

РАБОТА С ДАННЫМИ НА БАЗЕ MS SQL SERVER И T-SQL

Author: Pavel Belyakov

poulbelyakov@gmail.com

Содержание

1. Обзор MS SQL Server 2008
2. Манипулирование данными и объектами
3. Запрос и фильтрация данных
4. Группирование и агрегирование
5. Присоединение таблиц и вложенные запросы
6. Изменение данных в таблицах
7. Объекты для работы с данными

ОБЗОР MS SQL SERVER 2008

Обзор MS SQL Server 2008

MS SQL Server 2008 – система управления базами данных, разработанная Microsoft, с возможностью сетевого и многопользовательского доступа.

Первая версия появилась в 1989 году.

На текущий момент, наиболее распространенной является версия 2008 R2.

Готовится к выходу версия MS SQL 2012.

Обзор MS SQL Server 2008

Ключевые сервисы MS SQL Server 2008:

Сервис	Описание
SQL Server Database Engine	Ключевой компонент, обеспечивающий хранение и обработку данных
Analysis Services	Компоненты для работы с аналитической обработки данных
Reporting Services	Компоненты для работы с отчетами
Integration Services	Компоненты для обеспечения интеграции данных

Database Engine предоставляет ряд дополнительных сервисов:

- Полнотекстовый поиск

- Сервисный брокер

- Репликация данных

- Сервисы уведомлений

Обзор MS SQL Server 2008

SQL Server Database Engine – ключевой сервис, отвечающий за хранение, обработку и безопасность данных, а также выполняющий ряд других функций.

Основные компоненты:

Protocols – способы реализации внешнего взаимодействия с сервером

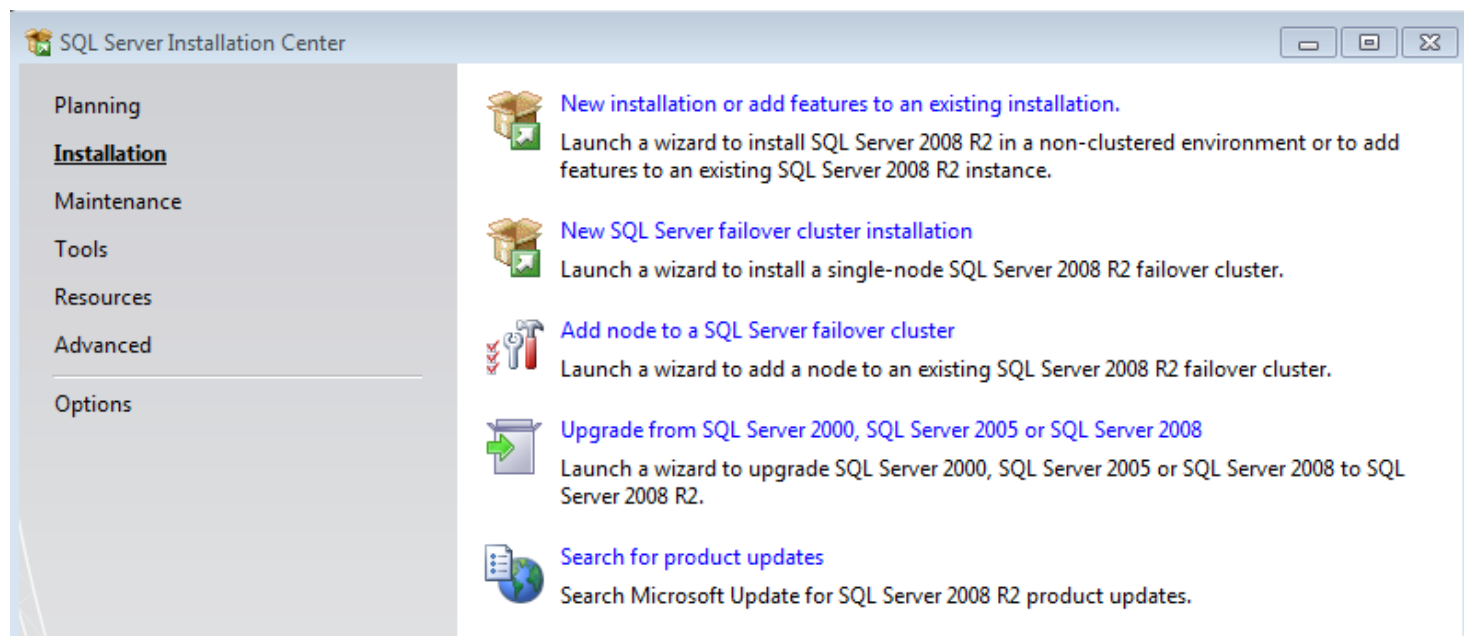
Storage Engine – ядро SQL-сервера, отвечает за обработку, хранение и безопасность данных

Relational Engine – интерфейс взаимодействия с Storage Engine, состоит из набора сервисов, предназначенных для взаимодействия с БД и другими сервисами

SQLOS – операционная система со своим API, управляющая работой всех компонентов SQL-сервера

Установка MS SQL Server 2008

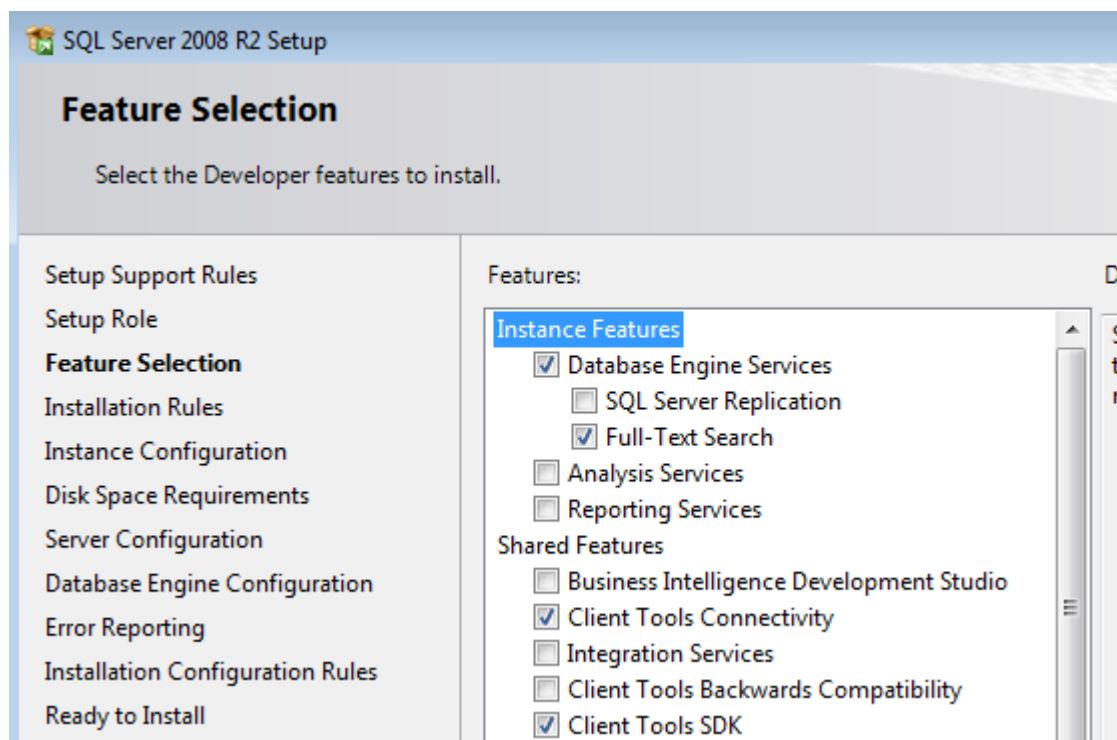
Запустив setup.exe, мы увидим окно установщика:



Нас интересует пункт «**New installation or add features...**», в закладке Installation.

