TIPURI DE DATE SQL SERVER (completare curs)

Întregi

Tip de date	**Lungime**	**Domeniu de valori**
Tinyint	1	0-255
Smallint	2	+/- 32767
Int	4	+/- 2.147.483.657
Bigint	8	+/- 2^63

Tipuri de date numerice(exacte, aproximatuive)

a) Date numerice exacte

Se specifică:
p = precizia (nr. cifre partea intreaga + zecimala);
s = scara (nr. cifre partea zecimala).

Obs.: - daca nu sunt specificate, SQL Sv foloseste p=18 si s=0;
- daca se stocheaza un numar care depaseste p si s, numarul se va trunchia.

b) Date numerice aproximative

- datele de tip *real* au o precizie de 7 cifre (nr. cifre total), ocupand 4 octeti;
- pentru tipul *float*, precizia e de 1-38 cifre; prestabilit, are 15 cifre.

Obs.: - daca declarati un *float* si specificati o precizie < 7, se creaza un *real*.

Tipuri de date monetare

Tip de date	**Lungime**	**Domeniu de valori**
Money	8	+/- 922.337.203.685.447,5808
smallmoney	4	+/- 214.748,3647

Obs.: - ambele tipuri au 4 zecimale;
- la introducerea de valori monetare se pune $ in fata nr.

Tipuri de date caracter

a) Lungime fixa char(n)

- ocupa n octeti;
- daca dimensiunea datelor e mai mica, restul de octeti e completat cu spatii;
- daca se stocheaza un sir cu lung. >n, datele sunt trunchiate.

b) Lungime variabila varchar(*n* | max)

- ocupa maxim n octeti; valoarea lui n este intre1 si 8000;
- nu se completeaza cu spatii;
- folositi **max** pentru a indica o dimensiune coloana de 2^31-1 octeti (2 GB);
- sporesc eficienta de stocare, micsoreaza performanta.

c) Text si image

- se folosesc cand sirurile depasesc 8000 caractere pe coloana;
- se numesc BLOB;

- pot stoca maxim 2 GB de date binare sau text;
- cand se declara un tip de date *text* sau *image*, la randul respectiv din tabel se adauga un pointer de 16 octeti, indicand o pagina separata de 8 KB unde se pastreaza informatii despre date; daca datele depasesc o pagina de 8 KB, se construiesc pointeri pt. alte pagini ale valorii BLOB;
- pot reduce performanta bazei de date, pt. ca in log se scriu cantitati mari de date la inserare, modificare şi stergere (cu c-da WRITETEXT nu se mai trec modificarile prin fis. de log);
- se recomanda pastrarea ca fisiere separate si sa se stocheze calea spre acestea in baza de date**.**

Tipuri de date Unicode

Datele Unicode folosesc setul de caractere Unicode UCS, un set de caractere pe mai multi octeti. Pt. caracterele ANSII normale e necesar un octet pt. orice caracter (se mai numeste set de caractere "ingust"). Unicode se mai cheama si set de caractere "lat"(multioctet). Unicode UCS-2 utilizeaza 2 octeti pt. reprezentarea unui caracter - e util pt. bazele de date in care se folosesc mai multe limbi (internationale). Un octet poate reprezenta 256 caractere, iar Unicode 65536 (se folosesc doi octeti/caracter-2^16).

- tipurile de date *nchar*, *nvarchar* şi *ntext* se utilizeaza pt. informatii Unicode:
- *nchar* şi *nvarchar* au limita maxima de 4000 caractere(8000 octeti)-o pagina de date;
- *ntext* accepta maximum 2.14 GB.

Obs. Pentru SQL Sv puteţi substitui numărul de caractere cu cuvântul cheie MAX, de ex. VARCHAR(MAX).Un tip de dată VARCHAR(MAX) sau NVARCHAR(MAX) vă permite să stocaţi până la 2 gigabytes (GB) de date.

Tipuri de date binare

Se stocheaza sub forma unor siruri sub forma de cifre de 1 si 0, care la intrare si iesire sunt reprezentate sub forma de perechi hexa(0-9 si A-F). Declaratia *binary(20)* semnifica 20 de octeti de date binare.

- *binary(n)* şi *varbinary(n)* pot contine max. 8000 octeti;
- pt. *binary* completarea se face cu spatii(0x20), nu si pt. *varbinary*;
- daca se depaseste lung. max., datele vor fi trunchiate;
- pt. a introduce date intr-un camp binar, se preced cu 0x(ex.: 0x10).

