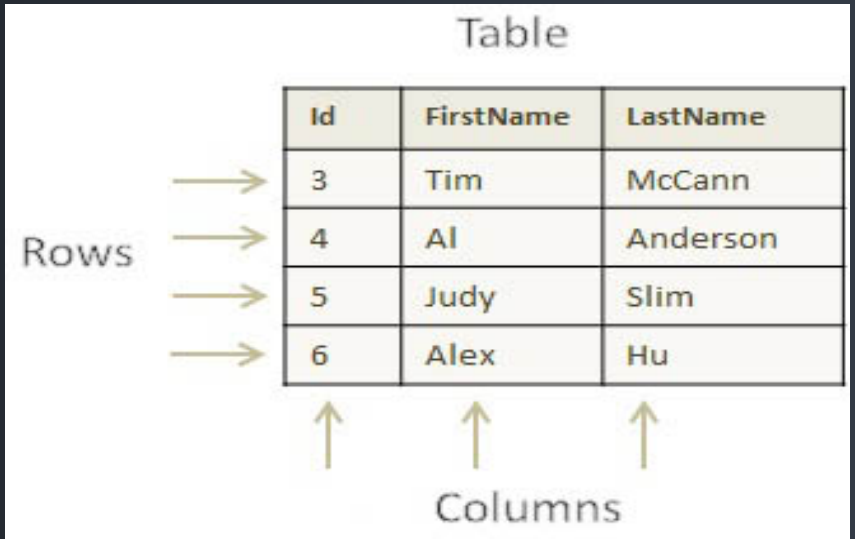
What is SQL

➢SQL is a language used to retrieve and manipulate data in a RDMS.

➢SQL stands for *Structured Query Language* 标准查语言

➢SQL lets you *access and manipulate databases* 访问 检索 操作数据库中的数据



What Can SQL do?

- execute queries against a database
- retrieve data from a database
- insert records in a database
- update records in a database
- delete records from a database
- create new databases
- create new tables in a database
- create stored procedures in a database
- create views in a database
- set permissions on tables, procedures, and views

关系型数据库

A relational database stores data in tables. Each table has a number of rows and columns.

SQL Clause 6种类型

DQL(Data Query Language)：SELECT，WHERE，ORDER BY，GROUP BY，HAVING

HAVING 子句应用于结果集中的行。 只有满足 *HAVING* 条件的组才会显示在查询输出中

```
SELECT titles.pub_id, AVG(titles.price)
FROM titles INNER JOIN publishers ON titles.pub_id = publishers.pub_id
WHERE publishers.state = 'CA'
GROUP BY titles.pub_id
HAVING AVG(price) > 10   这个可以独立存在，也可以和where同时存在
```

- DML(Data Manipulation Language ）：INSERT，UPDATE, DELETE
- TPL(Transaction Process Language)： BEGIN TRANSACTION，COMMIT , ROLLBACK
- DCL(Data Control Language)： GRANT， REVOKE
- DDL(Data Definition Language)： CREATE，DROP
- CCL(Pointer Control Language)： DECLARE CURSOR，FETCH INTO，UPDATE WHERE CURRENT

游标的滚动行为和用于生成游标所操作的结果集的查询

SQL Data Types 所有数据的(存储)类型

Exact numerics	[Bigint] [numeric] [bit] [smallint] [decimal] [small] [money] [int] [tinyint] [money]
Approximate numerics	[float] [real]
Date and time	[date] [datetime2] [datetimeoffset] [smalldatetime] [datetime] [time]
Character strings	[char] [varchar] [text]
Unicode character strings	[nchar] [nvarchar] [ntext]
Binary strings	[binary] [varbinary] [image]
Other data types	[cursor] [rowversion] [hierarchyid] [uniqueidentifier] [sql_variant] [xml] [Spatial Geometry Types] [Spatial Geography Types] [table]

SQL Statements – *INSERT INTO*

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

```
INSERT  INTO dbo.Employee
        ( ID ,
          Name ,
          Gender ,
          Age ,
          Birthday ,
          Nationality ,
          City
        )
VALUES  ( 3116910 , -- ID - int
          'Aleda' , -- Name - varchar(100) 添加字符串的信息
          'Female' , -- Gender - varchar(10)
          27, -- Age - int
          '1990-09-23' , -- Birthday - datetime
          'Germany' , -- Nationality - varchar(50)
          'Dusseldorf'  -- City - varchar(255)
        )
```

```
INSERT  INTO dbo.Employee
VALUES  ( 3116911 , -- ID - int
          'Eric' , -- Name - varchar(100)
          'Male' , -- Gender - varchar(10)
          29, -- Age - int
          '1988-07-15' , -- Birthday - datetime
          'Germany' , -- Nationality - varchar(50)
          'Dusseldorf'  -- City - varchar(255)
        )
```

SQL Statements - *SELECT*

➢ *SELECT general form*

```
SELECT [column-names]
FROM [table-name]
```

```
SELECT * FROM [dbo].[Employee]
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan

➢ *SELECT with condition*

```
SELECT [column-names]
FROM [table-name]
WHERE [condition]
```

```
SELECT   *
FROM     [dbo].[Employee]
WHERE    Nationality = 'Italy'
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
2	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan

➢ *SELECT with condition and sort*

```
SELECT [column-names]
FROM [table-name]
WHERE [condition]
ORDER BY [sort-order]
```

```
SELECT   *
FROM     [dbo].[Employee]
WHERE    Nationality = 'Germany'
ORDER BY Age ASC 递增 (DESC递减)
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg

SQL Operators

❑ **Comparison Operators:**

- Comparison operators are used to compare the column data with specific values in a condition.
- Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

Comparison Operators	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

```
SELECT   ID ,
         Name ,
         Age
FROM     dbo.Employee
WHERE    Age >= 30
```

	ID	Name	Age
1	3116917	Redwan	30
2	3116918	Tom	32
3	3116919	Tim	33
4	3116920	Billy	36

❑ **There are three Logical Operators namely, *AND, OR,* and *NOT*.**

Logical Operators	Description
OR	For the row to be selected at least one of the conditions must be true.
AND	For a row to be selected all the specified conditions must be true.
NOT	For a row to be selected the specified condition must be false.

```
SELECT   ID, Name, Gender, Age, Nationality
FROM     dbo.Employee
WHERE    ( Age < 30 AND Nationality = 'Germany')
         OR ( Age > 30 AND Nationality = 'Portugal')
```

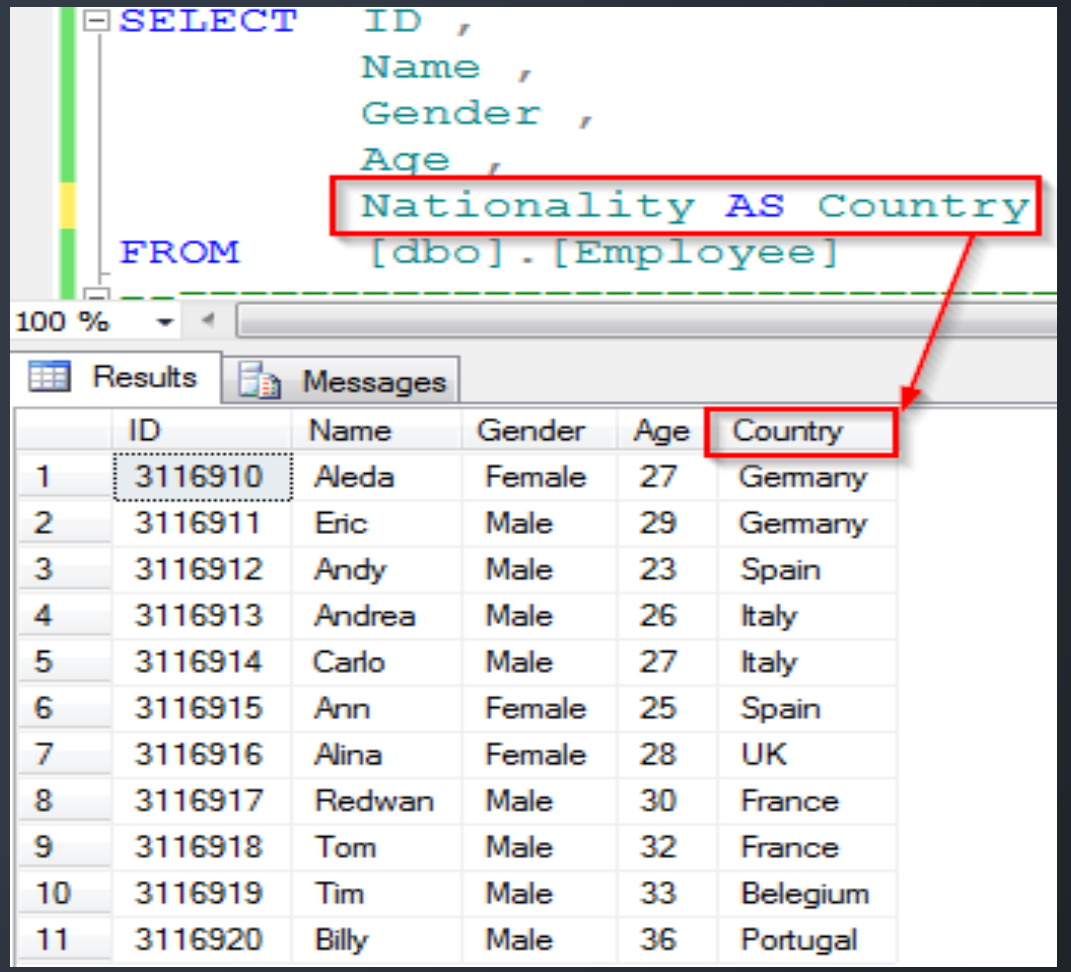
	ID	Name	Gender	Age	Nationality
1	3116910	Aleda	Female	27	Germany
2	3116911	Eric	Male	29	Germany
3	3116920	Billy	Male	36	Portugal

SQL Statement - *Alias* 列名称的别名

- SQL Aliases are defined for columns and tables. Basically aliases is created to make the column selected more readable.

Aliases for columns