Установка MS SQL Server 2008

Пройдя проверку и предустановку файлов, инсталлятор предложит вам выбрать компоненты к установке:



Обзор MS SQL Server 2008

Установка MS SQL Server 2008

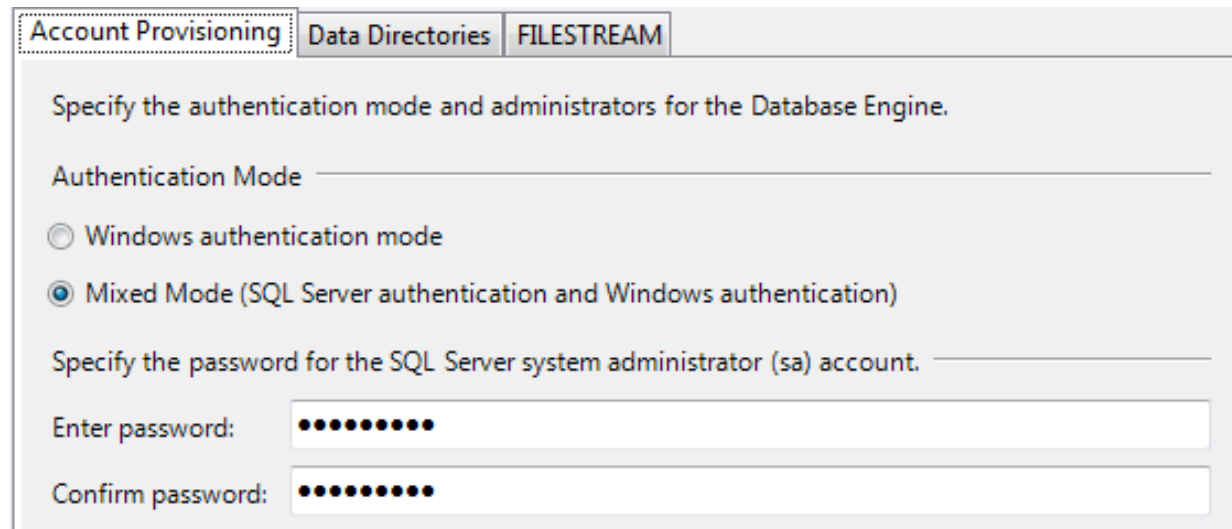
Установщик предложит задать имя для данного экземпляра, либо использовать имя по умолчанию:

The screenshot shows the 'Instance Configuration' step of the MS SQL Server 2008 setup. It includes radio buttons for 'Default instance' (selected) and 'Named instance'. Below these are text boxes for 'Instance ID' (MSSQLSERVER) and 'Instance root directory' (C:\Program Files\Microsoft SQL Server\). The 'SQL Server directory' is also shown as C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER. At the bottom, there is a table titled 'Installed instances' showing the existing SQL Express instance.

Instance Name	Instance ID	Features	Edition	Version
SQLEXPRESS	MSSQL10.SQLEXP...	SQLEngine, SQLEn...	Express	10.3.5500.0

Установка MS SQL Server 2008

По умолчанию на MS SQL сервере включена **только аутентификация на основании учетной записи Windows**. Выбирая Mixed Mode, мы включаем **SQL Server аутентификацию** – на основе логина и пароля:



The screenshot shows the 'Account Provisioning' tab of the SQL Server Enterprise Setup wizard. The 'Authentication Mode' section has two radio buttons: 'Windows authentication mode' (unselected) and 'Mixed Mode (SQL Server authentication and Windows authentication)' (selected). Below this, there is a prompt to 'Specify the password for the SQL Server system administrator (sa) account.' followed by two password input fields labeled 'Enter password:' and 'Confirm password:', both containing masked characters (dots).

Хотя SQL server аутентификация менее безопасна Windows аутентификации, в мире WEB, первая применяется гораздо чаще.

СХЕМЫ

Для идентификации объектов на сервере используются **схемы** – пространства имен, к которым относятся объекты.

При установке сервера, так же создается несколько пространств имен, наиболее примечательные из них:

dbo.*

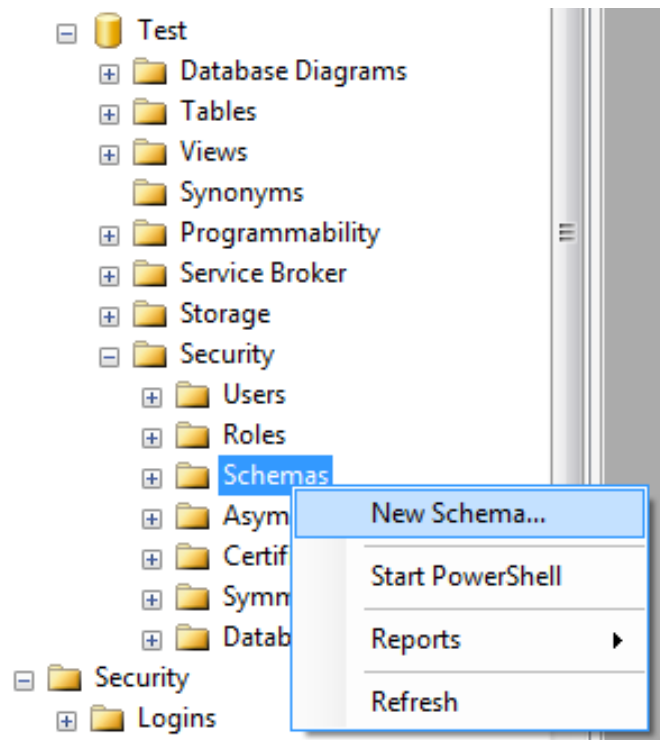
Такой префикс имеют объекты, созданные **пользователем-владельцем** БД, включенном в группу **db_owner**, если пользователь не принадлежит к этой группе и попытается создать объект, то префикс объекта будет равным его имени.

sys.*

Включает в себя набор объектов, содержащих информацию системного уровня: список всех схем сервера, список всех баз, пользователей и т.д.

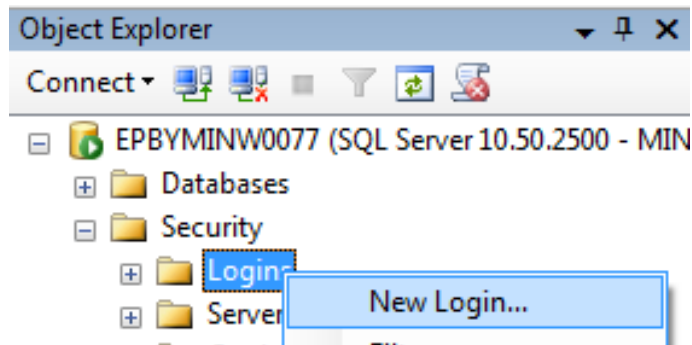
СХЕМЫ

Хорошей практикой считается создание своей собственной схемы для объектов, которые будут использоваться из-вне сервера:



LOGINS

Для работы с базой, обычно создают специализированного пользователя, имеющего права доступа только данной базе:

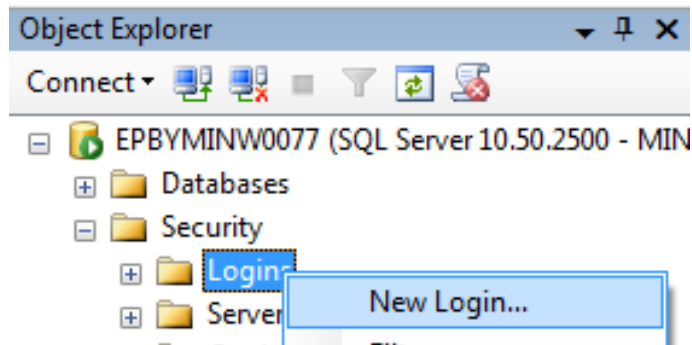


При создании логина надо будет указать:

- Способ аутентификации (а так же задать нужные параметры)
- Указать к каким БД этот логин имеет доступ и его роль для конкретной БД

LOGINS

Для работы с базой, обычно создают специализированного пользователя, имеющего права доступа только данной базе:

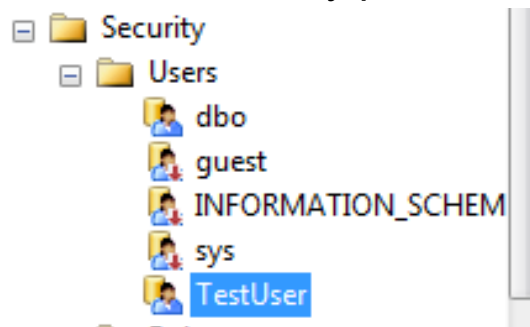


При создании логина надо будет указать:

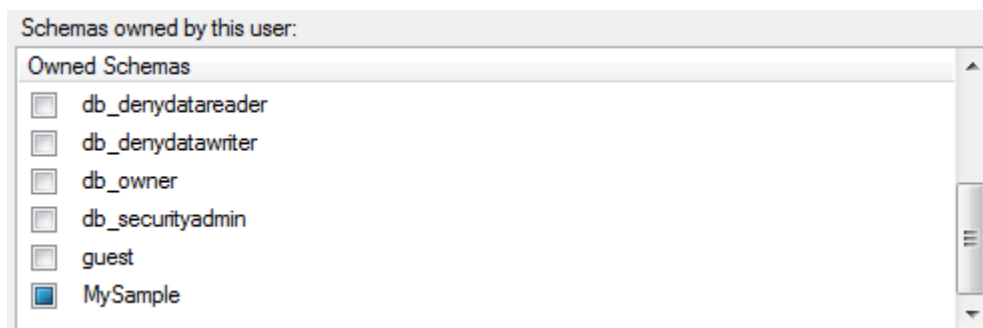
- Способ аутентификации (а так же задать нужные параметры)
- Указать к каким БД этот логин имеет доступ и его роль для конкретной БД

USERS

После создания логина, в нашей БД создался пользователь с публичным доступом (при создании дали ему public доступ):