Tipuri de date Global Identifier (RowGUID)

Se utilizeaza la replicare – fiecare coloana din tabelul replicat trebuie sa aiba un nume unic; pt. aceasta se va crea in tabelul replicat un camp de tipul *uniqueidentifier*. Acest tip de date are o proprietate numita ROWGUIDCOL –cand e activata, coloanei i se atribuie un identificator global unic (GUID – Global Unique Identifier). Daca se modifica un rand dintr-un tabel replicat cu un atribut ROWGUIDCOL, SQL Server modifica acest atribut; randurile replicate din doua baze de date diferite pot fi urmarite separat. Acest tip de date ocupa 16 octeti (este un tip de date binar).

- atributele GUID pot fi initializate in doua moduri:
    - cu functia NEWID;

      Ex.:

      ```
      DECLARE @UNI UNIQUEIDENTIFIER
      SET @UNI = NEWID()
      SELECT @UNI
      ```

- convertind o c-ta sir in una hex de forma: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

<u>Date de tip special</u>

a) **<u>bit</u>**

Stocheaza informatii booleene (logice). Specifica aspecte gen: activ/inactiv, adevarat/fals, da/nu. Valoarea stocata a unui asemenea tip e 0 sau 1. Coloanele cu acest tip nu pot fi indexate, dar pot avea valori NULL (necunoscute). Ocupa 1 octet.

- daca sunt mai multe coloane de tip bit intr-un tabel, SQL Sv grupeaza max. 8 campuri bit intr-un octet.

b) **<u>Cursor</u>**

A data type for variables or stored procedure OUTPUT parameters that contain a reference to a cursor. Any variables created with the **cursor** data type are nullable.
The operations that can reference variables and parameters having a **cursor** data type are:

- The DECLARE *@local_variable* and SET *@local_variable* statements.
- The OPEN, FETCH, CLOSE, and DEALLOCATE cursor statements.
- Stored procedure output parameters.
- The CURSOR_STATUS function.
- The **sp_cursor_list**, **sp_describe_cursor**, **sp_describe_cursor_tables**, and **sp_describe_cursor_columns** system stored procedures.

Important: The **cursor** data type cannot be used for a column in a CREATE TABLE statement.

c) **<u>uniqueidentifier</u>(vezi RowGUID)**

d) **<u>timestamp</u>**

Tipul timestamp (amprenta de timp) permite adaugarea unei valori temporale cand se insereaza o inreg. intr-un tabel cu un camp timestamp. Caracteristici:

- se stocheaza ca binary(8) pt. coloanele NOT NULL si ca varbinary(8) daca se permite NULL;
- seamana cu tipul *datetime*, dar nu e acelasi lucru;
- valoarea e generata de SQL server;
- la actualizarea unui rand, aceste valori se actualizeaza automat;
- reprezinta un contor a carui valoare creste continuu;
- se poate folosi pt. a tine evidenta ordinii in care se adauga sau se modifica valori de campuri in tabel;
- Valoarea timestamp e bazata pe un ceas intern si nu corespunde timpului real. Fiecare tabel poate avea o singura coloana de acest tip.

e) **<u>table</u>**

Se foloseste pt. stocarea seturilor de rezultate utilizate mai tarziu. Utilizarea e similara cu crearea unui tabel temporar.

- se poate folosi ca o variabila locala sau ca un tabel obisnuit in functii, loturi(batches) si proceduri stocate.

Ex.:

```
DECLARE @LOCAL_TABLEVARIABLE TABLE
(column_1 DATATYPE,
 column_2 DATATYPE,
 column_N DATATYPE
 )
```

f) **sql_variant**

Acest tip de data e similar cu tipul variant din Visual Basic. Permite stocarea oricarui tip de date SQL Server, exceptand *ntext*, *timestamp* si *sql_variant*.

- se pot folosi in coloane, ca variabile intr-o instr. DECLARE si ca parametri sau rezultate in functii definite de utilizator.

Ex.:

```
DECLARE @v1 sql_variant;
SET @v1 = 'ABC';
SELECT @v1;
```

f) sysname

sysname is a system-supplied user-defined data type that is functionally equivalent to **nvarchar(128)** and is used to reference database object names.

