ICA | INFORMATICA COMMUNICATIE ACADEMIE

Course Database Implementation

Constraints and Transact-SQL

HAN

Constraints

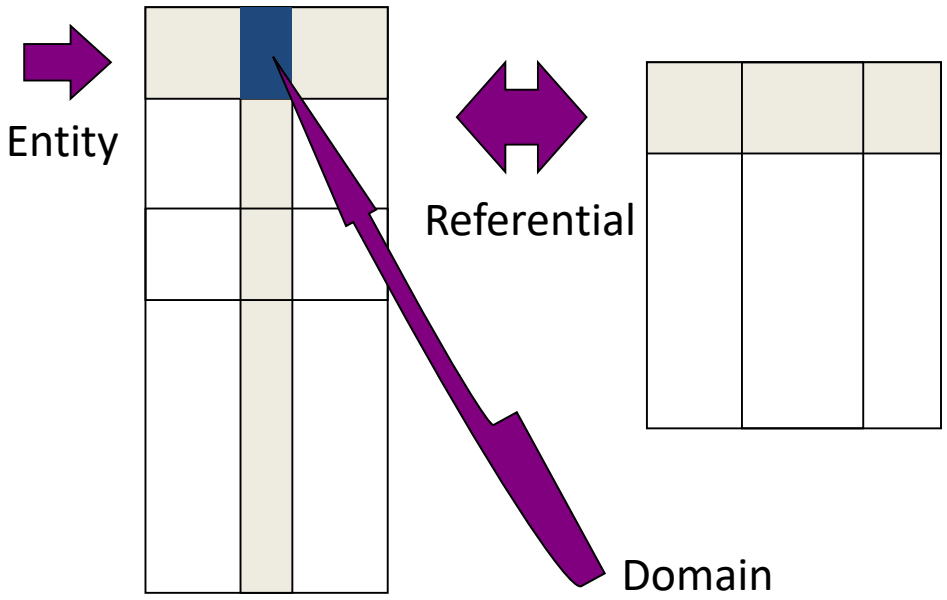
A constraint is a *restriction*
- Ensures data integrity

Constraint types:

Domain constraints
- On columns

Entity constraints
- On records as a whole

Referential integrity constraints
- On relationshiptypes *between* records



And constraints to realise complex business rules
– by database *programming*

Constraints – Example from Adventureworks

User defined data type

Domain

Entity

Domain

```
CREATE TYPE Flag FROM bit NOT NULL

CREATE TABLE Employee (
     BusinessEntityID   int           NOT NULL,
     JobTitle           nvarchar(50)  NOT NULL,
     BirthDate          date          NOT NULL,
     MaritalStatus      nchar(1)      NOT NULL,
     Gender             nchar(1)      NOT NULL,
     HireDate           date          NOT NULL,
     SalariedFlag       Flag          NOT NULL,
     ModifiedDate       datetime      NOT NULL,
     CONSTRAINT PK_Employee_BusinessEntityID PRIMARY KEY (BusinessEntityID)
)

ALTER TABLE Employee WITH CHECK
    ADD CONSTRAINT CK_Employee_BirthDate
              CHECK ((BirthDate >= '1930-01-01'
                     AND
                     BirthDate <= dateadd(year,(-18), getdate()))
                    )
```

