROM** Employees

**WHERE** **Name** **LIKE** **N**'Пет%'

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** LastName=**N**'Петров'

Если делать правильно, при сравнении с полем типа varchar (ASCII) нужно стараться использовать проверки с использованием '…', а при сравнении поля с типом nvarchar (Unicode) нужно стараться использовать проверки с использованием N'…'. Это делается для того, чтобы избежать в процессе выполнения запроса неявных преобразований типов. То же самое правило используем при вставке (INSERT) значений в поле или их обновлении (UPDATE).

При сравнении строк стоит учесть момент, что в зависимости от настройки БД (collation), сравнение строк может быть, как регистро-независимым (когда 'Петров'='ПЕТРОВ'), так и регистро-зависимым (когда 'Петров'<>'ПЕТРОВ').  
В случае регистро-зависимой настройки, если требуется сделать поиск без учета регистра, то можно, например, сделать предварительное преобразование правого и левого выражения в один регистр – верхний или нижний:

**SELECT** **ID**,**Name**

**FROM** Employees

**WHERE** **UPPER**(**Name**) **LIKE** **UPPER**(**N**'Пет%') *-- или LOWER(Name) LIKE LOWER(N'Пет%')*

**SELECT** **ID**,LastName

**FROM** Employees

**WHERE** **UPPER**(LastName)=**UPPER**(**N**'Петров') *-- или LOWER(LastName)=LOWER(N'Петров')*

## Немного о датах

При проверке на дату, вы можете использовать, как и со строками одинарные кавычки '…'.

Вне зависимости от региональных настроек в MS SQL можно использовать следующий синтаксис дат 'YYYYMMDD' (год, месяц, день слитно без пробелов). Такой формат даты MS SQL поймет всегда:

**SELECT** **ID**,**Name**,Birthday

**FROM** Employees

**WHERE** Birthday **BETWEEN** '19800101' **AND** '19891231' *-- сотрудники 80-х годов*

**ORDER** **BY** Birthday

В некоторых случаях, дату удобнее задавать при помощи функции DATEFROMPARTS:

**SELECT** **ID**,**Name**,Birthday

**FROM** Employees

**WHERE** Birthday **BETWEEN** **DATEFROMPARTS**(1980,1,1) **AND** **DATEFROMPARTS**(1989,12,31)

**ORDER** **BY** Birthday

Так же есть аналогичная функция DATETIMEFROMPARTS, которая служит для задания Даты и Времени (для типа datetime).

Еще вы можете использовать функцию CONVERT, если требуется преобразовать строку в значение типа date или datetime:

**SELECT**

**CONVERT**(date,'12.03.2015',104),

**CONVERT**(datetime,'2014-11-30 17:20:15',120)

Значения 104 и 120, указывают какой формат даты используется в строке. Описание всех допустимых форматов вы можете найти в библиотеке MSDN задав в поиске «MS SQL CONVERT».

Функций для работы с датами в MS SQL очень много, ищите «ms sql функции для работы с датами».

**Примечание.** Во всех диалектах языка SQL свой набор функций по работе с датами и применяется свой подход по работе с ними.

## Оператор INSERT.

Оператор Insert служит для добавления данных в таблицу. Существует два варианта синтаксиса запроса на добавление информации.

Insert into имя\_таблицы

values (значение1, значение2, значение3... значениеN)

В этом случае кол-во перечисленных значений должно строго соответствовать порядку и кол-ву полей в указанной таблице.

Insert into имя\_таблицы (имя\_поля1, имя\_поля2, имя\_поля3)

values (значение1, значение2, значение3)

В этом случае, кол-во значений должно строго совпадать кол-ву перечисленных столбцов.

Пример на первый вариант.

Insert into books

Values(769, 4895 , 1, 'Справочник по С 4-ое издание', 68.93, 'Питер', 685, '180\*240/30', 2005-08-29, 5000, 'Программирование', 'Справочник')

Вы должны помнить, что любая строка, указываемая в запросе должна быть заключена в ОДИНАРНЫЕ КАВЫЧКИ. Числа, указываемые в запросе, в кавычки НЕ ЗАКЛЮЧАЮТСЯ!!!