Tipuri de date calendaristice

Tip de date	**Lungime**	**Domeniu de valori**
Datetime	8	1/1/1753-12/31/9999
smalldatetime	4	1/1/1900-6/6/2079

***XML* Data Type**

The *XML* data type allows you to store and manipulate Extensible Markup Language (XML) documents natively. When storing XML documents, you are limited to a maximum of 2 GB, as well as a maximum of 128 levels within a document. Although you could store an XML document in a character column, the *XML* data type natively understands the structure of XML data and the meaning of XML tags within the document.

Because the *XML* data type natively understands an XML structure, you can apply additional validation to the XML column, which restricts the documents that can be stored based on one or more XML schemas.

XML schemas are stored within SQL Server in a structure called a *schema collection.* Schema collections can contain one or more XML schemas. When a schema collection is applied to an XML column, the only documents allowed to be stored within the XML column must first validate to the associated XML schema collection.

The following command creates an XML schema collection:

```
CREATE XML SCHEMA COLLECTION ProductAttributes AS
'<xsd:schema xmlns:schema="PowerTools"
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:sqltypes=http://schemas.microsoft.com/sqlserver/2004/sqltypes
targetNamespace="PowerTools" elementFormDefault="qualified">
<xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes"
schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd" />
<xsd:element name="dbo.PowerTools">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Category">
<xsd:simpleType>
    <xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033"
sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType
IgnoreWidth" sqltypes:sqlSortId="52">
<xsd:maxLength value="30" />
```

```
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Amperage">
<xsd:simpleType>
<xsd:restriction base="sqltypes:decimal">
<xsd:totalDigits value="3" />
<xsd:fractionDigits value="1" />
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Voltage">
<xsd:simpleType>
<xsd:restriction base="sqltypes:char" sqltypes:localeId="1033"
sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType
IgnoreWidth" sqltypes:sqlSortId="52">
<xsd:maxLength value="7" />
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>'
```

Spatial Data Types

SQL Server supports two data types to store spatial data: *GEOMETRY* and *GEOGRAPHY*.
Both spatial data types are implemented by using the Common Language Runtime (CLR) capabilities that were introduced in SQL Server 2005.
Geometric data is based on Euclidian geometry and **is used to store points, lines, curves, and polygons**.
Geographic data is based on an ellipsoid and is used to store data such as **latitudes and longitudes**.
You define spatial columns in a table using either the *GEOMETRY* or *GEOGRAPHY* data types. When values are stored in a spatial column, you have to create an instance using one of several spatial functions specific to the type of data being stored.
A *GEOMETRY* column can contain one of seven different geometric objects with each coordinate in the definition separated by a space, as shown in Table 1.
GeometryCollection allows multiple shapes to be combined into a single column to represent a complex shape. When the object is instantiated by storing the object within a column defined as either *GEOMETRY* or *GEOGRAPHY*, the data and the definition of the object instance are stored within the column. Because the type of object and the coordinate data values are inseparable, it is possible to store multiple different types of objects in a single column.

TABLE 1 Geometry Data Type Definitions

INSTANCE	**DESCRIPTION**
Point	Has *x* and *y* coordinates, with optional elevation and measure values.
LineString	A series of points that defines the start, end, and any bends in the line, with optional elevation and measure values.
Polygon	A surface defined as a sequence of points that defines an exterior boundary, along with zero or more interior rings. A polygon has at least three distinct points.
GeometryCollection	Contains one or more instances of other geometry shapes, such as a *Point* and a *LineString.*
MultiPolygon	Contains the coordinates for multiple *Polygons.*
MultiLineString	Contains the coordinates of multiple *LineStrings.*
MultiPoint	Contains the coordinates of multiple *Points.*

Geographic data is stored as latitude and longitude points. The only restriction on geographic data is that the data and any comparisons cannot span a single hemisphere.