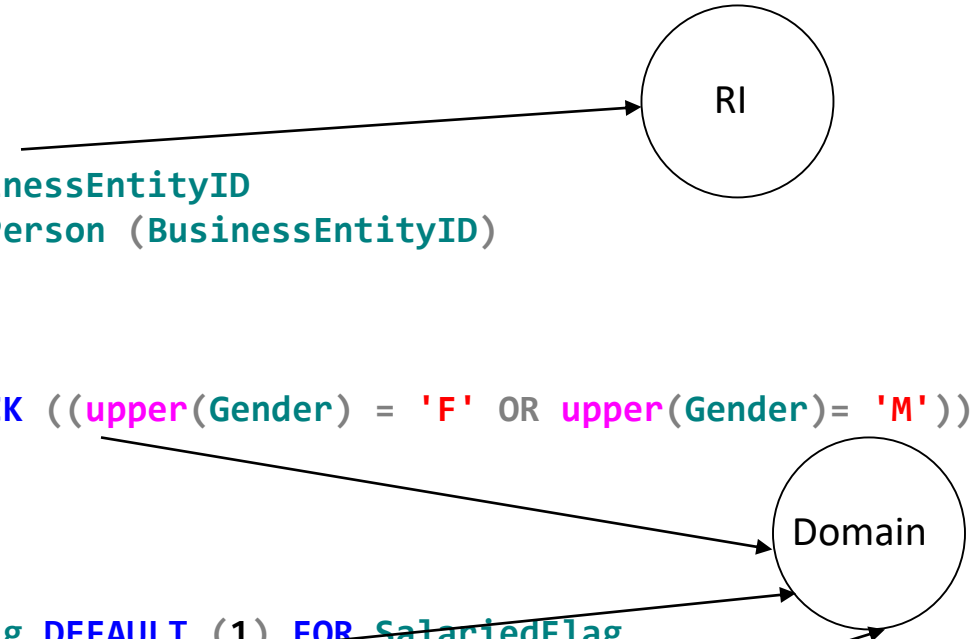
Constraints – Example from Adventureworks

```
ALTER TABLE Employee
    ADD CONSTRAINT FK_Employee_Person_BusinessEntityID
FOREIGN KEY(BusinessEntityID) REFERENCES Person (BusinessEntityID)


ALTER TABLE Employee WITH CHECK
    ADD CONSTRAINT CK_Employee_Gender CHECK ((upper(Gender) = 'F' OR upper(Gender)= 'M'))


ALTER TABLE Employee
    ADD CONSTRAINT DF_Employee_SalariedFlag DEFAULT (1) FOR SalariedFlag


ALTER TABLE Employee
    ADD CONSTRAINT DF_Employee_ModifiedDat DEFAULT (getdate()) FOR ModifiedDate
```



Entity and Domain Constraints

Entity Constraint

- requires every row to have a unique value for some column or combination of columns
- PRIMARY KEY (Implicit NOT NULL)
- UNIQUE (allows *one* NULL value)

Domain constraint

- requires that a particular column (or set of columns) meets some criteria
- data type
- required data (NULL / NOT NULL)
- CHECK
- DEFAULT

DEFAULT

Specifies which value must be inserted in a column when the column is not listed in the INSERT-statement

```
CREATE TABLE test_defaults (
  keycol         smallint,
  process_id     smallint     DEFAULT @@SPID,
  date_ins       datetime     DEFAULT GETDATE(),
  mathcol        smallint     DEFAULT 10 * 2,
  text1          char(3),
  text2          char(3)      DEFAULT 'xyz'
)

INSERT test_defaults (text1) VALUES ('qqq')

SELECT * FROM test_defaults
```

```
keycol process_id date_ins                mathcol text1 text2
------ ---------- ----------------------- ------- ----- -----
NULL   52         2016-02-03 14:06:55.417 20      qqq   xyz
```

Referential Integrity (RI) constraint

Requires that a value in one column matches the value in another column

- ‘child’ table refers to the ‘parent’ table
- in a different table or in the same table
  - maybe in another database
  - maybe on another server

RI constraint

Be aware that RI constraints can place restrictions on *both* tables

- **is it allowed to delete/update a parent record if that record is referenced from a child?**
- **if so, what happens with the child?**
  - **CASCADING DELETE/UPDATE?**
  - **or is the reference SET to NULL**
  - **or is the reference SET to a DEFAULT?**
  - **or is the DELETE/UPDATE RESTRICTED?**

Cardinality constraints

1:1, 1:N, N:M, 1-5, …

(Mandatory/Optional) NULL/NOT NULL

But how to enforce that a soccer team has no more than 11 players?