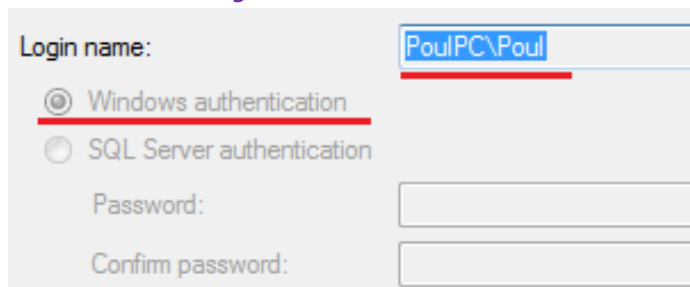
Дадим этому пользователю доступ к нашей схеме (в его свойствах):



Пользователи

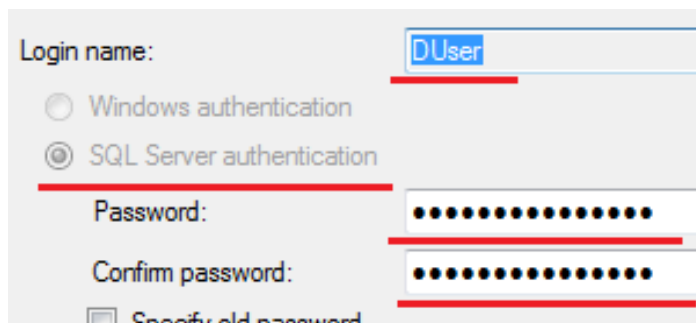
Прежде чем вы сможете работать с сервером, необходимо идентифицировать себя. В MS SQL Server доступно 2 способа аутентификации:

На основе учетной записи Windows:



The screenshot shows the 'Server Properties' dialog box for a SQL Server instance. The 'Login name' field contains 'PoulPC\Poul'. The 'Windows authentication' radio button is selected and highlighted with a red underline. The 'SQL Server authentication' radio button is unselected. The 'Password' and 'Confirm password' fields are empty.

На основе введенных имени пользователя и пароле:

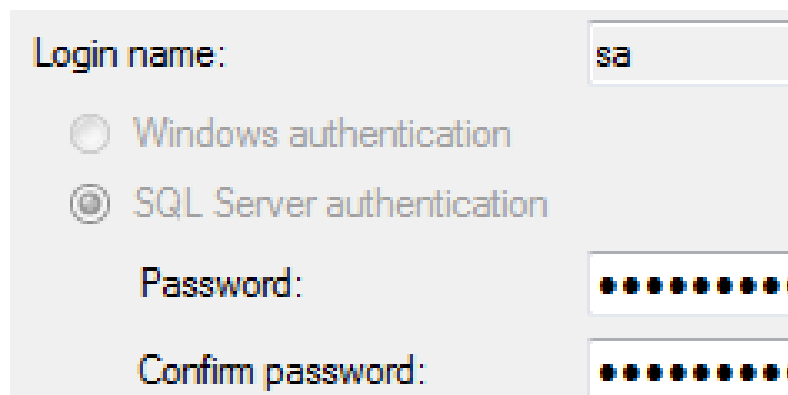


The screenshot shows the 'Server Properties' dialog box for a SQL Server instance. The 'Login name' field contains 'DUser'. The 'SQL Server authentication' radio button is selected and highlighted with a red underline. The 'Windows authentication' radio button is unselected. The 'Password' and 'Confirm password' fields are filled with dots, indicating masked passwords. A 'Specify old password' checkbox is visible at the bottom.

SQL Authentication

По умолчанию доступен специальный логин – sa.

Эта учетная запись является резервной, на тот случай если никакие другие пользователи не могут войти. По умолчанию пароль доступа к этой учетной записи – 12345.



Login name: sa

☐ Windows authentication

☒ SQL Server authentication

Password: [masked]

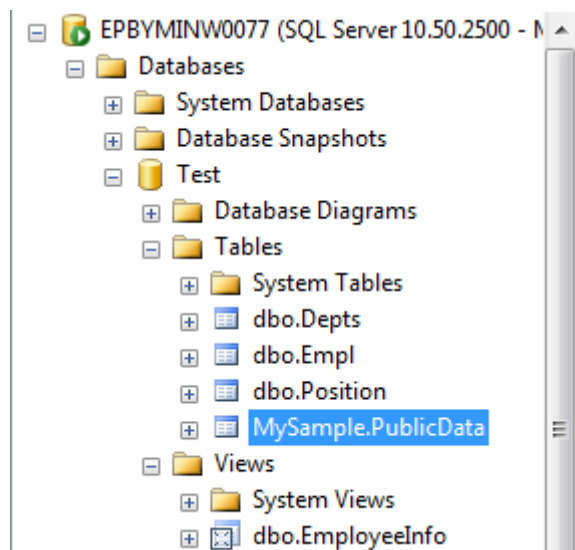
Confirm password: [masked]

Делиться этим логином с кем либо не рекомендуется.

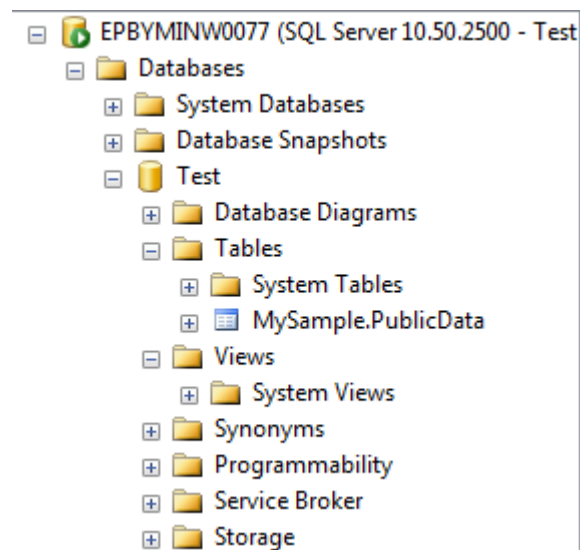
Итог всех манипуляций

В итоге всех этих действий, мы спрятали от обычных пользователей лишние данные:

Admin



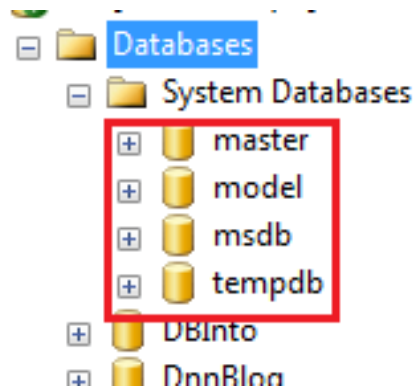
TestUser



МАНИПУЛИРОВАНИЕ ДАННЫМИ И ОБЪЕКТАМИ

Манипулирование данными и объектами

На каждом экземпляре сервера всегда в наличии есть 4 БД:



master – содержит общие настройки всего сервера;

model - является шаблоном для ваших БД;

msdb – используется для планирования задач, работы служб;

tempdb – база для хранения временной информации во время работы сервера.

Манипулирование данными и объектами

model:

Эта БД содержит в себе множество полезных объектов, обратившись к которым можно узнать структуру вашей базы и многое другое:

```
Select * from INFORMATION_SCHEMA.TABLES
```

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
1	DBInto	dbo	Countries	BASE TABLE
2	DBInto	dbo	Cities	BASE TABLE
3	DBInto	dbo	Departments	BASE TABLE
4	DBInto	dbo	Products	BASE TABLE
5	DBInto	dbo	OldProducts	BASE TABLE
6	DBInto	dbo	DepartmentsAndCities	VIEW
7	DBInto	dbo	sysdiagrams	BASE TABLE

Манипулирование данными и объектами

SQL-запросы

Transact SQL или **T-SQL** является основным языком программирования в MS SQL Server 2008.

T-SQL – используется для опроса данных на SQL-сервере, в виде **SQL-запроса** к данным, находящимся на сервере.

SQL-запросы – набор текстовых команд, передаваемых на сервер из приложения, обрабатывая который, сервер возвращает некоторый результат.

Прежде чем запрос будет исполнен, он обрабатывается **интерпретатором**.