JSON Data Type

JSON is a popular textual data format that's used for exchanging data in modern web and mobile applications. JSON is also used for storing unstructured data in log files or NoSQL databases such as Microsoft Azure Cosmos DB. Many REST web services return results that are formatted as JSON text or accept data that's formatted as JSON.
Now you can combine classic relational columns with columns that contain documents formatted as JSON text in the same table, parse and import JSON documents in relational structures, or format relational data to JSON text.
If you must modify parts of JSON text, you can use the JSON_MODIFY (Transact-SQL) function to update the value of a property in a JSON string and return the updated JSON string.
The following example updates the value of a property in a variable that contains JSON:

```
DECLARE @json NVARCHAR(MAX);
SET @json = '{"info": {"adresa": [{"oras": "Bucuresti"}, {"oras": "Paris"}, {"town":"Madrid"}]}}';
SET @json = JSON_MODIFY(@json, '$.info.adresa[1].oras', 'Londra');
SELECT JsonModificat = @json;
```

Results:

```
JsonModificat
{"info": {"adresa": [{"oras": "Bucuresti"}, {"oras": "Londra"}, {"town":"Madrid"}]}}
```

Column Properties

The seven properties that you can apply to a column are: *nullability, COLLATE, IDENTITY, ROWGUIDCOL, FILESTREAM, NOT FOR REPLICATION* and *SPARSE.*

Nullability

You can specify whether a column allows nulls by specifying *NULL* or *NOT NULL* for the column properties.
Just as with every command you execute, you should always specify explicitly each option that you want, especially when you are creating objects. If you do not specify the *nullability* option, SQL Server uses the default option when creating a table, which could produce unexpected results. In addition, the default option is not guaranteed to be the same for each database because you can modify this by changing the *ANSI_NULL_DEFAULT* database property.

COLLATE

Collation sequences control the way characters in various languages are handled. When you install an instance of SQL Server, you specify the default collation sequence for the instance.
You can set the *COLLATE* property of a database to override the instance collation sequence, which SQL Server then applies as the default collation sequence for objects within the database.
Just as you can override the default collation sequence at a database level, you can also override the collation sequence for an entire table or an individual column.
By specifying the *COLLATE* option for a character-based column, you can set language-specific behavior for the column.

IDENTITY

Identities are used to provide a value for a column automatically when data is inserted.
You cannot update a column with the *identity* property. Columns with any numeric data type,except *float* and *real,* can accept an identity property because you also have to specify a seed value and an increment to be applied for each subsequently inserted row. You can have only a single identity column in a table.
Identity columns frequently are unique, but they do not have to be. To make an identity column unique, you must apply a constraint to the column.
Although SQL Server automatically provides the next value in the sequence, you can insert a value into an identity column explicitly by using the *SET IDENTITY_INSERT* <table name> *ON* command. You can also change the next value generated by modifying the seed using the *DBCC CHECKIDENT* command.

ROWGUIDCOL

The *ROWGUIDCOL* property is used mainly by merge replication to designate a column that is used to identify rows uniquely across databases. The *ROWGUIDCOL* property is used to ensure that only a single column of this type exists and that the column has a *UNIQUEIDENTIFIER* data type.

FILESTREAM

Databases are designed to store well-structured, discrete data. As the variety of data within an organization expands, organizations need to be able to consolidate data of all formats within a single storage architecture. SQL Server has the ability to store all the various data within an organization, the majority of which exist as documents, spreadsheets and other types of files.
Prior to SQL Server 2008, you had to extract the contents of a file to store it in a *VARBINARY(MAX), VARCHAR(MAX),* or *NVARCHAR(MAX)* data type. However, you were limited to storing only 2 GB of data within a large data type.
To work around this restriction, many organizations stored the filename within SQL Server and maintained the file on the operating system. The main issue with storing the file outside the database is that it was very easy to move, delete, or rename a file without making a corresponding update to the database.
SQL Server 2008 introduces a new property for a column called *FILESTREAM.*

Binary large objects (BLOBs) stored in a *FILESTREAM* column are controlled and maintained by SQL Server; however, the data resides in a file on the operating system.
By storing the data on the file system outside of the database, you are no longer restricted to the 2-GB limit on BLOBs.
In addition, when you back up the database, all the files are backed up at the same time, ensuring that the state of each file remains synchronized with the database.
You apply the *FILESTREAM* property to columns with a *VARBINARY(MAX)* data type. The column within the table maintains a 16-byte identifier for the file. SQL Server manages the access to the files stored on the operating system.

NOT FOR REPLICATION

The *NOT FOR REPLICATION* option is used for a column that is defined with the *IDENTITY* property.
When you define an identity, you specify the starting value, seed, and an increment to be applied to generate the next value.
If you explicitly insert a value into an identity column, SQL Server automatically reseeds the column.
If the table is participating in replication, you do not want to reseed the identity column each time data is synchronized.
By applying the *NOT FOR REPLICATION* option, SQL Server does not reseed the identity column when the replication engine is applying changes.