При таком синтаксисе, запрос на добавление можно вызвать не более одного раза, т.к. в любой таблице должно существовать поле, являющиеся первичным ключом, значения в котором повторяться не могут. В примере запроса туда записывается значение 769. При попытке второй раз сохранить значение 769 в поле, являющиеся первичным ключом приведет к ошибке. (Учитывая, что база данных books импортировалась, первичный ключ в таблице books отсутствует.)

Иногда требуется заполнить только несколько столбцов из всей таблицы. Для этого используется второй вариант синтаксиса запроса на вставку. Но здесь возможна ситуация, когда некоторые поля в таблице будут являться обязательными для заполнения. В этом случае необходимо их также перечислять при заполнении.

Insert into books(N,New,Name,Date,Izd)

Values(770,1,'Microsoft SQL Server 2000',2005-08-29,'Бином')

**Оператор UPDATE.**

Служит для обновления (изменения) информации в файле.

Общий синтаксис:

Update имя\_таблицы

Set имя\_поля1=значение1, имя\_поля2=значение2, имя\_поля3=значение3

Where условие

Пример1. Необходимо увеличить цену книг новинок на 20%.

Update books

Set price=price\*1.2

where new=1

Если не указать условие, то изменение будет произведено со всеми строками в указанных полях.

Пример2. Необходимо установить категорию "неизвестная", там, где категория не указана.

Update books

Set category='Неизвестная'

where category is NULL

При помощи Update можно очистить содержимое столбца, указав имя\_поля=NULL и не указав условие.

**Оператор DELETE.**

Оператор Delete используется для удаления записей (строк) из таблицы.

Общий синтаксис:

Delete from имя\_таблицы

Where условие.

Учтите, что после удаления восстановить информацию невозможно.

**ВНИМАНИЕ!!!** Если не указать условие в запросе на удаление, то удалится вся информация из указанной таблицы.

Пример1. Необходимо удалить все книги, дата издания у которых неизвестна.

Delete from books

Where date is null

Пример2. Необходимо удалить все справочники (в названии категории есть слово справочник).

Delete from books

Where category like '%справочник%'

1. **Понятие транзакции. Свойства транзакции.**

После вступительной беседы, где нужно акцентировать внимание на недостатки выполнения сценариев как не контролированной группы инструкций перейти к рассмотрению понятия транзакции и ее основных свойств.

Прежде чем давать определение, выяснить у студентов их понимание транзакции и на примерах продемонстрировать применение транзакционных процессов, которые они уже наблюдали в обыденной жизни (например, сообщение в банкоматах о выполненной или не выполненной успешно транзакции).

Привести примеры процессов, которые можно назвать транзакционными: банковские операции двух операторов одновременно, выдача хвостовки и т.д.

Подчеркнуть, что существуют такие ситуации, для разрешения которых нужно предусмотреть способ, который может обеспечить возможность построения группы действий , которые будут выполняться по-принципу «все или ничего», т.е. будут выполнены все действия группы либо они будут отменены.

Далее подробно обсудить определение транзакции. Транзакция - это способ, с помощью которого есть возможность выполнить объединения определенного количества операций СУБД SQL Server в одну конструкцию. Особенно важно подчеркнуть, что данная конструкция выполняет подтверждение или отмену операций, которые описаны в блоке.

При рассмотрении определения транзакции особенно важно остановиться на ее свойствах, которые представляют собой методологию их использования.

Выделить четыре основных свойства транзакции Atomicity, Consistency, Isolation, Durability и рассмотреть их последовательно.

1. Atomicity(Атомарность) - неделимость блока операторов.

Все модификации, произведенные с данными в транзакции, в случае успеха фиксируются, если же модификация выполнена не успешно, что не фиксируется ни одна из выполненных инструкций. Вся последовательность инструкций должна выполняться как одна (атомарная) .