```
SELECT [Column] AS [Alias]
FROM [table];
or
SELECT [Column] [Alias]
FROM [table];
```



	ID	Name	Gender	Age	Country
1	3116910	Aleda	Female	27	Germany
2	3116911	Eric	Male	29	Germany
3	3116912	Andy	Male	23	Spain
4	3116913	Andrea	Male	26	Italy
5	3116914	Carlo	Male	27	Italy
6	3116915	Ann	Female	25	Spain
7	3116916	Alina	Female	28	UK
8	3116917	Redwan	Male	30	France
9	3116918	Tom	Male	32	France
10	3116919	Tim	Male	33	Belegium
11	3116920	Billy	Male	36	Portugal

SQL Statement – WHERE LIKE

❑ Wildcard

Wildcard	Wildcard & Description
%	The percent sign (%) Matches one or more characters. 匹配多个字符 MS Access uses the asterisk (*) wildcard character instead of the percent sign (%) wildcard character.
_	**The underscore (_)** Matches one character. 匹配单个字符 MS Access uses a question mark (?) instead of the underscore (_) to match any one character.

❑ LIKE

Comparision Operators	Description
LIKE	**column value is similar to specified character(s).**

```
SELECT [columns]
FROM [table]
WHERE [Column] LIKE 'A%';
```

```
SELECT   *
FROM     dbo.Employee
WHERE    Name LIKE 'A%'
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
3	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
4	3116915	Ann	Female	25	1992-02-16 00:00:00.000	Spain	Barcelona
5	3116916	Alina	Female	28	1989-09-21 00:00:00.000	UK	London

```
SELECT   *
FROM     dbo.Employee
WHERE    Name LIKE 'T_m'
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116918	Tom	Male	32	1985-06-18 00:00:00.000	France	Lyon
2	3116919	Tim	Male	33	1984-07-26 00:00:00.000	Belegium	Brussels

SQL Statement – *IN… \ NOT IN…*

❑ ***IN :***

The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition. 提供多个条件的备选项

❑ ***NOT IN :***

- used when you want to retrieve a column that has no entries in the table or referencing table. 排除指定的多个备选项

```
SELECT   *
FROM     dbo.Employee
WHERE    Nationality IN ( 'Italy',
'Germany' )
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg
3	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
4	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan

```
SELECT   *
FROM     dbo.Employee
WHERE    Nationality NOT IN ( 'Italy',
'Germany' )
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
2	3116915	Ann	Female	25	1992-02-16 00:00:00.000	Spain	Barcelona
3	3116916	Alina	Female	28	1989-09-21 00:00:00.000	UK	London
4	3116917	Redwan	Male	30	1987-01-29 00:00:00.000	France	Pairs
5	3116918	Tom	Male	32	1985-06-18 00:00:00.000	France	Lyon
6	3116919	Tim	Male	33	1984-07-26 00:00:00.000	Belegium	Brussels
7	3116920	Billy	Male	36	1981-02-03 00:00:00.000	Portugal	Lisbon

SQL Statement – *DISTINCT* 取消重复项

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

Without Distinct

```
SELECT   Nationality
FROM     dbo.Employee
```

	Nationality
1	Germany
2	Germany
3	Spain
4	Italy
5	Italy
6	Spain
7	UK
8	France
9	France
10	Belegium
11	Portugal

With Distinct

```
SELECT   DISTINCT
         Nationality
FROM     dbo.Employee
```

	Nationality
1	Belegium
2	France
3	Germany
4	Italy
5	Portugal
6	Spain
7	UK

SQL Statement – *Top*

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

并非所有的数据库都支持Top申明

*MySQL supports the **LIMIT** clause to fetch limited number of records*
*Oracle uses the **ROWNUM** command to fetch a limited number of records.*

```
SELECT TOP number|percent column_name(s)
FROM table_name
--WHERE [condition]
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg

```
SELECT TOP 1
       *
FROM   dbo.Employee
WHERE  Nationality IN ( 'Germany' )
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf

SQL Statement - *BETWEEN...AND*

BETWEEN...AND	column value is between two values, including the end values specified in the range. 约束值的范围

```
SELECT Column1, Column2,…
FROM Table
WHERE Column BETWEEN 10
AND 15;
```

>= AND <=

[minimum, Maximum]

```
SELECT   *
FROM     dbo.Employee
WHERE    Age BETWEEN 23 AND 30
ORDER BY Age ASC
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
2	3116915	Ann	Female	25	1992-02-16 00:00:00.000	Spain	Barcelona
3	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
4	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan
5	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
6	3116916	Alina	Female	28	1989-09-21 00:00:00.000	UK	London
7	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg
8	3116917	Redwan	Male	30	1987-01-29 00:00:00.000	France	Pairs

SQL Statement – *IS NULL \ IS NOT NULL*

判断是否有值存在

A column value is NULL if it does not exist.

The IS NULL operator is used to display all the rows for columns that do not have a value.

```
SELECT Column(s)
FROM Table
WHERE Column IS NULL
```

```
SELECT Column(s)
FROM Table
WHERE Column IS NOT NULL
```

```
SELECT   *
FROM     dbo.Employee
WHERE    Name IS NULL
```

	ID	Name	Gender	Age	Birthday	Nationality	City

```
SELECT   *
FROM     dbo.Employee
WHERE    Name IS NOT NULL AND City IN ('Roma',
'Milan', 'Madrid')
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
2	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
3	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan

SQL Statement - *Group By*

- The SQL GROUP BY Clause is used along with the group functions to retrieve data grouped according to one or more columns.

```
SELECT column(s), Count|SUM (column)
FROM employee
GROUP BY column(s);
```

✓ How many employees in each country ?

```
SELECT  Nationality ,
        COUNT(Name员工的名字) AS Count_Employee
FROM    dbo.Employee
GROUP BY Nationality
```

	Nationality	Count_Employee
1	Belegium	1
2	France	2
3	Germany	2
4	Italy	2
5	Portugal	1
6	Spain	2
7	UK	1

SQL Statement – *HAVING*

- HAVING filters records that work on summarized GROUP BY results.
- HAVING applies to summarized group records 结果集, whereas WHERE applies to individual records 单个行.
- Only the groups that meet the HAVING criteria will be returned. 只有符合条件的分组会被返回
- AVING requires that a GROUP BY clause is present.
- WHERE and HAVING can be in the same query.

General syntax

```
SELECT column-names
FROM table-name
WHERE condition
GROUP BY column-names
HAVING condition
```

General syntax with ORDER BY

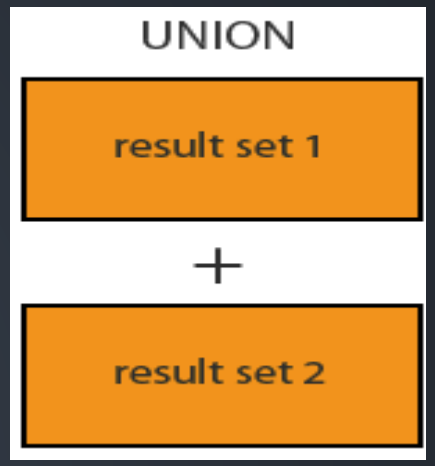
```
SELECT column-names
FROM table-name
WHERE condition
GROUP BY column-names
HAVING condition
ORDER BY column-names
```

```
SELECT Nationality, COUNT(Name) AS Count_Employee
FROM dbo.Employee
GROUP BY Nationality
HAVING COUNT(Name) > 1 过滤结果集中的数据，判断条件
ORDER BY Nationality ASC
```

	Nationality	Count_Employee
1	France	2
2	Germany	2
3	Italy	2
4	Spain	2

SQL Statement – *Union*

- UNION combines the result sets of two queries.
- Column data types in the two queries must match.
- UNION combines by column position rather than column name.