SPARSE

Designed to optimize storage space for columns with a large percentage of *NULLs*, the option to designate a column as sparse is new since SQL Server 2008.
To designate a column as *SPARSE*, the column must allow *NULLs*. When a *NULL* is stored in a column designated as *SPARSE*,no storage space is consumed. However, non-*NULL* values require 4 bytes of storage space in addition to the normal space consumed by the data type.
Unless you have a high enough percentage of rows containing a *NULL*, you should not designate a column as *SPARSE*.
You cannot apply the *SPARSE* property to:

- Columns with the *ROWGUIDCOL* or *IDENTITY* property;
- *TEXT, NTEXT, IMAGE, TIMESTAMP, GEOMETRY, GEOGRAPHY,* or user-defined data types;
- A *VARBINARY(MAX)* with the *FILESTREAM* property;
- A computed column of a column with a rule or default bound to it;
- Columns that are part of either a clustered index or a primary key;
- A column within an *ALTER TABLE* statement.

Computed Columns

Computed columns allow you to add to a table columns that, instead of being populated with data, are calculated based on other columns in the row.
When you create a computed column, only the definition of the calculation is stored. If you use the computed column within any data manipulation language (DML) statement, the value is calculated at the time of execution.
If you do not want to incur the overhead of making the calculation at runtime, you can specify the *PERSISTED* property.
If a computed column is *PERSISTED*, SQL Server stores the result of the calculation in the row and updates the value anytime data that the calculation relies upon is changed.

Row and Page Compression

SQL Server now allows you to compress rows and pages for tables that do not have a *SPARSE* column, as well as for indexes and indexed views.
Row-level compression allows you to compress individual rows to fit more rows on a page, which in turn reduces the amount of storage space for the table because you don’t need to store as many pages on a disk. Because you can uncompress the data at any time and the uncompress operation must always succeed, you cannot use compression to store more than 8,060 bytes in a single row.
Page compression reduces only the amount of disk storage required because the entire page is compressed.
When SQL Server applies page compression to a heap (a table without a clustered index), it compresses only the pages that currently exist in the table. SQL Server compresses new data added to a heap only if you use the *BULK INSERT* or *INSERT INTO. . .WITH (TABLLOCK)* statements. Pages that are added to the table using either *BCP* or an *INSERT* that does not specify a table lock hint are not compressed. To compress any newly added, uncompressed pages, you need to execute an *ALTER TABLE. . .REBUILD* statement with the *PAGE* compression option.
The compression setting for a table does not pass to any nonclustered indexes or indexed views created against the table.
You need to specify compression for each nonclustered index or indexed view that you want to be compressed.
VARCHAR(MAX), NVARCHAR(MAX), and *VARBINARY(MAX)* store data in specialized structures outside the row.
In addition, *VARBINARY(MAX)* with the *FILESTREAM* option stores documents in a directory external to the database. Any data stored outside the row cannot be compressed.

EXERCIŢII

În aceste exerciţii creaţi o schemă pt. a stoca un set de tabele.De asemenea adăugaţi constrângeri şi configuraţi opţiunile de stocare pentru linii şi pagini într- o tabelă.

1. Executaţi codul următor pentru a crea schema test B.D. *AdventureWorks*:

```
USE AdventureWorks
GO
CREATE SCHEMA test AUTHORIZATION dbo
GO
```

2. Executaţi codul următor pentru a crea o tabelă cu o coloană *IDENTITY* şi alta *SPARSE:*

```
CREATE TABLE test.Customer
(CustomerID INT IDENTITY(1,1),
LastName VARCHAR(50) NOT NULL,
FirstName VARCHAR(50) NOT NULL,
CreditLine MONEY SPARSE NULL,
CreationDate DATE NOT NULL)
GO
```

3. Executaţi codul următor pentru a crea o tabelă cu o coloană calculată şi cu compresie la nivel de linie:

```
CREATE TABLE test.OrderHeader
(OrderID INT IDENTITY(1,1),
CustomerID INT NOT NULL,
OrderDate DATE NOT NULL,
OrderTime TIME NOT NULL,
SubTotal MONEY NOT NULL,
ShippingAmt MONEY NOT NULL,
OrderTotal AS (SubTotal + ShippingAmt))
WITH (DATA_COMPRESSION = ROW)
GO
```