При выполнении одного или несколько запросов всегда говорят о целостности данных. Такое же свойство, как наследство является обязательным и для транзакции.

1. Consistency (логичность, последовательность, связность) – поддержка целостности данных.

Здесь обсудить уже изученные возможность поддержки целостности данных, например, с помощью ограничений. А также возможности, которые предлагает механизм транзакций для ее обеспечения.

1. Isolation (изолированность)

Акцентировать внимание студентов на то, что транзакции не могут взаимодействовать между собой, что должно учитываться при их разработке и использовании. Данной свойство позволяет организовать их надежность и эффективность.

1. Durability (надежность, живучесть) **-**  независимость от внешнего воздействия.

Рассмотреть еще одно из самых полезных качеств транзакции – это возможность восстановления базы данных при отказах или потере данных в процессе работы. Для этого можно использовать резервные копии журналов, конечно же, транзакций.

Подчеркнуть в этом случае важность механизма работы СУБД посредством таких журналов, а также важность обеспечения защиты и резервирования таких объектов.

Перед переходом к рассмотрению различных режимов работы транзакций указать, что любой запрос можно считать транзакцией в ее минимально-возможном представлении. Запрос невозможно выполнить частично.

1. **Режимы транзакций.**

После рассмотрения теоретической водной часть перейти к рассмотрению применения и реализации транзакций. Начать рассмотрение с указания различных способов (режимов) использования транзакций и особенностей синтаксиса и принципов работы этих способов.

Транзакция может быть выполнена в одном из трех режимов:

- автофиксация (autocommit),

- явный режим (explicit)

- неявный режим (implicit).

По умолчанию для SQL Server принят режим автофиксации. Перейти к рассмотрению каждого из режимов.

Режим автофиксации

В этом режиме каждый оператор будет зафиксирован после того как будет завершен. Никаких дополнительных операторов в этом режиме не используется.

Т.е., как было сказано раннее, один оператор всегда представляет собой минимальную единицу транзакции и будет фиксироваться сразу после его завершения.

Данный режим по-умолчанию будет актуален до тех пор пока вы не активируете другие режимы.

Явный режим

Начать рассмотрение данного режима с указанием того, где актуально применение данного режима и его преимущества над предыдущим режимом. Например, для приложений, которые работают с СУБД, организации работы других групп инструкций (хранимые процедуры, триггеры, сценарии) желательно применение именно явного режима.

Для применения транзакции в вышеупомянутых ситуациях требуется указанные начала и завершения такой группы операторов, возможность выполнения фиксации или отката выполненных действий.

Соответственно, при явном указании начала и конца транзакции используется явный режим. Явная транзакция может задаваться с помощью операторов T-SQL или с помощью функций API.

Для запуска транзакции используется операторы

BEGIN TRAN[SACTION] [имя\_транзакции | @переменная\_с\_именем\_транзакции]

COMMIT [TRAN[SACTION] [имя\_транзакции | @переменная\_с\_именем\_транзакции]]

ROLLBACK [TRAN[SACTION] [имя\_транзакции | @переменная\_с\_именем\_транзакции

| имя\_точки\_сохранения | @переменная\_с\_именем\_точки\_сохранения]]

Далее привести примеры транзакций, их фиксирования и откатов.

Подчеркнуть важность механизма фиксирования транзакций и возможность их восстановления.

Фиксированной можно назвать транзакцию, у которой все модификации стали частью базы данных. Записи о модификации вносятся в журнал транзакций.

Соответственно, все данные модификации могут быть помещены на диск или в кеш-память. Данный механизм позволяет в случае сбоя произвести повтор транзакции. Используя при этом журнал транзакции, избегая при этом потерю информации.

Используемые транзакцией ресурсы (например блокировки) будут освобождаться после того как зафиксирована транзакция.