```
SELECT column-names
FROM table-name
UNION
SELECT column-names
FROM table-name
```

```
SELECT * FROM dbo.Employee
UNION ALL
SELECT * FROM [dbo].[Consultant]
```

dbo.Employee

[dbo].[Consultant]

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan
6	3116915	Ann	Female	25	1992-02-16 00:00:00.000	Spain	Barcelona
7	3116916	Alina	Female	28	1989-09-21 00:00:00.000	UK	London
8	3116917	Redwan	Male	30	1987-01-29 00:00:00.000	France	Pairs
9	3116918	Tom	Male	32	1985-06-18 00:00:00.000	France	Lyon
10	3116919	Tim	Male	33	1984-07-26 00:00:00.000	Belegium	Brussels
11	3116920	Billy	Male	36	1981-02-03 00:00:00.000	Portugal	Lisbon
12	79981	Fabio	Male	31	1986-01-17 00:00:00.000	Italy	Roma

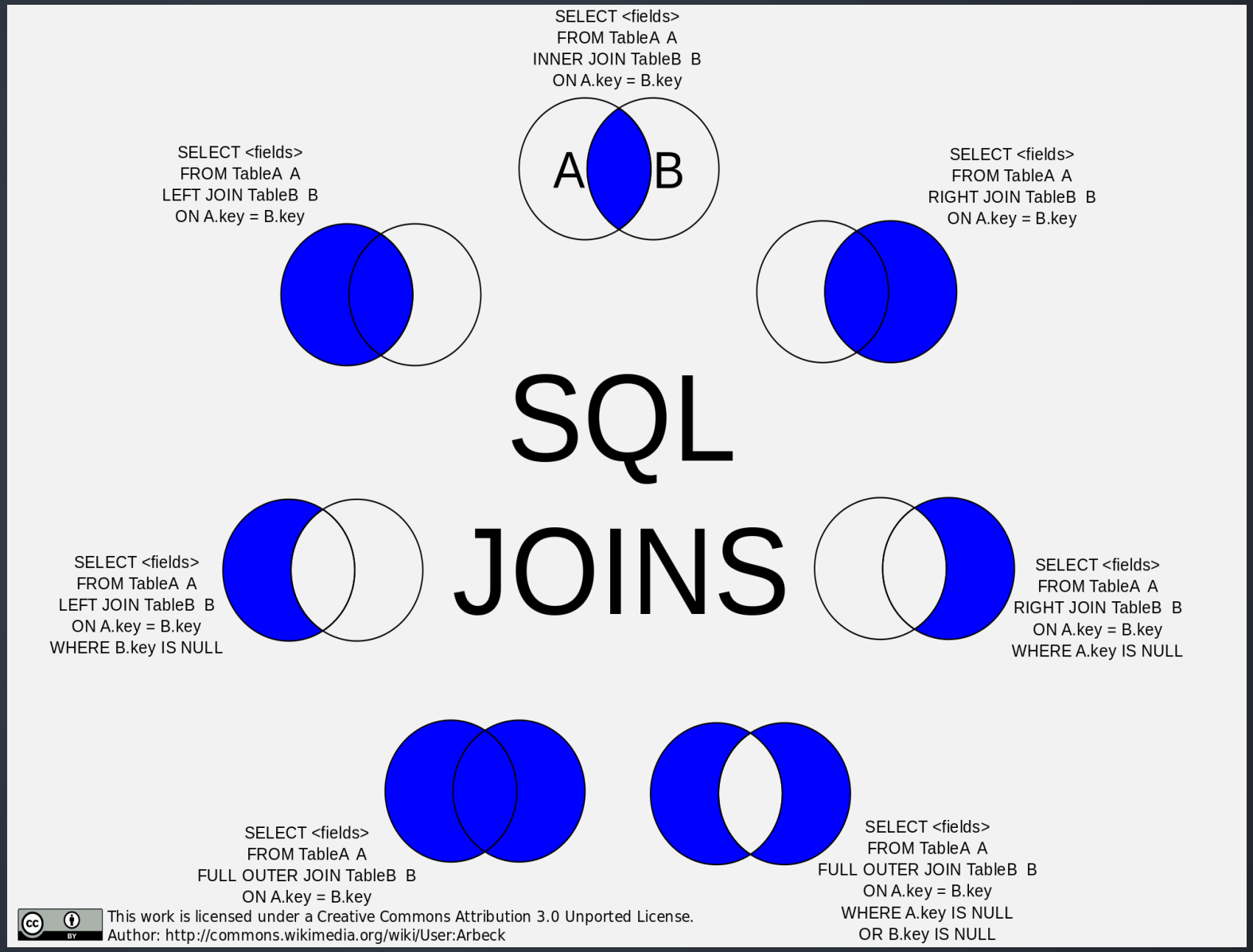
SQL Statements - JOIN

A SQL JOIN combines records from two tables.

JOIN ON 后面提供Key的等式

A query can contain zero, one, or multiple JOIN operations.

INNER JOIN is the same as JOIN; the keyword INNER is optional.



SQL Statements – JOIN example

✓ *Get employee salary*

```
SELECT   E.ID ,
         E.Name ,
         Nationality ,
         S.Salary_IN_USD
FROM     dbo.Employee AS E
         LEFT JOIN dbo.Salary AS S ON E.ID = S.ID
```

	ID	Name	Nationality	Salary_IN_USD
1	3116910	Aleda	Germany	2239.19
2	3116911	Eric	Germany	2850.97
3	3116912	Andy	Spain	2600.16
4	3116913	Andrea	Italy	2131.91
5	3116914	Carlo	Italy	1850.36
6	3116915	Ann	Spain	3010.3
7	3116916	Alina	UK	2100.27
8	3116917	Redwan	France	2727.19
9	3116918	Tom	France	3109.17
10	3116919	Tim	Belegium	2200.2
11	3116920	Billy	Portugal	2819

✓ *Get employee salary and department*

```
SELECT   E.ID ,
         E.Name ,
         Nationality ,
         S.Salary_IN_USD ,
         D.Dept ,
         D.[Direct Manager]
FROM     dbo.Employee AS E
         LEFT JOIN dbo.Salary AS S ON E.ID = S.ID
         LEFT JOIN dbo.Department AS D ON E.ID = D.ID
```

	ID	Name	Nationality	Salary_IN_USD	Dept	Direct Manager
1	3116910	Aleda	Germany	2239.19	NTD	NTD Manager
2	3116911	Eric	Germany	2850.97	TMO	TMO Manager
3	3116912	Andy	Spain	2600.16	TMO	TMO Manager
4	3116913	Andrea	Italy	2131.91	NTD	NTD Manager
5	3116914	Carlo	Italy	1850.36	TMO	TMO Manager
6	3116915	Ann	Spain	3010.3	TMO	TMO Manager
7	3116916	Alina	UK	2100.27	TMO	TMO Manager
8	3116917	Redwan	France	2727.19	FBB Product Line	FBB Line Manager
9	3116918	Tom	France	3109.17	MBB Product Line	MBB Line Manager
10	3116919	Tim	Belegium	2200.2	TMO	NTD Manager
11	3116920	Billy	Portugal	2819	Core Network Product Line	Core Network Line Manager

SQL Statement - Select Into

- SELECT INTO copies data from one table into a new table.
- SELECT INTO creates a new table located in the default file-group.