- Use a stored procedure or trigger….

Exercise

Workbook, Theme CONSTRAINTS

Exercises 1 and 2

Taxonomy of Constraints - 1

***static* constraints**

- rules which every record ***at any instance of time*** must obey
- ex: Domain constraints
  - Gender of a Person must be ‘M’ or ‘F’
  - Data types
    - *limits the possible values (numeric, character), but that is usually not sufficient!*

***dynamic* constraints**

- rules which ***restrict the transition*** between two instantiations of the database
  - ex: It is not allowed to update the marital status of a Person from ‘Not Married’ to ‘Divorced’

Taxonomy of Constraints - 2

***declarative* constraints**

- constraints are 'declared' by means of *DDL*- or *DCL*-statements
- specify WHAT has to be done
  - CHECK, NULL/NOT NULL, FOREIGN KEY, PRIMARY KEY, UNIQUE
- DBMS decides HOW

***procedural* constraints**

- developer writes *code* to enforce constraints
  - triggers, stored procedures, user defined functions
- developer decides HOW

Why procedural constraints?

use declarative constraints whenever possible!

- fast, correct

but there are many things that DDL-statements can't specify

- complex relationships
- state transitions
- vendor specific:
  - RI across databases
  - Cascading updates/deletes in some cases
  - ...

but coding is non standard

- MS SQL Server: Transact-SQL (T-SQL)

Transact-SQL

Database Server

Most servers provide SQL-89 level functionality

Most servers include some SQL-92 features

Quite a few servers offer proprietary versions of SQL3 stored procedures, triggers, and rules

SQL Extensions

Relational DBMS's often have built-in procedural extensions to standard SQL

- A procedural programming language
  - variables, if-statements, while-statements, ...
- Stored procedures
- Triggers
- Rules

Useful, but non-standard

Vendor-specific SQL is no longer pure declarative

- Declarative: specify WHAT has to be done
- Procedural: specify HOW it has to be done
- MS SQL Server, Sybase: TRANSACT-SQL (T-SQL)
- Oracle: PL/SQL

Transact-SQL

Native language of SQL Server

- Extension of SQL with programming constructs

T-SQL is simple and limited

- Local variables
- Control of flow: IF, CASE, WHILE
- Operators, special functions
- **No** user interface programming constructs
- **No** file I/O

Program execution within the database engine

Consult *Books Online***!**

ICA | INFORMATICA COMMUNICATIE ACADEMIE

Transact-SQL Syntax Conventions

Convention	Used for
UPPERCASE	Transact-SQL keywords.
italic	User-supplied parameters of Transact-SQL syntax.
bold	Database names, table names, column names, index names, stored procedures, utilities, data type names, and text that must be typed exactly as shown.
underline	Indicates the default value applied when the clause that contains the underlined value is omitted from the statement.
| (vertical bar)	Separates syntax items enclosed in brackets or braces. You can use only one of the items.
[ ] (brackets)	Optional syntax items. Do not type the brackets.
{ } (braces)	Required syntax items. Do not type the braces.
[,...*n*]	Indicates the preceding item can be repeated *n* number of times. The occurrences are separated by commas.
[...*n*]	Indicates the preceding item can be repeated *n* number of times. The occurrences are separated by blanks.
[;]	Optional Transact-SQL statement terminator. Do not type the brackets.
<label> ::=	The name for a block of syntax. This convention is used to group and label sections of lengthy syntax or a unit of syntax that can be used in more than one location within a statement. Each location in which the block of syntax can be used is indicated with the label enclosed in chevrons: <label>.