Манипулирование данными и объектами

Категории запросов T-SQL

Data Manipulation Language (DML) - запросы на изменение/добавление/удаление данных:

```
INSERT INTO (Table) VALUES (list of values)
```

Data Query Language (DQL) – относиться **DML**, но такие запросы не позволяют изменять данные:

```
SELECT (list of columns) FROM (Table)
```

Data Definition Language (DDL) – язык описания и изменения схемы хранения данных:

```
CREATE (object) (name)
```

Transaction Control Language (TCL) – язык для работы с транзакциями:

```
COMMIT TRANSACTION (name)
```

Data Control Language (TCL) – работа с доступом пользователей к объектам БД:

```
GRANT CONTROL (object) TO (user)
```

Манипулирование данными и объектами

При работе с SQL-сервером вы будете оперировать одним или несколькими объектами:

Объект	Описание
Table	Вся информация в SQL-сервере хранится в виде таблиц
View	Можно представить как «виртуальную» таблицу, т.е. некоторую выборку из реальных таблиц при помощи заранее определенных запросов
Indexes	Ускоряют операции доступа к данным, генерируются на основе одного или нескольких полей
Triggers	Позволяют выполнять SQL-код при выполнении операций добавления/удаления/обновления данных в определенные таблицы
Stored procedures	Выполнение заранее откомпилированного SQL-кода на сервере, возможна передача входных и выходных параметров, возвращение набора данных
Constraints	Препятствуют вставке противоречивых данных
Rules	Задают значения, которые можно вставлять в базу

Работа с объектами

Помимо работы с данными, можно так же работать с объектами базы, используя запросы CREATE, DROP, ALTER:

Создание объектов:

```
CREATE TABLE ...  
CREATE PROCEDURE...
```

Обновление объектов:

```
ALTER TABLE ...  
ALTER PROCEDURE...
```

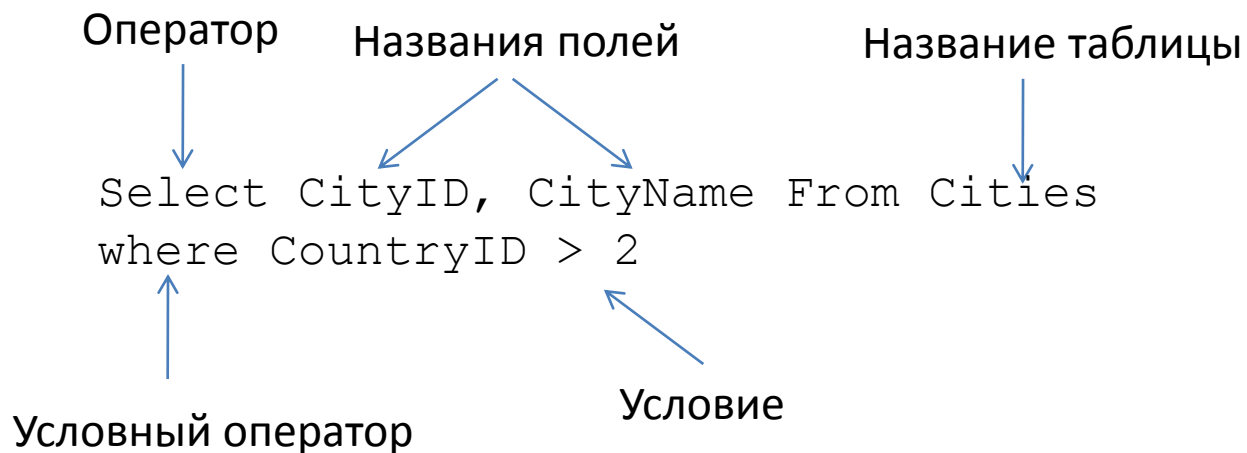
Удаление объектов:

```
DROP TABLE ...  
DROP PROCEDURE...
```

ЗАПРОС И ФИЛЬТРАЦИЯ ДАННЫХ

Запрос и фильтрация данных

Все SQL-запросы имеют похожий вид



Запрос и фильтрация данных

Запрос **SELECT** считывает данные и возвращает их:

Получить все список всех городов Беларуси:

```
Select CityID, CityName From Cities  
where CountryID = 1
```

	CityID	CityName
1	1	Minsk
2	2	Brest
3	3	Borisov
4	4	Grodno

Запрос и фильтрация данных

Почти всегда запрос **SELECT** возвращает **набор данных в виде отношения**, и это является важной особенностью о которой нельзя забывать, так как над результирующим набором можно совершать такие же действия как и над обычной таблицей:

Предыдущий пример вернул вот такой набор:

	CityID	CityName
1	1	Minsk
2	2	Brest
3	3	Borisov
4	4	Grodno

Хотя он и отличается от нашей первоначальной таблицы Cities, все таки это тоже набор но состоящий из 2-х столбцов и 4-х строк.

Запрос и фильтрация данных

Вместо указания конкретных имен полей, которые вы ходите выбрать, можно указать сокращенную запись:

```
Select * From Countries
```

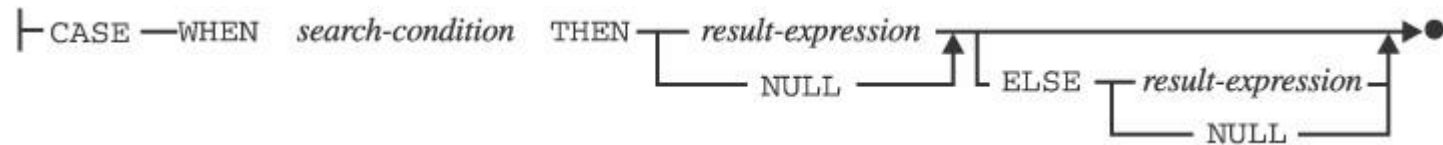
Вот результат:

	CountryID	CountryName
1	1	Belarus
2	2	Russia
3	3	Ukraine

Запрос и фильтрация данных

Использование CASE

Оператор CASE позволяет расширять SELECT, определенным условием наподобие IF...THEN...ELSE.



Запрос и фильтрация данных

Пример CASE

```
Select CityID, CityName,  
case CountryID  
  when 1 then 'Belarus'  
  when 2 then 'Russia'  
  when 3 then 'Ukraine'  
  else 'Unknown'  
End From Cities
```

В результате получим:

	CityID	CityName	(No column name)
1	1	Minsk	Belarus
2	2	Brest	Belarus
3	3	Borisov	Belarus
4	4	Grodno	Belarus
5	5	Moscow	Russia
6	6	St. Piterburg	Russia
7	7	Novosibirsk	Russia
8	8	Volgograd	Russia
-	-

Запрос и фильтрация данных

Использование DISTINCT

В некоторых случаях у вас может возникнуть дублирование в результирующем наборе, на пример, если модифицировать предыдущий оператор:

```
Select case CountryID
  when 1 then 'Belarus'
  when 2 then 'Russia'
  when 3 then 'Ukraine'
  else 'Unknown'
End From Cities
```

В результате увидим:

	(No column name)
1	Belarus
2	Belarus
3	Belarus
4	Belarus
5	Russia
6	Russia
7	Russia

Запрос и фильтрация данных

Для того что бы убрать дублирование, нужно использовать оператор **DISTINCT**:

```
Select DISTINCT  
case CountryID  
when 1 then 'Belarus'  
...
```

И видим что дублирования больше нет:

	(No column name)
1	Belarus
2	Russia
3	Ukraine

Запрос и фильтрация данных

Условие выборки **WHERE**

Таблицы с данными могут содержать очень много записей, миллионы записей. Выбирать все записи обычным **SELECT** требуется очень редко.

В большинстве случаев, Вы будете выбирать данные следуя некоторому условию. Реализовать такую выборку позволяет оператор **WHERE**:

Выбрать все департаменты города Минска (ID = 1):

```
Select * from Departments where CityID = 1
```

Запрос и фильтрация данных

Поддерживаемые типы данных:

Numeric

Exact:

*int – bigint – bit – decimal –
smallint – money – smallmoney –
tinyint*

Approximate:

float – real

Date and time

*date – time – datetime – datetime2
– datetimeoffset – smalldatetime*

Binary strings

binary – varbinary – image

Strings

Character:

char – varchar – text

Unicode character:

nchar – nvarchar – ntext

Other

*cursor – timestamp – table
...and more*

Запрос и фильтрация данных

Поддерживаемые операции:

Арифметические: +, -, *, /, %

Присваивания: =

Сравнения: =, >, <, <> (not equal to), ! (not), >=, <=

Логические: AND, OR, NOT

Объединение строк: +

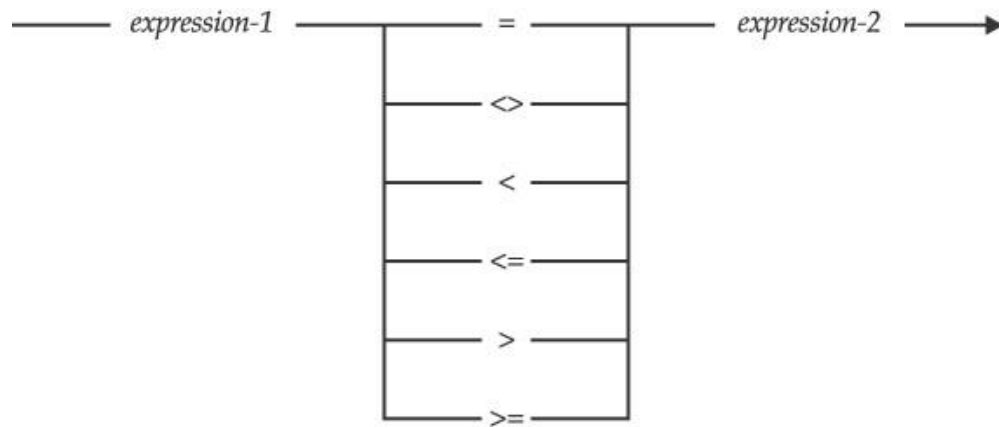
Запрос и фильтрация данных

SQL сервер позволяет проверять различные условия:

- Сравнение на соответствие, больше, меньше;
- Вхождение в диапазон;
- Вхождение в определенный набор значений;
- Соответствие шаблону;
- Проверка на null.