```
SELECT column-names
  INTO new-table-name
  FROM table-name
```

```
SELECT   ID ,
         Name ,
         Gender ,
         Age ,
         Birthday
INTO     tmp_Employee
FROM     dbo.Employee
WHERE    Age > 30

生成结果集到新的Table表中
SELECT * FROM tmp_Employee
```

dbo.Employee

	ID	Name	Gender	Age	Birthday
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000
6	3116915	Ann	Female	25	1992-02-16 00:00:00.000
7	3116916	Alina	Female	28	1989-09-21 00:00:00.000
8	3116917	Redwan	Male	30	1987-01-29 00:00:00.000
9	3116918	Tom	Male	32	1985-06-18 00:00:00.000
10	3116919	Tim	Male	33	1984-07-26 00:00:00.000
11	3116920	Billy	Male	36	1981-02-03 00:00:00.000

dbo.tmp_Employee

	ID	Name	Gender	Age	Birthday
1	3116918	Tom	Male	32	1985-06-18 00:00:00.000
2	3116919	Tim	Male	33	1984-07-26 00:00:00.000
3	3116920	Billy	Male	36	1981-02-03 00:00:00.000

SQL Statement – *Insert Into… Select…*

- INSERT INTO SELECT copies data from one table to another table. 复制表的数据到另一个表中，列的信息需要完全匹配
- INSERT INTO SELECT requires that data types in source and target tables match.
  - The general syntax is:

```
INSERT INTO table-name (column-names)
SELECT column-names
FROM table-name
WHERE condition
```

```
INSERT   INTO [dbo].[tmp_Employee]
         ( ID ,
           Name ,
           Gender ,
           Age ,
           Birthday
         )
         SELECT   ID ,
                  Name ,
                  Gender ,
                  Age ,
                  Birthday
         FROM     dbo.Employee
WHERE Age < 30
```

	ID	Name	Gender	Age	Birthday
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000
6	3116915	Ann	Female	25	1992-02-16 00:00:00.000
7	3116916	Alina	Female	28	1989-09-21 00:00:00.000

SQL Statement – *Update… SET…*

- The SQL UPDATE statement allows you to update an existing record in the database.
- The UPDATE command uses a WHERE clause. If you don't use a WHERE clause, all rows will be updated. In fact, the syntax for a basic UPDATE statement is very similar to a SELECT statement.

❖ Updating Single Field

```
UPDATE table-name
SET column-name = 'value'
WHERE condition;
```

❖ Updating Multiple Fields

```
UPDATE Individual
SET column-name1 = 'value1',
column-name2 = 'value2'
WHERE condition;
```

```
UPDATE   dbo.Employee
SET      Nationality = 'American' ,
         City = 'New York'
WHERE    Name = 'Eric'
```

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116911	Eric	Male	29	1988-07-15 00:00:00.000	Germany	Nuremgburg

	ID	Name	Gender	Age	Birthday	Nationality	City
1	3116911	Eric	Male	29	1988-07-15 00:00:00.000	American	New York

SQL Statement – *Delete… From…*

- The SQL DELETE statement allows you to delete a record from the database.
- The DELETE command uses a WHERE clause. If you don't use a WHERE clause, all rows in the table will be deleted. Again, as with the UPDATE statement, the syntax for a basic DELETE statement is similar to a SELECT statement.

```
DELETE
FROM table-name
WHERE column-name = 'value';
```

```
DELETE   FROM dbo.tmp_Employee
WHERE    Name LIKE 'A%'
```

	ID	Name	Gender	Age	Birthday
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000
6	3116915	Ann	Female	25	1992-02-16 00:00:00.000
7	3116916	Alina	Female	28	1989-09-21 00:00:00.000

	ID	Name	Gender	Age	Birthday
1	3116911	Eric	Male	29	1988-07-15 00:00:00.000
2	3116914	Carlo	Male	27	1990-04-19 00:00:00.000

SQL Statement – *Create & Drop Database*

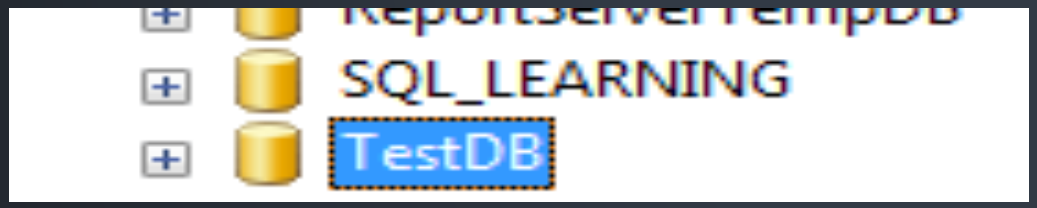
- The CREATE DATABASE statement is used to create a new SQL database.

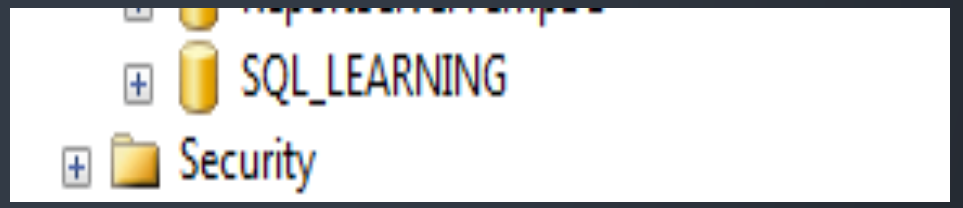
```
CREATE DATABASE databasename;
```

```
DROP DATABASE databasename;
```

```
CREATE DATABASE TestDB
```

```
DROP DATABASE TestDB
```



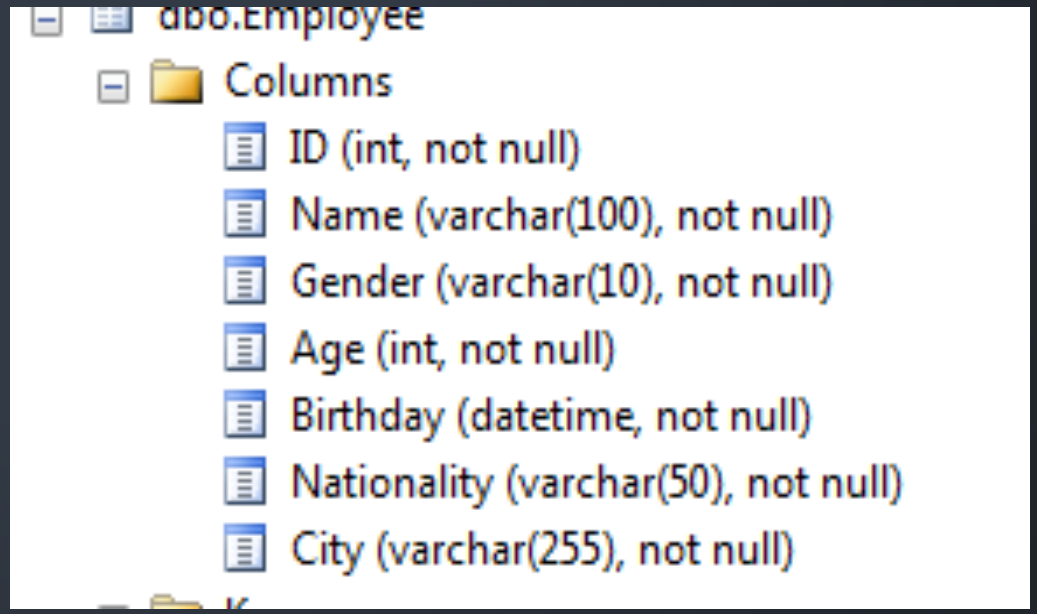


SQL Statement – *Create Table*

- You create a table using the CREATE TABLE command.

```
CREATE TABLE table_name
(
      column_name_1 datatype,
      column_name_2 datatype,
...
);
```

```
USE [Book Store] 需要指定数据库名称，然后添加表格 !!!
CREATE TABLE [dbo].[Employee](
[ID] [int] NOT NULL,
[Name] [varchar](100) NOT NULL,
[Gender] [varchar](10) NOT NULL,
[Age] [int] NOT NULL,
[Birthday] [datetime] NOT NULL,
[Nationality] [varchar](50) NOT NULL,
[City] [varchar](255) NOT NULL
) ON [PRIMARY]
GO
```



SQL Statement – *Create Table with constraint*

```
CREATE TABLE table_name (
    column1 datatype 数据(存储)类型 constraint 约束条件,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

```
CREATE TABLE [dbo].[Employee](
[ID] [int] NOT NULL,
[Name] [varchar](100) NOT NULL,
[Gender] [varchar](10) NOT NULL,
[Age] [int] NOT NULL,
[Birthday] [datetime] NOT NULL,
[Nationality] [varchar](50) NOT NULL,
[City] [varchar](255) NOT NULL
) ON [PRIMARY]
GO
```

SQL Statement – *Alter Table*

- Modify the table using the ALTER TABLE command.