USE library

GO

BEGIN TRAN update\_t\_cards

UPDATE T\_Cards SET DateIn = GETDATE()

WHERE DateIn IS NULL

COMMIT TRAN update\_t\_cards

GO

Обратить внимание студентов на эту транзакцию и указать, что это повлияло на две строки. Если будет потребность вернуть таблицу к ее исходному состоянию (как если бы вместо фиксирования произошел откат), выполните следующую транзакцию:

USE library

GO

BEGIN TRAN undo\_update\_t\_cards

UPDATE T\_Cards SET DateIn = NULL

WHERE library.dbo.transformaDate(DateIn)=library.dbo.transformaDate(GETDATE())

COMMIT TRAN undo\_update\_t\_cards

GO

update\_state (модифицировать\_состояние) и undo\_update\_state (отменить\_модификацию\_состояния), это имена транзакций. Но их использование в операторе COMMIT TRAN, будет СУБД в SQL Server проигнорировано. В большей степени эти имена используются для удобства программиста, например так можно указать имя транзакции, которая фиксируется.

Неявный режим

Для того, чтобы начать неявный режим используются определенные операторы T-SQL. И то тех пор пока не появится оператор, который сигнализирует окончание транзакции все операторы будут входить в ее группу. Операторы окончания транзакции - COMMIT или ROLLBACK. Если ни же один из приведенных операторов не был указан, то происходит откат указанной транзакции.

Следующие операторы T-SQL являются началом новой транзакции в неявном режиме:

ALTER TABLE, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE, UPDATE

Если внутри транзакции снова встретятся эти операторы эта транзакция продолжается После явного фиксирования или отката данной транзакции следующее появление этих операторов является началом новой транзакции. Этот процесс продолжает действовать, пока не будет отключен неявный режим.

Чтобы задать неявный режим транзакций, вы можете использовать следующий оператор T-SQL:

SET IMPLICIT\_TRANSACTIONS {ON | OFF}

Значение ON активизирует неявный режим, и OFF отключает его. После отключения неявного режима используется режим автофиксации.

Акцентировать внимание студентов, что такие транзакции актуальны в сценариях, которые модифицируют данные, которые требуют защиты внутри транзакции. Есть возможность включить неявный режим в начале сценария, выполнить необходимые модификации и отключить этот режим в конце сценария.

Также в СУБД В SQL Server возможны вложения транзакций, т.е. транзакции внутри транзакции.

Рассмотреть пример вложенной транзакции, включающей в себя хранимую процедуру.

USE library

GO

CREATE PROCEDURE Place\_Order --Создает хранимую процедуру

AS

BEGIN TRAN place\_order\_tran

PRINT 'Здесь должны быть SQL-операторы, выполняющие задачи по заказам'

COMMIT TRAN place\_order\_tran

GO

BEGIN TRAN Order\_tran --Начинает внешнюю транзакцию

PRINT 'Поместите заказ'

EXEC Place\_Order --Вызывает хранимую процедуру, которая --начинает внутреннюю транзакцию

COMMIT TRAN Order\_tran --Фиксирует внутреннюю и внешнюю --транзакции

GO

Указать, что после выполнения этой программы, можно получить результаты двух операторов PRINT. Транзакция place\_order\_tran обязана включать оператор COMMIT. Эта операция отмечет завершение этой транзакции, но как выясняется данное фиксирование не будет происходить. Так будет до тех пор, пока не будет выполнено фиксирование транзакции Order\_tran. Фиксирование или откат place\_order\_tran будет зависеть только от фиксирования Order\_tran.

1. **Откаты и точки сохранения**

Откаты транзакций

Следующий этап посвятить более детальному рассмотрению окатов транзакций и их видов. Указать, что откаты могут быть в двух видах: автоматический откат ( выполняется СУБД SQL Server) и программируемый вручную.

Подчеркнуть, что для обеспечения целостности, надежности, согласованности советуется явно применять операторы для отката транзакций.