T-SQL CREATE TABLE

(simplified)

```
CREATE TABLE
[ database_name.[schema name].  | schema name.]  table_name
  (
    { <column_definition> | computed_column_definition> }
    [ <table_constraint>] [,...n]
    [;]
  )


<column_definition> ::= column_name <data_type>
    [NULL | NOT NULL]
    [
     [ CONSTRAINT constraint_name] DEFAULT
    constant_expression ]
     | [ IDENTITY [(seed, increment ) ] ]
    ]
    [ <column_constraint>] [ ...n]
```

T-SQL CREATE TABLE

(simplified)

```
<table_constraint> ::= [CONSTRAINT constraint_name]
{
    [ { PRIMARY KEY | UNIQUE }
     [CLUSTERED | NONCLUSTERED]
        { ( column[,...n] ) }
    ]

    | FOREIGN KEY
            [(column[,...n])]
            REFERENCES ref_table [(ref_column[,...n])]

    | CHECK  (search_conditions)
}
```

Note: Multicolumn key constraints are created as table constraints.

Multi-part names

Object reference format	Description
server . *database* . *schema* . *object*	Four-part name.
server . *database* .. *object*	Schema name is omitted.
server .. *schema* . *object*	Database name is omitted.
server ... *object*	Database and schema name are omitted.
database . *schema* . *object*	Server name is omitted.
database .. *object*	Server and schema name are omitted.
schema . *object*	Server and database name are omitted.
object	Server, database, and schema name are omitted.

For *Schema* study BOL, Transact-SQL Syntax Conventions, Multi-part Names

Script, example

```
USE SomeDatabase

DECLARE @Ident INT                    -- declaration of variable

INSERT INTO Orders (CustomerID, OrderDate)
    VALUES (25, DATEADD(DAY,-1,GETDATE())) -- system functions

SELECT @Ident = @@IDENTITY          /* variable assignment
                                     @@: system functions */

INSERT INTO Details (OrderID, ProductID, UnitPrice, Quantity)
    VALUES (@Ident, 1, 50, 25)     -- retrieval of variable

SELECT 'The OrderID of the inserted row is ' + -- operator +
       CAST(@Ident AS varchar(8))
```

Variables

Declaration:

```
DECLARE  @variable_name  datatype
```

Assignment of a value:

```
SELECT  @variable = expression
[FROM … [WHERE …]]
```

Or

```
SET @variable = expression
```

Retrieval of a value:

```
SELECT @variable
```

For use in a batch, stored procedure or trigger or udf

- A variable exists only for the life of the batch

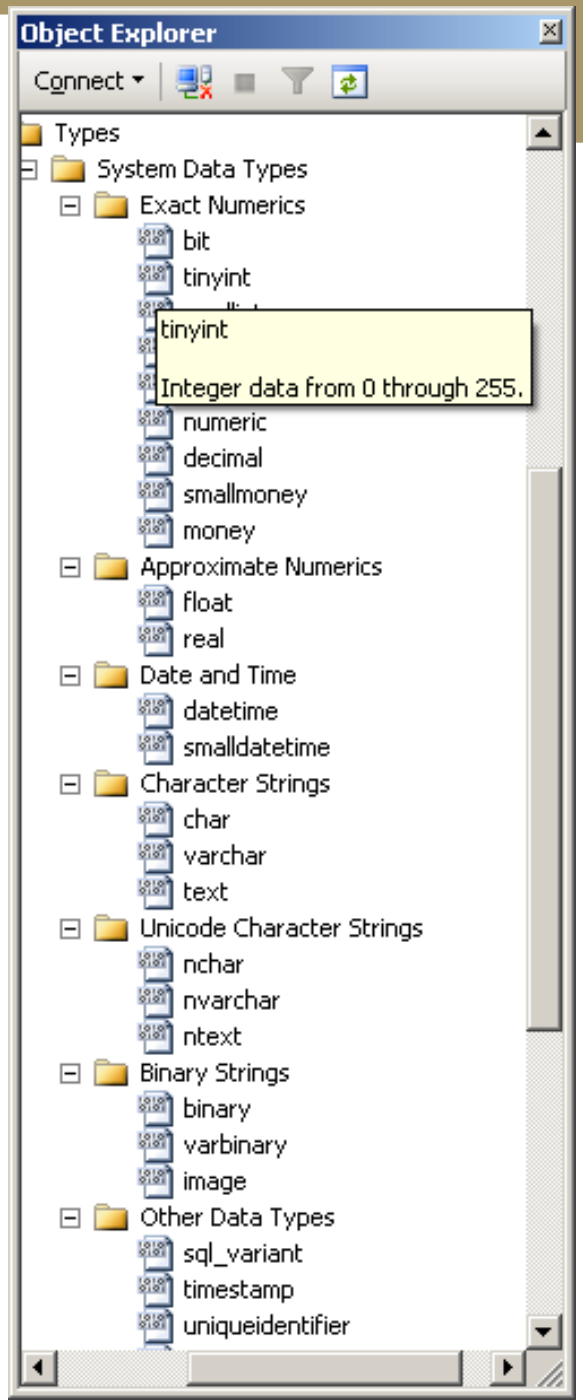
Arghh... Exercise

```
-- execute

DECLARE @composerName varchar(20)

SELECT @composerName = name
FROM Composer

SELECT @composerName
```