- Here's the syntax for adding a column.

```
ALTER TABLE table_name
ADD column_name datatype;
```

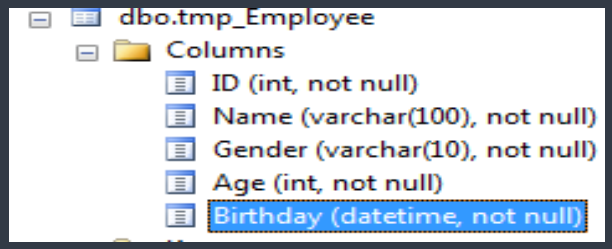
```
ALTER TABLE dbo.tmp_Employee
ADD Married VARCHAR(10) NULL
```

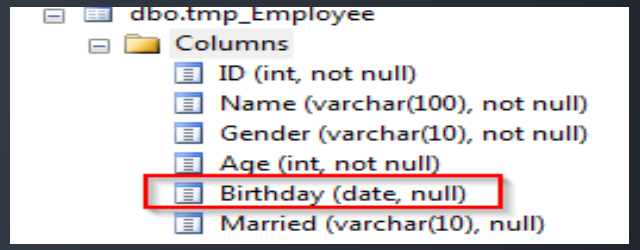
	ID	Name	Gender	Age	Birthday	Maried
1	3116911	Eric	Male	29	1988-07-15 00:00:00.000	NULL
2	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	NULL

- To change the data type, use ALTER COLUMN instead of ADD.

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

```
ALTER TABLE dbo.tmp_Employee
ALTER COLUMN Birthday DATE
```

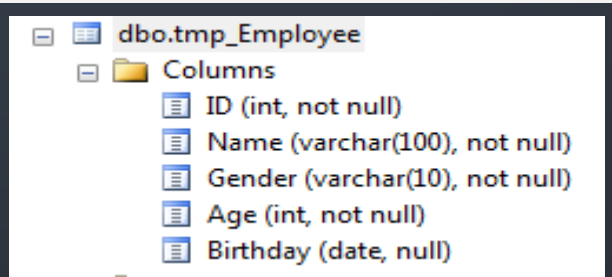




- Use DROP COLUMN followed by the name of the column you want to drop.

```
ALTER TABLE table_nameDROP COLUMN column_name;
```

```
ALTER TABLE dbo.tmp_Employee
DROP COLUMN [Married];
```



SQL Statement – *Drop Table & Truncate table*

- The DROP TABLE statement is used to drop an existing table in a database.

```
DROP TABLE table_name;
```

```
DROP TABLE dbo.tmp_Employee
```

Delete the table object permanently

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE dbo.tmp_Employee
```

Delete the table data permanently

SQL Statement – *PRIMARY KEY Constraint*

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.

```
CREATE TABLE Persons (
    column-name1 int NOT NULL PRIMARY KEY,
    column-name2 varchar(255) NOT NULL,
    column-name3 varchar(255),
    column-name4 Age int
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    Name varchar(255) NOT NULL,
    Country varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,Name)
);
```

- Allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns.

SQL Statement – *CREATE INDEX*

❑ The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

❖ General Syntax:

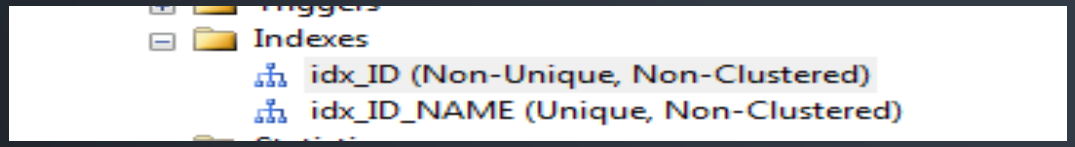
```
CREATE INDEX index_name
ON table_name
(column1, column2, ...);
```

❖ CREATE UNIQUE INDEX Syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1,
column2, ...);
```

```
CREATE INDEX idx_ID
ON dbo.Employee
(ID)
```

```
CREATE UNIQUE INDEX idx_ID_NAME ON
dbo.Employee
(ID, Name)
```



Note: *Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.*

SQL Statement – *Working With Date & Time*

- The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- **DATE:** defines a date in SQL Server; format YYYY-MM-DD…
- **DATETIME**: defines a date that is combined with a time of day with fractional seconds that is based on a 24-hour clock.
  - *Date Range: [January 1, 1753, through December 31 9999]*
  - *Time Range: [00:00:00 through 23:59:59.997]*
- **DATETIME2:** datetime2 can be considered as an extension of the existing datetime type that has a larger date range;
  - *Date Range: 0001-01-01 through 9999-12-31; January 1,1 CE through December 31, 9999 CE*
  - *Time Range: 00:00:00 through 23:59:59.9999999*
- **SMALLDATETIME**: defines a date that is combined with a time of day. format: YYYY-MM-DD HH:MI:SS
  - Date Range: 1900-01-01 through 2079-06-06 (January 1, 1900, through June 6, 2079)
  - Time Range: 00:00:00 through 23:59:59 *(2007-05-09 23:59:59 will round to 2007-05-10 00:00:00)*
- **TIME**: defines a time of a day.
  - *Range: 00:00:00.0000000 through 23:59:59.9999999 (00:00:00.000 through 23:59:59.999 for Informatica)*
  - *Default string literal format : hh:mm:ss[.nnnnnnn] for Informatica)*

- *https://docs.microsoft.com/en-us/sql/t-sql/data-types/date-and-time-types*

SQL Server String Functions

- SQL Server has many built-in functions. That contains the string, numeric, date, conversion, and advanced functions in SQL Server.

- ***SQL Server String Functions***

Function	Description
ASCII	Returns the number code that represents the specific character
CHAR	Returns the ASCII character based on the number code
CHARINDEX	Returns the location of a substring in a string
CONCAT	Concatenates two or more strings together
Concat with +	Concatenates two or more strings together
DATALENGTH	Returns the length of an expression (in bytes)
LEFT	Extracts a substring from a string (starting from left)
LEN	Returns the length of the specified string
LOWER	Converts a string to lower-case
LTRIM	Removes leading spaces from a string
NCHAR	Returns the Unicode character based on the number code
PATINDEX	Returns the location of a pattern in a string
REPLACE	Replaces a sequence of characters in a string with another set of characters
RIGHT	Extracts a substring from a string (starting from right)
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string with a specified number of spaces
STR	Returns a string representation of a number
STUFF	Deletes a sequence of characters from a string and then inserts another sequence of characters into the string, starting at a specified position
SUBSTRING	Extracts a substring from a string
UPPER	Converts a string to upper-case

```
SELECT
CONCAT('Name=', Name,
' Nationality=',
Nationality,
' Age=', Age)
FROM    dbo.Employee
```

	(No column name)
1	Name=Aleda Nationality=Germany Age=27
2	Name=Eric Nationality=American Age=29
3	Name=Andy Nationality=Spain Age=23
4	Name=Andrea Nationality=Italy Age=26
5	Name=Carlo Nationality=Italy Age=27
6	Name=Ann Nationality=Spain Age=25
7	Name=Alina Nationality=UK Age=28
8	Name=Redwan Nationality=France Age=30
9	Name=Tom Nationality=France Age=32
10	Name=Tim Nationality=Belegium Age=33
11	Name=Billy Nationality=Portugal Age=36

SQL Server Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
AVG	Returns the average value of an expression
CEILING	Returns the smallest integer value that is greater than or equal to a number
COUNT	Returns the count of an expression
FLOOR	Returns the largest integer value that is equal to or less than a number
MAX	Returns the maximum value of an expression
MIN	Returns the minimum value of an expression
RAND	Returns a random number or a random number within a range
ROUND	Returns a number rounded to a certain number of decimal places
SIGN	Returns a value indicating the sign of a number
SUM	Returns the summed value of an expression

```
SELECT AVG(Age)
FROM dbo.Employee
```

	(No column name)
1	28

```
SELECT   MAX(Age)
FROM     dbo.Employee
```

	(No column name)
1	36

```
SELECT
SUM(Salary_IN_USD)
FROM     dbo.Salary
```

	(No column name)
1	27638.72

SQL Server Date Functions

Function	Description
CURRENT_TIMESTAMP	Returns the current date and time
DATEADD	Returns a date after a certain time/date interval has been added
DATEDIFF	Returns the difference between two date values, based on the interval specified
DATENAME	Returns a specified part of a given date, as a string value
DATEPART	Returns a specified part of a given date, as an integer value
DAY	Returns the day of the month (from 1 to 31) for a given date
GETDATE	Returns the current date and time
GETUTCDATE	Returns the current UTC date and time
MONTH	Returns the month (from 1 to 12) for a given date
YEAR	Returns the year (as a four-digit number) for a given date

```
SELECT DATEADD(month, 2, '2017-10-12') AS DateAdd;
```