Автоматические откаты

Указать некоторые ситуации, при которых будет задействован первый способ откатов, например:

- потеря сетевого соединения;

- отказ клиентского приложения;

- отказ компьютера и т.д.

Во время отката происходит отмена всех модификаций, выполненных в данной транзакции, и освобождение всех ресурсов, использовавшихся этой транзакцией.

Программируемые откаты

Точку транзакции, в которой возможен откат транзакции можно указать с помощью оператора ROLLBACK

Данный оператор прекратит выполнение всех нижеследующих операторов транзакции и выполнить отмену уже выполненных операторов.

Соответственно, если будет запущен откат в середине транзакции, то остальная часть этой транзакции игнорируется.

Фиксация транзакции

begin tran

delete from tmp where empl\_num = 101

select \* from tmp //сотрудник удален

commit tran

select \* from tmp // удаление зафиксировалось

Откат транзакции

use library

begin tran

delete from Teachers where FirstName = 'Алекс'

select \* from Teachers --преподаватель удален

rollback tran

select \* from Teachers -- удаление отменено

select \* from tmp //удаление отменено

Выдача кретдита: покупатель может сделать несколько покупок в кредит, но так чтобы их общая стоимость не превышала предела допустимого для него кредита.

Создать 2 таблицы:

Покупатель: ФИО, предел кредита, текущий кредит, с ограничением check

Покупки: название, цена

create table customer(id int identity, name varchar(20), credit\_limit money, credit money)

alter table customer add constraint ch\_credit check (credit\_limit > credit)

create table package(customer\_id int, name varchar(20), price money)

insert customer values('Mary',1000,0)

ХП: добавляет покупку, увеличивает текущий кредит, в случае нарушения ограничения отменяет все действия и выводит сообщение.

create proc spBuy (@customer\_id int, @name as varchar(20),@price as money)

as

begin

begin tran tr\_buy

insert into package values(@customer\_id, @name,@price)

begin try

update customer set credit = credit + @price

end try

begin catch

print 'Превышение кредита'

rollback tran tr\_buy

return -1

end catch

print 'Покупка выполнена'

commit tran tr\_buy

return 0

end

Проверка

exec spBuy 1, 'Notebook',800 //покупка выполнена

select \* from customer //кредит 800

select \* from package //1 запись - Notebook

exec spBuy 1, 'TV',400//превышение кредита (откат транзакции)

select \* from customer//кредит 800

select \* from package//1 запись – Notebook

(вставка в package и изменение поля credit customer отменены)

exec spBuy 1, 'Phone',100//покупка выполнена

select \* from customer //кредит 900

select \* from package //2 записи – Notebook, Phone

Точки сохранения

Далее указать на то, что есть возможность избежать отката всей транзакции, применя для этого специальный механизм – точку сохранения.

Точка сохранения дает возможность производить откат только до определенной точки в транзакции, а не до самого начала транзакции. Изменени, которые были произмедены до использования точки сохранения, будут зафиксированы и не подвержены откату; все операторы, которые были выполнены после точки сохранения будут отменены. После этого будут выполнены операторы, следующие за оператором ROLLBACK.

Отметить, что при откате транзакции до точки сохранения SQL Server не освобождает блокированные ресурсы. Они будут освобождены после фиксирования транзакции или после отката всей транзакции.

Возможность указания точки сохранения в транзакции, можно осуществить с помощью оператора:

SAVE TRAN[SACTION] {имя\_точки\_сохранения | @переменная\_с\_именем\_точки\_сохранения}

Синтаксис:

ROLLBACK TRAN имя\_точки\_сохранения

Напомнить студентам не забывать включать оператор COMMIT или ROLLBACK после первого оператора ROLLBACK, чтобы завершить всю транзакцию.