Запрос и фильтрация данных

Сравнение на соответствие, больше, меньше:

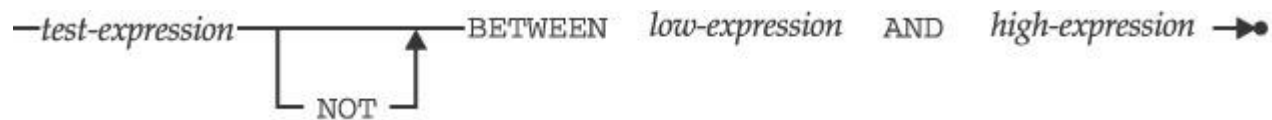


Все департаменты, открытые до 1 января 2011 года:

```
Select * from Departments  
where DateOfCreation < '2011-01-01'
```

Запрос и фильтрация данных

Вхождение в диапазон (**BETWEEN**):

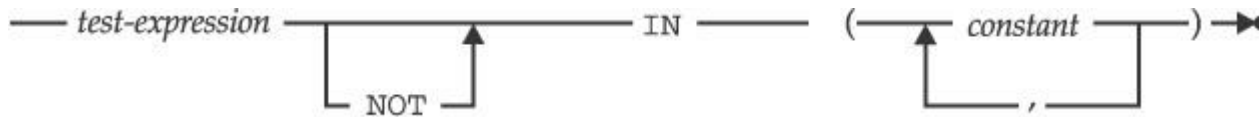


Все департаменты, открытые с января по декабрь 2010:

```
Select * from Departments
where DateOfCreation between '2010-01-01' and '2011-12-01'
```


Запрос и фильтрация данных

Вхождение в определенный набор (**IN**)



Найти все департаменты в Минске, Москве и Киеве:

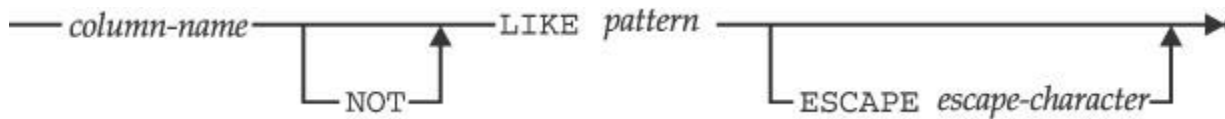
```
Select * from Departments  
where CityID in (1,5,9)
```

Найти все департаменты вне Минска:

```
Select * from Departments  
where CityID not in (1)
```

Запрос и фильтрация данных

Соответствие шаблону (**LIKE**)



Шаблонные символы:

% - позволяет заменить любое количество символов;

_ - заменяет один символ;

\$ - задает escape-последовательность.

Запрос и фильтрация данных

LIKE и %

```
Select * from Departments  
where DepartmentName like '%Store'
```

Такой запрос вернет все строки, поле с именем департамента которых оканчивается на “Store”;

	DepartmentID	CityID	DepartmentName	DateOfCreation
1	2	2	Product Store	2010-01-01
2	3	3	Store	2010-01-01
3	6	6	Store	2010-01-01
4	10	10	Store	2010-02-01
5	14	11	Store	2011-05-01
6	15	10	Product Store	2011-05-01
7	18	8	Store	2011-05-01
8	22	7	Store	2011-10-01

Запрос и фильтрация данных

LIKE и _

```
Select * from Departments  
where DepartmentName like ' _ Store'
```

Такой запрос вернет все строки, поле с именем департамента которых оканчивается на “Store” и имеет один символ в начале;

	DepartmentID	CityID	DepartmentName	DateOfCreation
1	6	6	A Store	2010-01-01
2	10	10	B Store	2010-02-01
3	14	11	% Store	2011-05-01

Запрос и фильтрация данных

LIKE и \$

```
Select * from Departments  
where DepartmentName like '$% St%' escape '$'
```

Такой запрос вернет все строки, поле с именем департамента которых оканчивается на “Store” и в начале имеет символ ‘%’;

	DepartmentID	CityID	DepartmentName	DateOfCreation
1	14	11	% Store	2011-05-01

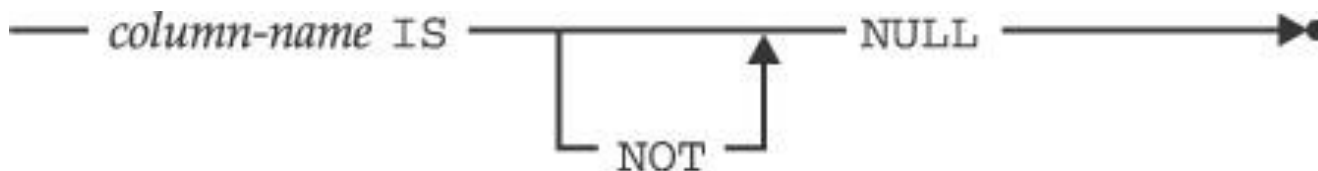
Запрос и фильтрация данных

NULL-значения:

- **NULL** являются **UNKNOWN** значениями (используется в операциях, как индикатор ошибки)
- **NULL** не равно **0** или **пустой строке**
- Сравнение **NULL** с любым другим значением (а так же с **NULL**) вернет **UNKNOWN**
- **NULL** не равно **0** или **пустой строке**
- Получить записи, содержащие **NULL** можно при помощи оператора **IS [NOT] NULL**

Запрос и фильтрация данных

Проверка на NULL



Найти все департаменты, у которых не задана дата создания:

```
Select * from Departments  
where DateOfCreation IS NULL
```

	DepartmentID	CityID	DepartmentName	DateOfCreation
1	21	5	Office	NULL
2	22	7	Store	NULL

Сложные условия выборки (AND, OR, NOT)

Условия выборки можно комбинировать используя операторы **AND**, **OR**, **NOT**, создавая более сложные операторы выборки:

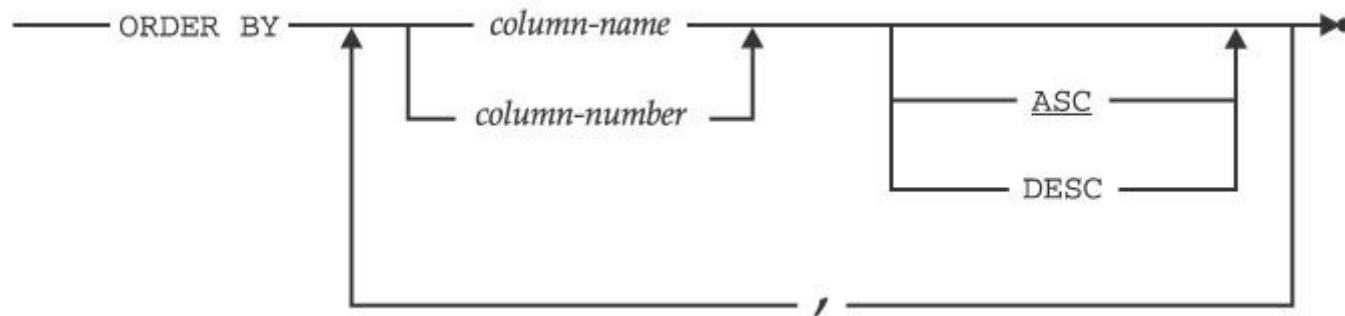
Выбрать все департаменты находящиеся не в Минске, не имеющие даты создания или созданные до 01.01.2011 и имеющие тип "Office":

```
Select * from Departments where  
  (CityID <> 1)  
and  
  ((DateOfCreation is Null) or (DateOfCreation < '2011-01-01'))  
and  
  (DepartmentName = 'Office')
```


Запрос и фильтрация данных

Сортировка ORDER BY:

Результирующий набор можно отсортировать перед дальнейшим использованием. Осуществляется это при помощи оператора ORDER BY:



Запрос и фильтрация данных

Применение ORDER BY:

По умолчанию данные сортируются в порядке возрастания:

```
Select DepartmentName, DateOfCreation from Departments  
order by DepartmentName
```

DepartmentName	DateOfCreation
% Store	2011-05-01
A Store	2010-01-01
B Store	2010-02-01
HR	2010-02-01
HR	2010-01-01

Что бы отсортировать в обратном порядке следует использовать такой синтаксис:

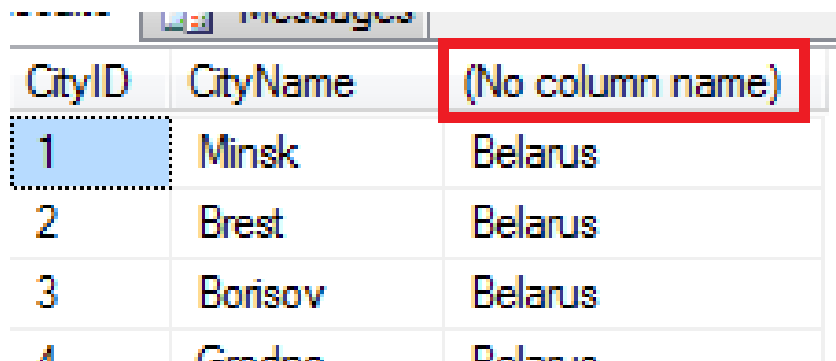
```
Select DepartmentName, DateOfCreation from Departments  
order by DepartmentName desc
```