	DateAdd
1	2017-12-12 00:00:00.000

```
SELECT DATEADD(month, -2, '2017-10-12') AS DateAdd;
```

	DateAdd
1	2017-08-12 00:00:00.000

SQL Server Conversion Functions

Function	Description
CAST	Converts an expression from one data type to another
CONVERT	Converts an expression from one data type to another

```
SELECT CAST(25.65 AS int);
```

	(No column name)
1	25

```
SELECT CAST('2017-11-12' AS datetime);
```

	(No column name)
1	2017-11-12 00:00:00.000

```
SELECT CONVERT(int, 25.65);
```

	(No column name)
1	25

```
SELECT CONVERT(datetime, '2017-11-12');
```

	(No column name)
1	2017-11-12 00:00:00.000

SQL Server Advanced Functions

Function	Description
COALESCE	Returns the first non-null expression in a list
CURRENT_USER	Returns the name of the current user in the SQL Server database
ISDATE	Returns 1 if the expression is a valid date, otherwise 0
ISNULL	Lets you return an alternative value when an expression is NULL
ISNUMERIC	Returns 1 if the expression is a valid number, otherwise 0
NULLIF	Compares two expressions
SESSION_USER	Returns the user name of the current session in the SQL Server database
SESSIONPROPERTY	Returns the setting for a specified option of a session
SYSTEM_USER	Returns the login name information for the current user in the SQL Server database
USER_NAME	Returns the user name in the SQL Server database

```
SELECT ISDATE('2017/13/26')
```

(No column name)
0

```
SELECT ISDATE('2017/12/26')
```

(No column name)
1

User-defined Functions ==> 自定义函数

- A user-defined function is a database object that encapsulates one or more Transact-SQL statements for reuse. This definition is similar to the one for stored procedures, but there are many important differences between user-defined functions and stored procedures—the most pronounced being what types of data they can return. Let's create one so you can see how easy they are to create and reference.
- SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.

❖ Advantages

- ✓ It allow modular programming.
- ✓ Allow faster execution.
- ✓ Can reduce network traffic.

❖ Types of functions

Scalar Function

- User-defined scalar functions return a single data value of the type defined in the RETURNS clause. For an inline scalar function, there is no function body; the scalar value is the result of a single statement. For a multistatement scalar function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that return the single value. The return type can be any data type except **text**, **ntext**, **image**, **cursor**, and **timestamp.**

Table

- User-defined table-valued functions return a **table** data type. For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement.

System

- SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified.

User-defined Functions – Create Functions

- The general syntax is:

```
CREATE FUNCTION function_name  -- function name
( @Val1 int, @Val2 int )  -- parameter list
RETURNS int -- return type
AS
       BEGIN
       --Function body Sql statements  -- Function implementation process
       RETURN result;   -- return the final result
END
```

```
CREATE FUNCTION fn_SumTotalValues(
      @Val1 FLOAT ,
      @Val2 FLOAT
)RETURNS FLOAT
AS
    BEGIN
        RETURN @Val1 + @Val2;
    END
GO
```

```
SELECT [dbo].[fn_SumTotalValues](19.189, 29037.0978)
```

Function name

Parameters

	(No column name)
1	29056.2868

User-defined Functions – Create Functions

```
CREATE FUNCTION fn_GetEmployeeInfo(
      @EmployeeName VARCHAR(50)
)RETURNS TABLE
AS RETURN
    ( SELECT    E.ID ,
                E.Name ,
                S.Salary_IN_USD ,
                D.Dept ,
                D.[Direct Manager]
      FROM      dbo.Employee AS E
                LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
                LEFT JOIN dbo.Department AS D ON D.ID = E.ID
      WHERE     E.Name = @EmployeeName
    );
GO
```

```
SELECT * FROM [dbo].[fn_GetEmployeeInfo]('Eric')
```

User-defined Functions – Create Functions

```
CREATE FUNCTION fn_InsertEmployeeInfo(
      @EmployeeName VARCHAR(50)
)RETURNS @result TABLE (  自定义一个返回的Table表格 格式
      name NVARCHAR(20) ,
      age INT ,
      city NVARCHAR(20) ,
      salary FLOAT
)AS
    BEGIN
        INSERT INTO @result VALUES  ( @EmployeeName, 21, 'Roma', 1780.00 );
        INSERT INTO @result ( name, age , city, salary  )
                SELECT  E.Name ,
                        E.Age ,
                        E.City ,
                        S.Salary_IN_USD
                FROM    dbo.Employee AS E
                        LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
        RETURN;
    END
GO
```

	name	age	city	salary
1	Vivian	21	Roma	1780
2	Aleda	27	Dusseldorf	2239.19
3	Eric	29	New York	2850.97
4	Andy	23	Madrid	2600.16
5	Andrea	26	Roma	2131.91
6	Carlo	27	Milan	1850.36
7	Ann	25	Barcelona	3010.3
8	Alina	28	London	2100.27
9	Red...	30	Pairs	2727.19
10	Tom	32	Lyon	3109.17
11	Tim	33	Brussels	2200.2
12	Billy	36	Lisbon	2819

- ***Table [dbo.Employee] not changed***

User-defined Functions – Alter & Drop Functions

❖ *Alter Function*

```
Alter FUNCTION function_name
( @V11 int, @Val2 int )
RETURNS int
AS
       BEGIN
       --Function body Sql
statements
       RETURN result;
END
```

```
ALTER FUNCTION [dbo].[fn_GetEmployeeInfo]

      @EmployeeName VARCHAR(50)
    )
RETURNS TABLE
AS
        RETURN
    ( SELECT    E.ID ,
                E.Name ,
                S.Salary_IN_USD ,
                D.Dept
      FROM      dbo.Employee AS E
                LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
                LEFT JOIN dbo.Department AS D ON D.ID = E.ID
      WHERE     E.Name = @EmployeeName
    );
GO
```

❖ *Drop Function*

```
Drop Function function_name
```

```
DROP FUNCTION dbo.fn_GetEmployeeInfo
```

SQL VIEW – CREATE VIEW Statement

- A view is a virtual table 虚拟的结果表试图 based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
CREATE VIEW dbo.v_EmployeeInfo
AS
    SELECT   E.* ,
             S.Salary_IN_USD ,
             D.Dept
    FROM     dbo.Employee AS E
             LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
             LEFT JOIN dbo.Department AS D ON D.ID = E.ID
    WHERE    E.Age < 35
             AND S.Salary_IN_USD < 3500.00
Go
```

✓ *Then, we can query the view as follows:*

```
SELECT   *
FROM     dbo.v_EmployeeInfo
```

	ID	Name	Gender	Age	Birthday	Nationality	City	Salary_IN_USD	Dept
1	3116910	Aleda	Female	27	1990-09-23 00:00:00.000	Germany	Dusseldorf	2239.19	NTD
2	3116911	Eric	Male	29	1988-07-15 00:00:00.000	American	New York	2850.97	TMO
3	3116912	Andy	Male	23	1994-05-15 00:00:00.000	Spain	Madrid	2600.16	TMO
4	3116913	Andrea	Male	26	1991-07-15 00:00:00.000	Italy	Roma	2131.91	NTD
5	3116914	Carlo	Male	27	1990-04-19 00:00:00.000	Italy	Milan	1850.36	TMO
6	3116915	Ann	Female	25	1992-02-16 00:00:00.000	Spain	Barcelona	3010.3	TMO
7	3116916	Alina	Female	28	1989-09-21 00:00:00.000	UK	London	2100.27	TMO
8	3116917	Redwan	Male	30	1987-01-29 00:00:00.000	France	Pairs	2727.19	FBB Product Line
9	3116918	Tom	Male	32	1985-06-18 00:00:00.000	France	Lyon	3109.17	MBB Product Line
10	3116919	Tim	Male	33	1984-07-26 00:00:00.000	Belegium	Brussels	2200.2	TMO

SQL VIEW – Alter & Drop VIEW Statement

✓ *Alter view contents*

```
ALTER VIEW view_name
AS
        --other SQL statements
        select statement
Go
```

```
ALTER VIEW [dbo].[v_EmployeeInfo]
AS
    SELECT  E.* ,
            S.Salary_IN_USD ,
            D.Dept
    FROM    dbo.Employee AS E
            LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
            LEFT JOIN dbo.Department AS D ON D.ID = E.ID
    WHERE   E.Age < 30
            AND S.Salary_IN_USD < 3000.00