Пример.

use library

go

Begin Transaction

insert into Teachers (FirstName,LastName,Id\_Dep) values ('Teachers13','T13',1)

insert into Teachers (FirstName,LastName,Id\_Dep) values ('Teachers14','T14',4)

Save transaction sv1

insert into Teachers (FirstName,LastName,Id\_Dep) values ('Teachers15','T15',6)

insert into Teachers (FirstName,LastName,Id\_Dep) values ('Teachers16','T16',9)

if @@ERROR<>0

begin

print 'Ошибка!!!'

Rollback Transaction sv1

Commit Transaction

end

else

Commit Transaction

go

1. **Блокировки. Параллельная организация работы.**

Следующим этапом перейти к рассмотрение организации параллельной работы с БД. Параллельная организация работы – можно назвать процесс предоставление возможности нескольким пользователям одновременно использовать объекты БД. Основою же параллельной работы является блокировки, поэтому основную часть материала посвятить рассмотрению разновидностей блокировок, а также ситуаций их применения.

Подчеркнуть то , что блокировки предоставляют инструмент, позволяющий предотвратить выполнение с объектом данных операций, которые конфликтуют с операциями, ранее выполненными с этим объектом.

Перечислить возможные нарушения в работе, предотвращаемые с помощью блокировок:

- чтение незафиксированных данных

- неповторяемое чтение

- фантомы

- потерянные обновления.

Далее рассмотреть каждый из перечисленных отдельно.

1. **Чтение незафиксированных данных.**

Происходит в том случае, когда происходит чтение данных, которые, в свою очередь. модифицируются в другой транзакции, при этом другая транзакция не завершена.

**ТР1**: изменение данных

**ТР2**: чтение

**ТР1**: откат

В результате в ТР2 будут прочитаны данные, которых фактически нет в БД.

По-умолчанию в СУБД SQL Server уровень транзакции read committed, и именно благодаря ему данная проблема и решается по-умолчанию.

Пример:

Для имитации 2 пользователей открываем 2 SQL Query.

ТР1

use test

go

begin tran

update customer set credit\_limit = 1500 where name = 'Mary'

*select \* from tmp*

WAITFOR delay '0:0:10'

rollback tran

ТР2

*begin tran*

*--set transaction isolation level read uncommitted (1) - нарушение*

*--set transaction isolation level read committed (1) - OK*

*select \* from tmp*

*commit tran*

(1)Выполняем ТР1.

Выполняем ТР2: получаем age = 1500. Через 5 сек. выполняем еще раз, получаем age = 1000.

(2) select в ТР2 не выполнится, пока не завершится ТР1.

1. **Неповторяемое чтение**

Происходит в том случае, когда строка считывается несколько раз, а в то же время происходит изменение данных, которые считываются в отдельной транзакции. Для исправления уровень изоляции repeatable read или serializable

**ТР1**: читает строку данных

**ТР2**: изменяет данные

**ТР1**: читает ту же строку, но получает другие данные

Пример:

Пример с проверкой счета, а уж потом выполнение работы со счетом

Повторное чтение строки данных

ТР1:

use test

go

begin tran

select \* from customer

WAITFOR delay '0:0:5'

select \* from customer

commit tran

ТР2:

use test

go

update customer set credit\_limit = 2000 where name = 'Mary'

(1)Запускаем ТР1, и пока она ждет 10 сек, выполняем ТР2.

В результате 1-ая выборка данных возвращает age = 1500, 2-ая age = 2000.

(2) ТР2 не начнется, пока не закончится ТР1, в результате обе выборк вернут один и тот же результат.

1. **Фантомы**

Строки, появление и удаление которых приводит к непредсказуемым результатам при параллельной работе с данными. При описании данного процесса желательно привести доступные примеры, например одновременная установка минимальной зарплаты и начисление зарплаты посредством запросов.