Запрос и фильтрация данных

Применение ORDER BY:

В некоторых запросах результирующие наборы могут иметь колонки без названий, рассмотрим ранний пример:

```
Select CityID, CityName,  
case CountryID  
when 1 then 'Belarus'  
...
```



CityID	CityName	(No column name)
1	Minsk	Belarus
2	Brest	Belarus
3	Borisov	Belarus
4	Grodno	Belarus

Запрос и фильтрация данных

Применение ORDER BY:

Что бы отсортировать по колонке, не имеющей имени можно указать её номер:

```
Select CityID, CityName,  
case CountryID  
... end  
From Cities order by 3 desc
```

Результат этого запроса:

CityID	CityName	(No column name)
9	Kiev	Ukraine
10	Lvov	Ukraine
11	Odessa	Ukraine
12	Sudak	Ukraine
5	Moscow	Russia

Запрос и фильтрация данных

Применение ORDER BY:

Сортировку можно выполнять по нескольким полям, указав их имена через запятую:

```
Select CityID, CityName, case CountryID  
End From Cities order by 3, CityName
```

В результате увидим:

CityID	CityName	(No column name)
3	Borisov	Belarus
2	Brest	Belarus
4	Grodno	Belarus
1	Minsk	Belarus
5	Moscow	Russia
7	Novosibirsk	Russia

Важно отметить, что сначала сортировка идет по колонке с названиями стран, а затем записи отдельной страны сортируются по названию городов.

Ограничение результирующего набора

Ограничить размер набора можно при помощи ключевого слова TOP:

```
DECLARE @n as bigint  
SET @n = 3  
SELECT TOP (@n) * FROM Empl
```

В результате получим первые n (в нашем случае 3) строк из таблицы:

	ID	Name	DepartmentID	PositionID
1	1	Mike	1	1
2	2	John	2	2
3	3	Eric	NULL	5

Постраничное считывание записей

Один из способов считать данные в определенном диапазоне строк (наиболее простой):

```
SELECT * FROM  
(SELECT Row_Number() OVER (ORDER BY ID) AS RowIndex, *  
FROM Empl) AS Sub  
WHERE Sub.RowIndex BETWEEN 1 and 3
```

Получаем первую страницу с 3-мя записями:

	RowIndex	ID	Name	DepartmentID	PositionID
1	1	1	Mike	1	1
2	2	2	John	2	2
3	3	3	Eric	NULL	5

ГРУППИРОВАНИЕ И АГРЕГИРОВАНИЕ

Группирование и агрегирование

Функции

Microsoft SQL Server имеет ряд встроенных функций.

Функции из категории “**Aggregate**”, позволяют выполнять операции над наборами данных, но при этом возвращают только **одно сводное значение**:

AVG()

MIN()

MAX()

COUNT()

GROUPING()

CHECKSUM_AGG()

STDEVP()

VAR()

STDEV()

SUM()

VARP()

COUNT_BIG()

Агрегатные функции

```
SELECT  
AVG(pr.Price), SUM(pr.Count) FROM Products AS pr
```

В первой колонке подсчитается средняя цена по колонке Price, во второй количество всех товаров:

81,6666	7
---------	---

Комбинирование результатов нескольких запросов

Можно соединить результирующие наборы нескольких запросов SELECT, для этого запросы должны соответствовать следующим критериям:

- Оба набора должны содержать одинаковое количество полей;
- Типы данных в полях должны соответствовать;
- Можно отсортировать наборы вместе и по отдельности.

Группирование и агрегирование

Комбинирование результатов нескольких запросов

Комбинирование осуществляется при помощи оператора
UNION:

```
SELECT CityID, CityName FROM Cities WHERE CountryID = 2  
UNION  
SELECT CityID, CityName FROM Cities WHERE CountryID >= 2
```

CityID	CityName
5	Moscow
6	St. Piterburg
7	Novosibirsk
8	Volgograd
9	Kiev
10	Lvov
11	Odessa
12	Sudak

Группирование и агрегирование

Комбинирование результатов нескольких запросов:

Комбинировать можно многое....:

```
SELECT CityID, CityName FROM Cities WHERE CountryID = 2  
UNION  
SELECT DepartmentID, DepartmentName FROM Departments  
ORDER BY 2 DESC
```

CityID	CityName
8	Volgograd
3	Store
18	Store
22	Store
6	St. Piterburg
8	RD

Группирование и агрегирование

Результирующие наборы можно группировать по некоторому критерию

Подсчитать количество офисов всех типов:

```
SELECT dp.DepartmentName, COUNT(dp.DepartmentName)
FROM Departments AS dp
GROUP BY dp.DepartmentName
```

Результат:

	DepartmentName	(No column name)
1	% Store	1
2	A Store	1
3	B Store	1
4	HR	4
5	Main Office	1
6	Office	9
7	Product Store	2
8	RD	2
9	Store	3

Группирование и агрегирование

Группирование по нескольким параметрам

Подсчитать количество офисов всех типов в отдельных городах:

```
SELECT dp.DepartmentName, ct.CityName, COUNT(dp.DepartmentName)
FROM Departments as dp
JOIN Cities AS ct ON ct.CityID = dp.CityID
GROUP BY dp.DepartmentName, ct.CityName
```

Результат:

5	HR	Grodno	1
6	Office	Grodno	1
7	Office	Kiev	2
8	B Store	Lvov	1
9	Product Store	Lvov	1
10	HR	Minsk	1

Группирование и агрегирование

Группирование с фильтрацией

Подсчитать количество офисов всех типов в отдельных городах, где более 2-х офисов:

```
SELECT ...  
GROUP BY dp.DepartmentName, ct.CityName  
HAVING COUNT(dp.DepartmentName) > 1
```

Результат:

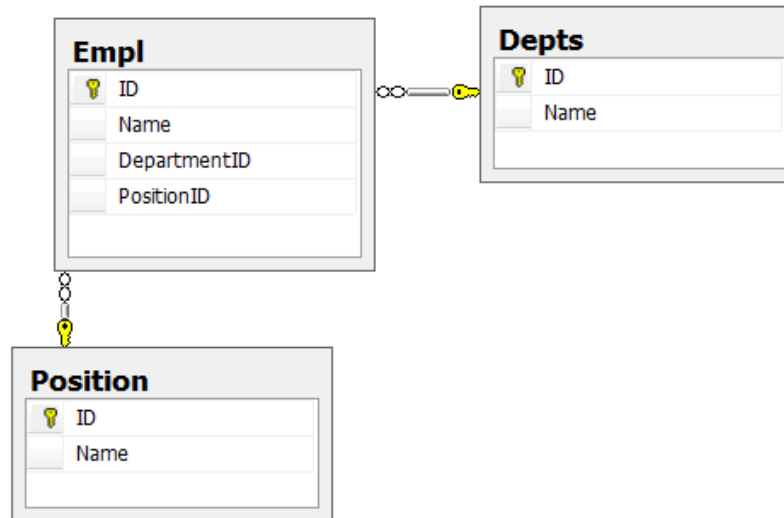
	DepartmentName	CityName	(No column name)
1	Office	Kiev	2
2	Office	Moscow	2

ПРИСОЕДИНЕНИЕ ТАБЛИЦ И ВЛОЖЕННЫЕ ЗАПРОСЫ

Присоединение таблиц и вложенные запросы

JOINING

В большинстве случаев, необходимые данные находятся в разных отношениях:



Механизм позволяющий нам получать данные из нескольких таблиц называется **соединением (joining)**.

Присоединение таблиц и вложенные запросы

Декартово произведение

Попробуем получить имена сотрудников с их ролями:

```
SELECT Empl.Name, Depts.Name from Empl, Depts
```

Такой запрос вернет целых **24** записи, хотя сотрудников у нас только **6**:

	Name	Name
1	Mike	Production
2	John	Production
3	Eric	Production
4	Paul	Production
5	Jessie	Production
6	Jonny	Production
7	Mike	TC
8	John	TC
9	Eric	TC
10	Paul	TC

ID	Name	Department
14	John	Maintenance
15	Eric	Maintenance
16	Paul	Maintenance
17	Jessie	Maintenance
18	Jonny	Maintenance
19	Mike	NULL
20	John	NULL
21	Eric	NULL
22	Paul	NULL
23	Jessie	NULL
24	Jonny	NULL

Мы получили **Декартово произведение** – комбинацию всех записей из левой и правой таблиц. Этот набор надо правильно **соединить** (отфильтровать).

Присоединение таблиц и вложенные запросы

Присоединение сравнением

Простейший способ выполнить подобный запрос – использовать оператор **WHERE**:

```
SELECT Empl.Name, Depts.Name from Empl,Depts  
WHERE Empl.DepartmentID = Depts.ID
```

В результате увидим:

	Name	Name
1	Mike	Production
2	John	TC
3	Paul	Maintenance
4	Jessie	Production

Всего **4** записи, так как у 2-х сотрудников значение DepartmentID равно NULL.