GO
```

✓ *Drop views*

```
Drop VIEW view_name
```

```
DROP VIEW [dbo].[v_EmployeeInfo]
```

SQL Stored Procedure – Create Procedure

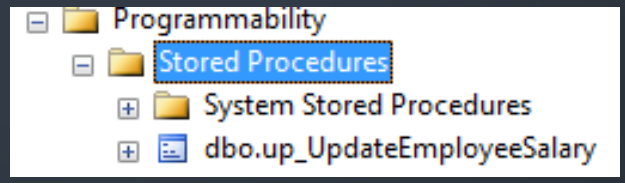
- A stored procedure is nothing more than ***prepared SQL code*** that you save so you can reuse the code over and over again. So if you think about a query that you write over and over again, instead of having to write that query each time you would save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.
- In addition to running the same SQL code over and over again you also have the ability to ***pass parameters*** to the stored procedure, so depending on what the need is the stored procedure can act accordingly based on the parameter values that were passed.

➢ *Creating a simple stored procedure*

```
CREATE PROCEDURE Procedure_Name
AS
    BEGIN
        -- SET NOCOUNT ON added to prevent
        -- extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

        -- Insert statements for procedure here
        SELECT …
        UPDATE …
          INSERT …
    END
GO
```

```
CREATE PROCEDURE dbo.up_UpdateSalary
AS
    BEGIN
        UPDATE  dbo.Salary
        SET     Salary_IN_USD = 2300.00
        WHERE   ID = 3116910
    END
GO
```



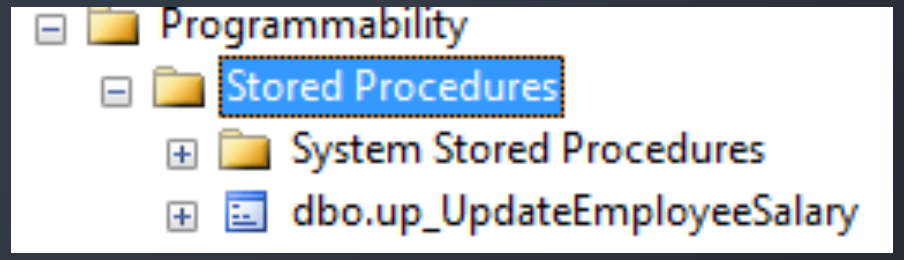
SQL Stored Procedure – Create Procedure with Parameters

➢ *Creating a stored procedure with parameters*

```
CREATE PROCEDURE <Procedure_Name>
(
-- Add the parameters for the stored procedure
here
@Param1, sysname,
@Param2, sysname
)
AS
    BEGIN
        -- SET NOCOUNT ON added to prevent
        -- extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

        -- Insert statements for procedure
here
        SELECT …
        UPDATE …
          INSERT …
    END
GO
```

```
CREATE PROCEDURE dbo.up_UpdateEmployeeSalary
    (
      @ID INT ,
      @newSalary FLOAT
    )
AS
    BEGIN
        UPDATE  dbo.Salary
        SET     Salary_IN_USD = @newSalary
        WHERE   ID = @ID
    END
GO
```



SQL Stored Procedure – Create Procedure Returning Parameters

➢ *Creating a stored procedure with returning(output)*

```
CREATE PROCEDURE <Procedure_Name>
(
-- Add the parameters for the stored procedure
here
@Param1, datatype,
@Param2, datatype OUTPUT
)
AS
    BEGIN
    -- SET NOCOUNT ON added to prevent
    -- extra result sets from
    -- interfering with SELECT statements.
         SET NOCOUNT ON;
    -- Insert statements for procedure here
         UPDATE …
         INSERT …
         SELECT …
    END
GO
```

```
CREATE PROCEDURE
dbo.up_Get_Employee_Count_By_Nationality
    @Nationality VARCHAR(30) ,
    @Count INT OUTPUT
AS
    BEGIN
        SELECT  @Count = COUNT(*)
        FROM    dbo.Employee
        WHERE   Nationality = @Nationality

        PRINT @Count;
    END
```

SQL Stored Procedure – Execute Procedure

- *The general syntax is:*

```
EXECUTE procedure_name
```

```
EXECUTE dbo.up_UpdateSalary
```

Or

```
EXEC procedure_name
```

```
EXEC dbo.up_UpdateSalary
```

Or

```
EXEC procedure_name <param1, param2>
```

```
EXEC dbo.up_UpdateEmployeeSalary 3116910, 2300.00
```

SQL Stored Procedure – Alter & Drop Procedure

Alter procedure: the general syntax

```
ALTER PROCEDURE procedure_name
     (
       @ID INT ,
       @newSalary FLOAT
     )
AS
     BEGIN
        --SQL statements
     END
```

```
ALTER PROCEDURE [dbo].
[up_UpdateEmployeeSalary]
     (
       @ID INT ,
       @newSalary FLOAT
     )
AS
     BEGIN
          UPDATE  dbo.Salary
          SET     Salary_IN_USD = @newSalary
          WHERE   ID = @ID
     END
```

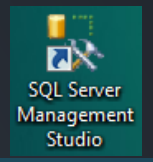
Drop procedure: the general syntax

```
Drop Procedure procedure_name
```

```
DROP PROCEDURE [dbo].[up_UpdateEmployeeSalary]
```

SQL Server Management Studio Tutorial

➢ *Start → All Programs → MS SQL Server 2012 → SQL Server Management Studio*



SQL Tutorial for Beginner-Leature-3.sql - G00207466-PA.SQL_LEARNING (CHINA\g00207466 (54)) - Microsoft SQL Server Management Studio (Administrator)

File Edit View Project Debug SQL Prompt 5 Tools Window Help

New Query | duration | Run SQL(F5)

SQL_LEARNING | Execute Debug | Add-ins SQL Search

Current working Database

Object Explorer

Connect

G00207466-PA (SQL Server 11.0.2218 - CHINA\g00207 — database Server

- Databases
  - System Databases
  - Database Snapshots
  - DXC
  - FR.SFR.DCCAP.KPI
  - License
  - ReportServer
  - ReportServerTempDB
  - SQL_LEARNING — Database
    - Database Diagrams
    - Tables
      - System Tables
      - FileTables
      - dbo.Consultant
      - dbo.Department
      - dbo.Employee
      - dbo.Salary
      - dbo.tmp_Employee
    - Views
      - System Views
      - dbo.v_EmployeeInfo
    - Synonyms
    - Programmability
      - Stored Procedures
        - System Stored Procedures
        - dbo.up_UpdateEmployeeSalary
      - Functions
        - Table-valued Functions
        - Scalar-valued Functions
          - dbo.fn_SumTotalValues
        - Aggregate Functions
        - System Functions
      - Database Triggers
      - Assemblies
      - Types
      - Rules

Database tree

Tutorial for Be...INA\g00207466 (54))

SQL editor area