Появление строк, которые противоречат применяемым операторам update, delete

**ТР1**: обновление (удаление) по условию where

**ТР2**: вставка данных, которые противоречат этому условию

**ТР1**: чтение данных: обнаруживаются данные, которые противоречат условию where

Для исправления используется уровень изоляции serializable

Пример:

ТР1: Увеличиваем квоту и добавляем ограничение

use test

go

begin tran

--set transaction isolation level read committed (1) - нарушение

--set transaction isolation level serializable (2) - OK

update customer set credit\_limit = 2000 where name = 'Mary'

waitfor delay '0:0:10'

alter table customer add constraint ch\_quota check ( credit\_limit>= 2000)

commit tran

ТР2: Вставляем данные, которые протворечат заданному ограничению

use test

go

update customer set credit\_limit = 2000 where name = 'Mary'

insert into customer (name,credit\_limit,credit) values('Juli',1000,500)

1. Запускаем ТР1, и пока она ждет 10 сек, выполняем ТР2.

Задание ограничения приводит к ошибке, т.к. в ТР2 вставились данные, которые противоречат этому ограничению.

(2) Cначала завершится ТР1, затем выполнится ТР2, при выполнении ТР2 возникнет ошибка, т.к. ограничение уже задано.

(alter table tmp drop constraint ch\_quota – удаление ограничения)

1. **Потерянное обновление**

Происходит в том случае, если при одном обновлении данные были записаны в базу данных и при этом данные результаты перекрываются с другой транзакцией. Такая ситуация, например, возможна, если в двух транзакциях производится изменение в одной строке. После чего в одной транзакции в строку записываются новые данные и то же происходит и во второй транзакции.

При описании данного процесса желательно привести доступные примеры, например одновременная работа двух операторов с банковскими реквизитами одного и того же клиента.

Пример: форма редактирования запоминает значения полей записи в переменные, при обновлении любого поля, все сохраненные в переменных значения записываются в БД. Эксперт по кредитам и менеджер одновременно открыли эту форму для одной и той же записи. Эксперт увеличил кредит и записал в БД, затем менеджер изменил адрес и записал в БД. В результате значение кредита останется не измененным, т.к. записалось его старое значение, которое было сохранено формой менеджера.

Потерянные обновления:

ТР1: считывает строку и запоминает все ее поля в переменные

ТР2: считывает строку и запоминает все ее поля в переменные

ТР1: обновляет одно поле строки

ТР2: обновляет другое поле строки

Обновления, сделанные в ТР1, теряются.

Далее обсудить со студентами способы определения уровней изоляции, требуются для установки в ситуациях обсуждаемых выше.

Предоставляется четыре уровня изоляции, которые уже упоминались:

Read committed

Read uncommitted

Repeatable read

Serializable

READ UNCOMITTED - позволяет считывать данные, которые были изменены, но не были фиксированы. Может использоваться, например, если выполняются длительные операции (подготовка отчетов) для которых не требуется абсолютно точных данных

READ COMMITTED – не позволяет считывать данные, которые были изменены, но не были фиксированы. Используется по умолчанию.

REPETABLE READ – то же, что READ COMMITTED + другая транзакция не может изменять данные, считанные данной транзакцией до ее завершения.

SERIALIZABLE - то же, что REPETABLE READ + другие транзакции не могут вставить/изменять/удалять данные, которые попадают в диапазон записей, удовлетворяющих условия where, заданному в любом операторе транзакции.

Задание уровня изоляции транзакции:

SET TRANSACTION ISOLATION LEVEL

<READ COMMITTED | READ UNCOMITTED | REPETABLE READ | SERIALIZABLE>

Подчеркнуть, что изменение уровня будет иметь значение только для текущей транзакции и не действует отрицательно на работу пользователей работающих параллельно.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень изоляции  транзакции | Нарушения в работе, которые исключаются при использовании заданного уровня изоляции транзакции | | | |
| чтение незафиксированных данных | неповторяемое чтение | фантомы | потерянные обновления |
| READ UNCOMITTED |  |  |  |  |
| READ COMMITTED | + |  |  |  |
| REPETABLE READ | + | + |  |  |
| SERIALIZABLE | + | + | + |  |

Чем выше степень изоляции, тем выше вероятность, что другие пользователи не смогут получить доступ к нужным им объектам БД.