Фильтрация набора вручную – устаревший подход.

JOINS

Вместо фильтрации вручную следует использовать один из встроенных типов присоединений:

OUTER JOIN
FULL, LEFT, RIGHT

INNER JOIN

CROSS JOIN

SELF JOIN

FULL LEFT JOIN (LEFT JOIN)

Возвращает **возможные комбинации** из **правой** таблицы для каждой записи **левой** (в том числе и значение NULL), удовлетворяющие условию выборки:

```
SELECT Empl.Name, Depts.Name from Empl  
LEFT JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

В результирующем наборе видим даже тех сотрудников, у которых не задан департамент:

	Name	Name
1	Mike	Production
2	John	TC
3	Eric	NULL
4	Paul	Maintenance
5	Jessie	Production
6	Jonny	NULL

FULL RIGHT JOIN (RIGHT JOIN)

Возвращает **все возможные комбинации** из **левой** таблицы для каждой записи **правой** (в том числе и значение NULL) , удовлетворяющие условию выборки

```
SELECT Empl.Name, Depts.Name from Empl  
RIGHT JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

У нас есть департамент «Secret», к которому не приписан ни один сотрудник, и мы теперь его видим:

	Name	Name
1	Mike	Production
2	Jessie	Production
3	John	TC
4	Paul	Maintenance
5	NULL	Secret

FULL OUTER JOIN (FULL JOIN)

По сути комбинирует результаты LEFT и RIGHT соединения:

```
SELECT Empl.Name, Depts.Name from Empl  
FULL JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

Видим сотрудников без отделов, а так же отделы без сотрудников:

	Name	Name
1	Mike	Production
2	John	TC
3	Elic	NULL
4	Paul	Maintenance
5	Jessie	Production
6	Jonny	NULL
7	NULL	Secret

Присоединение таблиц и вложенные запросы

INNER JOIN (JOIN)

Выбирает строки, строго удовлетворяющие условию:

```
SELECT Empl.Name, Depts.Name from Empl  
JOIN Depts  
ON Empl.DepartmentID = Depts.ID
```

Результат:

	Name	Name
1	Mike	Production
2	John	TC
3	Paul	Maintenance
4	Jessie	Production

Присоединение таблиц и вложенные запросы

CROSS JOIN

Возвращает Декартово произведение:

```
SELECT Empl.Name, Depts.Name from Empl  
CROSS JOIN Depts
```

Аналогично результатам ранее:

	Name	Name
1	Mike	Production
2	John	Production
3	Eric	Production
4	Paul	Production
5	Jessie	Production
6	Jonny	Production
7	Mike	TC
8	John	TC
9	Eric	TC
10	Paul	TC

ID	NAME	DEPARTMENT
14	John	Maintenance
15	Eric	Maintenance
16	Paul	Maintenance
17	Jessie	Maintenance
18	Jonny	Maintenance
19	Mike	NULL
20	John	NULL
21	Eric	NULL
22	Paul	NULL
23	Jessie	NULL
24	Jonny	NULL

Присоединение таблиц и вложенные запросы

SELF JOIN

Присоединение таблицы к самой себе:

```
SELECT e1.PositionID, e1.Name, e2.PositionID, e2.Name from  
Empl e1  
JOIN Empl e2  
on e1.PositionID < e2.PositionID
```

Получим что-то вроде иерархии сотрудников:

	PositionID	Name	PositionID	Name
1	1	Mike	2	John
2	1	Mike	5	Eric
3	2	John	5	Eric
4	2	Paul	5	Eric
5	2	Jessie	5	Eric
6	1	Mike	2	Paul
7	1	Mike	2	Jessie

Присоединение таблиц и вложенные запросы

Вложенные запросы

Запросы могут содержать внутри еще запросы

```
select * from Products where Price > (Select avg(Price)
from Products)
```

Этот запрос возвратит все товары, цена которых выше средней:

	ID	Name	Price	Count
1	2	Product 1	223,00	4

Присоединение таблиц и вложенные запросы

Вложенные запросы

Так же можно использовать различные условия выборки:

```
SELECT * FROM Empl  
WHERE Empl.PositionID IN (SELECT Position.ID FROM  
Position WHERE PositionID > 1)
```

Увидим список всех сотрудников, выше JSE:

	ID	Name	DepartmentID	PositionID
1	2	John	2	2
2	3	Eric	NULL	5
3	4	Paul	3	2
4	5	Jessie	1	2

Присоединение таблиц и вложенные запросы

Вкладывать запросы можно вместо названия поля в запросах

```
Select dp.DepartmentName,  
(Select CityName from Cities where CityID = dp.CityID )  
from Departments as dp
```

Такой запрос вернет следующий набор:

	DepartmentName	(No column name)
1	Office	Minsk
2	Product Store	Brest
3	Store	Borisov
4	Office	Grodno
5	Office	Moscow
6	A Store	St. Piterburg
7	HR	Novosibirsk

Correlated Subqueries

Или **соотнесенные вложенные запросы** – такой вид запросов, в которых внутренний запрос зависит от внешнего:

```
SELECT [outer].Name, [outer].DepartmentID FROM Empl [outer]  
WHERE DepartmentID =  
  (SELECT ID FROM Depts [inner]  
   WHERE [inner].ID = [outer].DepartmentID)
```

Этот запрос аналогичен **INNER JOIN**:

	Name	DepartmentID
1	Mike	1
2	John	2
3	Paul	3
4	Jessie	1

ИЗМЕНЕНИЕ ДАННЫХ В ТАБЛИЦАХ

Изменение данных в таблицах

Помимо чтения данных, очень часто приходится выполнять вставку новых данных, обновление уже существующих записей, а так же удаление данных:

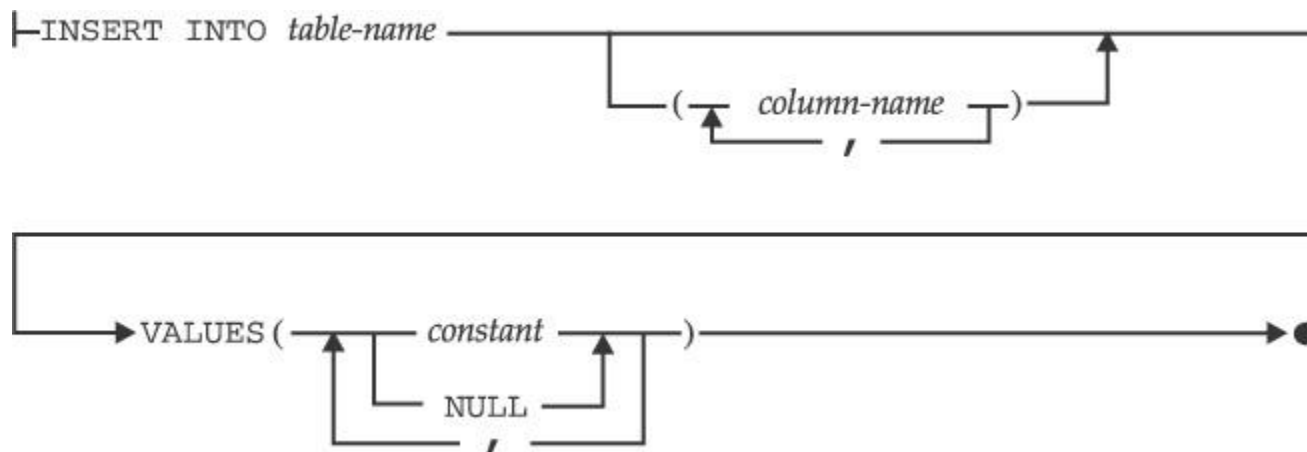
Для выполнения этих операций применяются следующие операторы:

- INSERT
- UPDATE
- DELETE

Изменение данных в таблицах

Вставка новых записей

Для вставки записей применяется оператор INSERT:



Вставлять можно как одну строку за раз, так и несколько.

Пример INSERT запроса

```
INSERT INTO Products (Name, Price, Count)
VALUES ('ProductN', '12,00', 5)
```

Значение в поле с идентификатором не вставляется, так как оно генерируется автоматически:

```
TABLE [dbo].[Products] (
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
```

Так как количество полей соответствует количеству полей в таблице то можно применить сокращенную запись, убрав имена полей:

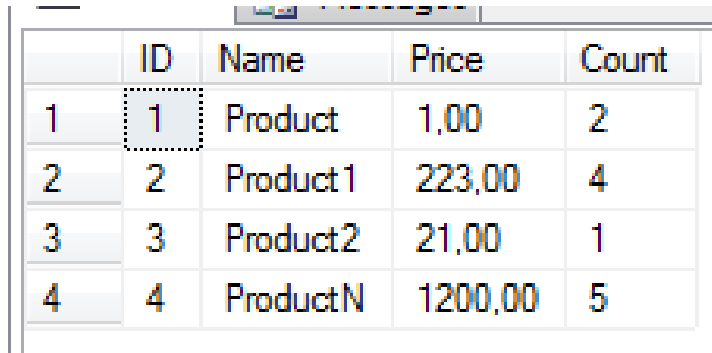
```
INSERT INTO Products
VALUES ('ProductN', '12,00', 5)
```

Вставка нескольких строк

Скопировать все продукты в архив:

```
INSERT INTO OldProducts  
SELECT Name, Price, [Count] FROM Products
```

Все продукты скопированы:

A screenshot of a database application window showing a table with 5 columns: ID, Name, Price, and Count. The table contains 4 rows of data. The first row has ID 1, Name Product, Price 1,00, and Count 2. The second row has ID 2, Name Product1, Price 223,00, and Count 4. The third row has ID 3, Name Product2, Price 21,00, and Count 1. The fourth row has ID 4, Name ProductN, Price 1200,00, and Count 5. The first row is highlighted with a dashed border.

	ID	Name	Price	Count
1	1	Product	1,00	2
2	2	Product1	223,00	4
3	3	Product2	21,00	1
4	4	ProductN	1200,00	5

Удаление строк производится оператором DELETE

Удалять можно как одну строку за раз:

```
DELETE FROM OldProducts WHERE ID='1'
```

Так и несколько сразу:

```
DELETE FROM OldProducts WHERE Price > '1,00'
```

Либо полностью очистить таблицу:

```
DELETE FROM OldProducts
```

Обновление данных

Обновить существующие записи можно используя оператор UPDATE:

```
UPDATE Departments  
SET  
  DepartmentName = 'Human Resources'  
WHERE DepartmentName = 'HR'
```

Обновить название департамента “HR”, заменив его на “Human Resources”

ОБЪЕКТЫ ДЛЯ РАБОТЫ С ДАННЫМИ

Объекты для работы с данными

MS SQL Server имеет ряд **встроенных объектов и механизмов**, предназначенных для работы с данными.

Основные:

- Views или представления
- Пользовательские функции
- Триггеры
- Хранимые процедуры

Views

Представление (View) – сохраненный запрос, результатом которого является виртуальная таблица с данными.

```
SELECT * FROM EmployeeInfo
```

В результате видим сводную информацию по трем таблицам, собранную вместе:

	ID	Name	Department	Position
1	1	Mike	Production	JSE
2	2	John	TC	Developer
3	4	Paul	Maintenance	Developer
4	5	Jessie	Production	Developer

Views

На самом деле мы видим результат работы View:

```
CREATE VIEW EmployeeInfo AS  
SELECT Empl.ID, Empl.Name, Depts.Name as  
Department, Position.Name as Position FROM Empl  
JOIN Depts on Empl.DepartmentID = Depts.ID  
JOIN Position on Empl.PositionID = Position.ID
```

Основное назначение представлений:

- Ограничить доступ пользователя к определенным колонкам или записям;
- Агрегировать данные из нескольких таблиц в одну

Пользовательские функции

Возможности:

Повторное использование логики обработки данных

Можно использовать как отдельный вызов, так и внутри выражений.

Быстро исполняются

Пользовательские функции исполняются быстрее текстовых запросов так как заранее компилируются.

Уменьшение количества трафика

Код функций хранится на сервере, а не передается по каналам связи

Пользовательские функции

Пример простой функции, возвращающей текстовое значение должности сотрудника:

```
ALTER FUNCTION dbo.EmplPos (@id int)
RETURNS nvarchar(50)
WITH EXECUTE AS CALLER
AS
BEGIN
    DECLARE @position nvarchar(50);
    SET @position = (SELECT Position.Name FROM Position
                     JOIN Empl on Position.ID = Empl.PositionID
                     WHERE Empl.ID = @id);
    RETURN(@position);
END;
GO
```

Эту функцию можно использовать так:

```
SELECT Name, dbo.EmplPos(ID) FROM Empl
```

Хранимые процедуры

Набор откомпилированных команд T-SQL:

Возможности

- Имеют входные и выходные параметры
- Могут возвращать наборы и скалярные значения
- Возвращают статус выполнения процедуры

Особенности

- Хранятся на стороне сервера
- Заранее откомпилированы
- Возвращают статус выполнения процедуры
- Позволяют получать данные из таблиц, без прямого доступа к ним

Хранимые процедуры

Пример процедуры, возвращающей идентификатор нового сотрудника:

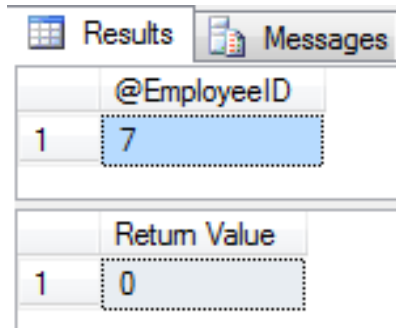
```
CREATE PROCEDURE dbo.NewEmployee
@EmployeeName nvarchar(50),
@DeptName nvarchar(50),
@PositionName nvarchar(50),
@EmployeeID int OUTPUT
AS
BEGIN
INSERT INTO Empl (Name, DepartmentID, PositionID)
VALUES
(
@EmployeeName,
(SELECT ID FROM Depts WHERE Depts.Name = @DeptName),
(SELECT ID FROM Position WHERE Position.Name = @PositionName)
);
SET @EmployeeID = SCOPE_IDENTITY()
END
```

Хранимые процедуры

Вызываем процедуру:

```
DECLARE @return_value int, @EmployeeID int
EXEC      @return_value = [dbo].[NewEmployee]
           @EmployeeName = N'EmplFromSP',
           @DeptName = N'Production',
           @PositionName = N'JSE',
           @EmployeeID = @EmployeeID OUTPUT
SELECT    @EmployeeID as N'@EmployeeID'
SELECT    'Return Value' = @return_value
```

И видим, что вставка сработала как надо:



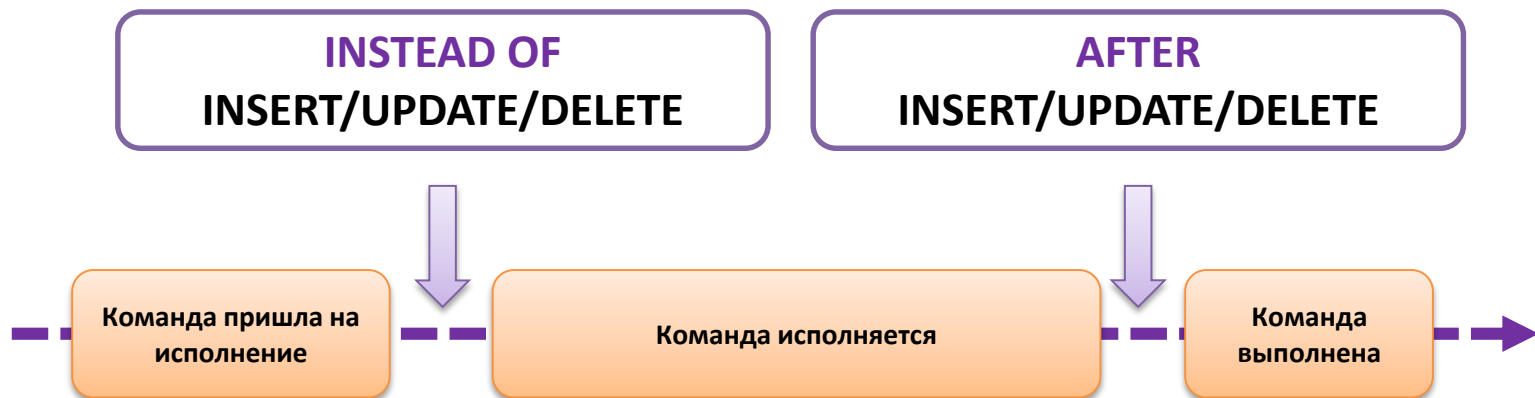
	@EmployeeID
1	7

	Return Value
1	0

Триггеры

Триггеры – особый вид хранимых процедур, которые выполняются автоматически при **вставке/обновлении/удалении** данных.

DML триггеры бывают двух видов:



Триггеры

Особенности триггеров:

Исполнение

- Могут откатывать текущую транзакцию
- Любое выражение после, идущее после отката транзакции будет выполнено
- Модификация данных после отката будет выполнена

Ограничения

- Триггер может быть создан только для одной таблицы
- Триггер выполняется только для одной БД
- Триггер должен находиться в той же схеме, что и объект триггера
- **INSTEAD OF** триггер не может быть определен для таблицы, имеющий внешний ключ с установленным каскадным правилом

Пример триггера

Триггер, проверяющий имя сотрудника перед вставкой, откатывающий транзакцию если имя уже есть:

```
CREATE TRIGGER nameCheck ON Empl
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON
    DECLARE @name nvarchar(50);
    SET @name = (SELECT Name FROM inserted)
    IF (NOT EXISTS(SELECT Name FROM Empl WHERE Empl.Name = @name))
        INSERT INTO Empl SELECT Name, DepartmentID, PositionID FROM
        inserted
    ELSE
        PRINT('ERROR')
END
```

СПАСИБО ЗА ВНИМАНИЕ!

ВОПРОСЫ?

Работа с данными на базе MS SQL Server и T-SQL

Author: Pavel Belyakov

poulbelyakov@gmail.com