```
22  --================================
23  GO
24  CREATE FUNCTION fn_GetEmployeeInfo
25      (
26        @EmployeeName VARCHAR(50)
27      )
28  RETURNS TABLE
29  AS
30            RETURN
31      ( SELECT      E.ID ,
32                    E.Name ,
33                    S.Salary_IN_USD ,
34                    D.Dept ,
35                    D.[Direct Manager]
36        FROM        dbo.Employee AS E
37                    LEFT JOIN dbo.Salary AS S ON S.ID = E.ID
38                    LEFT JOIN dbo.Department AS D ON D.ID = E.ID
39        WHERE       E.Name = @EmployeeName
40      );
41  GO
42
43
44  SELECT   *
45  FROM     [dbo].[fn_GetEmployeeInfo]('Eric')
46  --================================
```

100 %

Messages

2

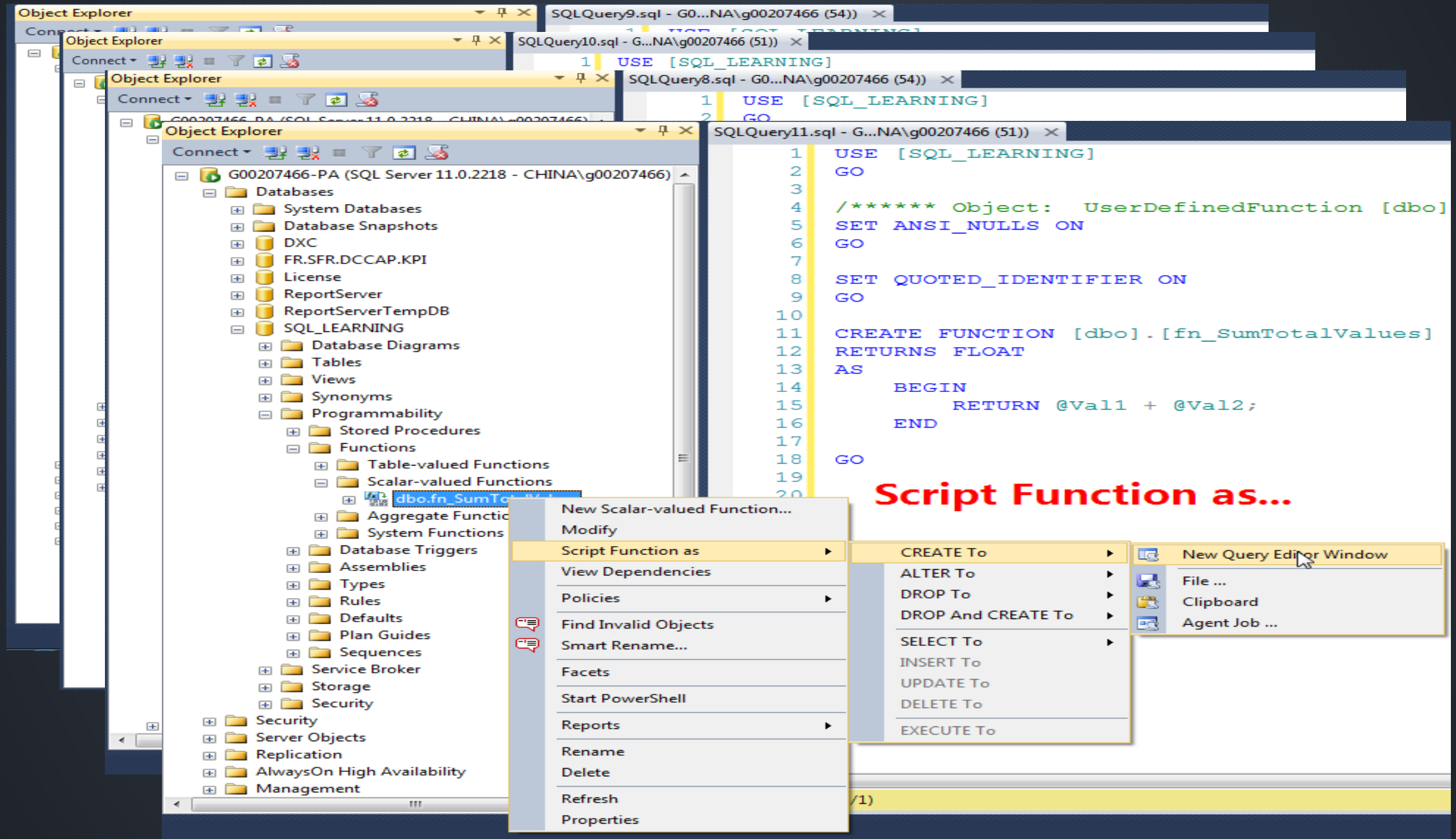
output area

100 %

Query executed successfully.

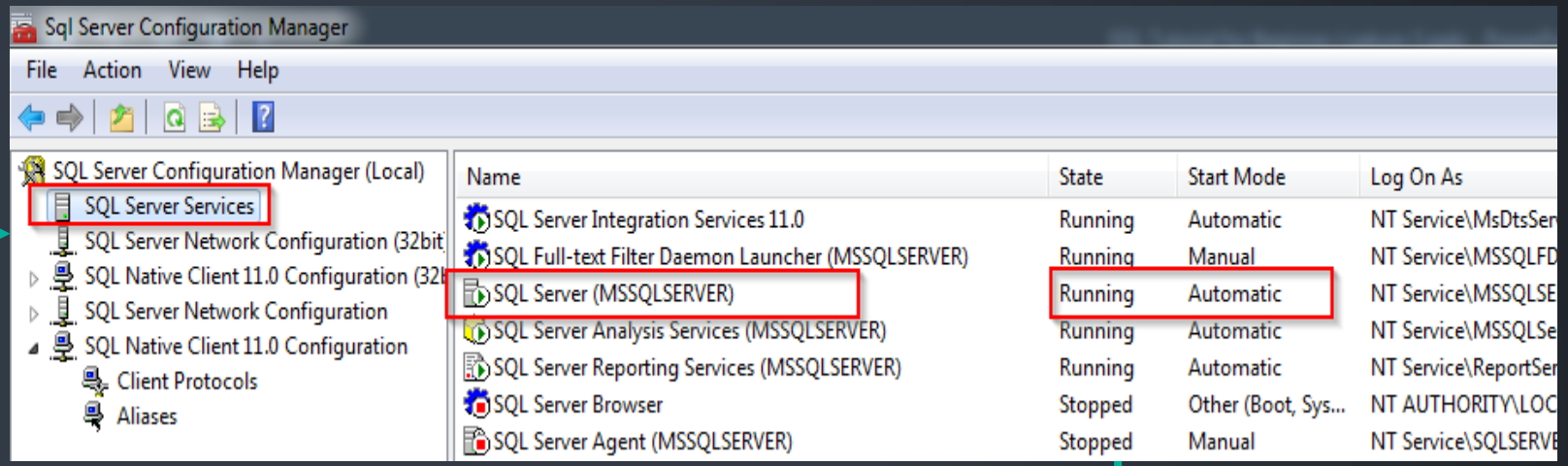
Ready

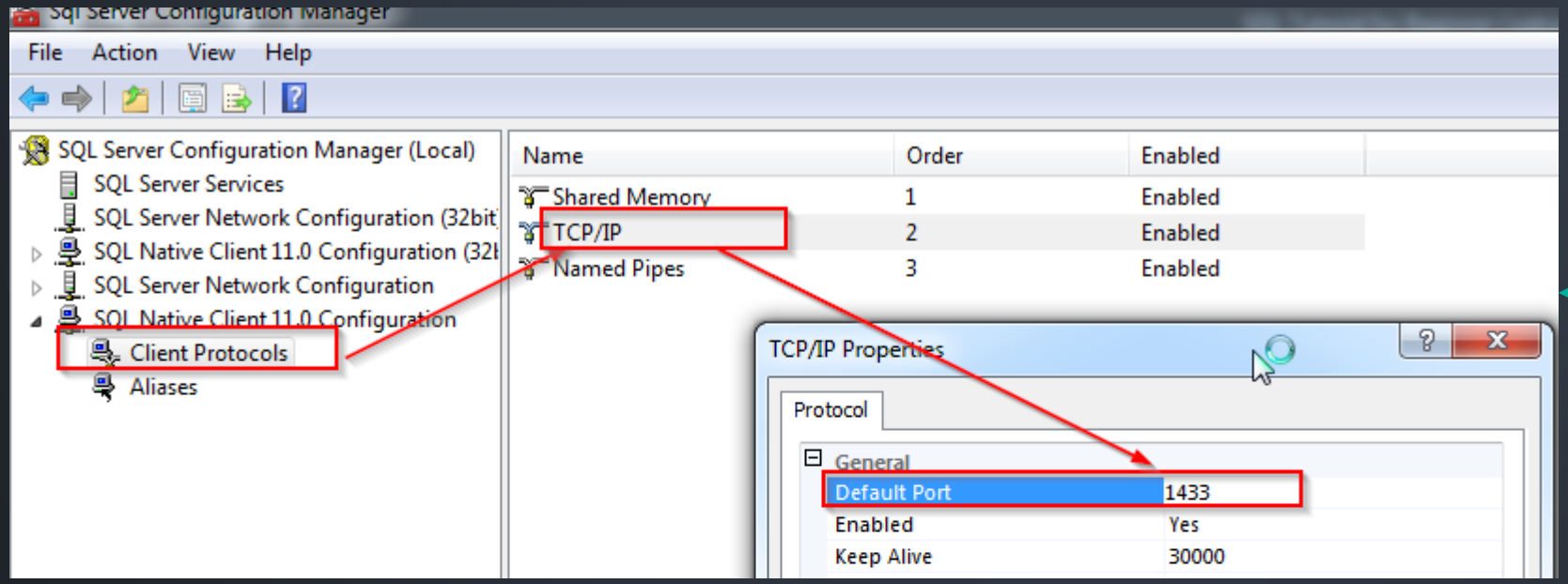
SQL Server Management Studio – Script object as



SQL Server Services

➢ *SQL Server Instance Service*





➢ *Remote access port…*