T-SQL Data types

Study:

- **Conversion table**
- **BOL**
- **Object Explorer**

HAN

Identity Property

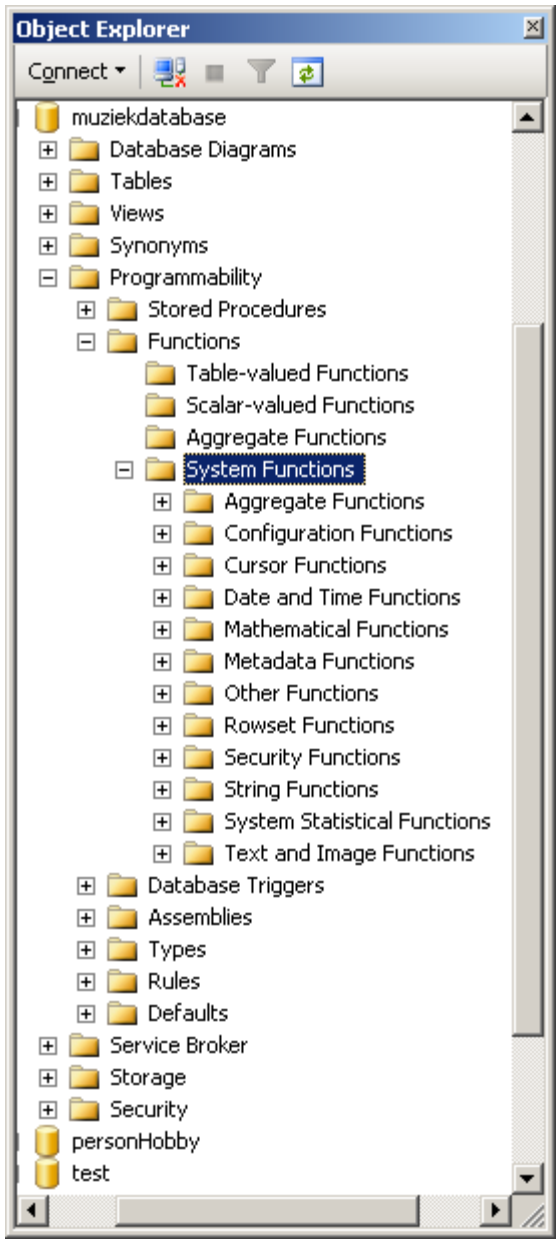
Creates an *identity column* in a table

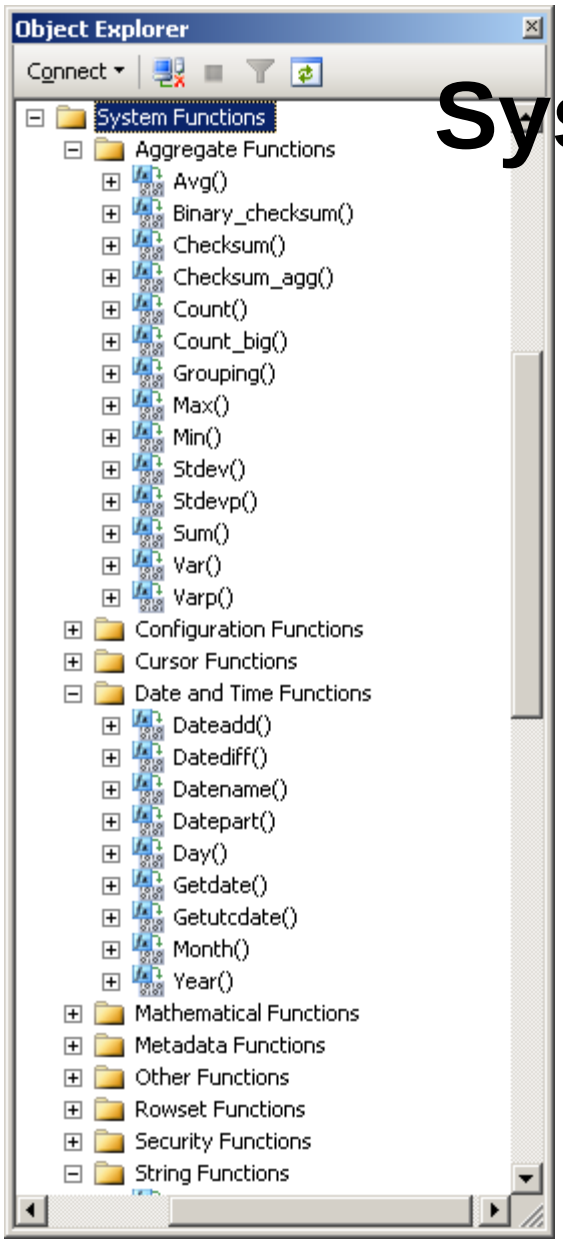
Syntax: IDENTITY [(*seed* , *increment*)]

```
CREATE TABLE new_employees (
  emp_id    INT PRIMARY KEY    IDENTITY(1,1),
  fname     VARCHAR(20),
  middle    CHAR(1),
  lname     VARCHAR(30)
)

INSERT new_employees (fname, middle, lname)
  VALUES ('Karin', 'F', 'Josephs')
```

System Functions





T-SQL Functions, Operators

Usage:

```
SELECT GETDATE()
SELECT COS(PI())
SELECT @my_pi = ROUND(PI(), 1)
```

Operators: `+, -, *, /, %, =, >, <, >=, <=, <>, ...`

Study Transact-SQL Help

Batch

A batch is a set of SQL-statements that is sent at one time to the server

The batch is by the server:

1. **Parsed as a whole**
   - The server checks the syntax of each statement.
2. **Optimised and compiled as a whole**
   - The server specifies the most efficiënt method to execute the queries
3. **Executed statement by statement**

Batch Example 1

A batch of statements of which one statement has a syntax error will not be executed

```
CREATE TABLE test
 (col1 INT PRIMARY KEY,
  col2 INT NOT NULL)
```

one batch

```
INSERT test (col1, col2) VALUES (6,10)
INSERT test (col1, col2) VLUES (6,51)
UPDATE test SET col2 = 4
SELECT * FROM test
```

```
Msg 102, Level 15, State 1, Line 2
Incorrect syntax near 'VLUES'.
```

Batch Example 2

If the server decides to execute the batch, it is of course possible that run-time errors occur

correct batch

```
INSERT test (col1, col2) VALUES (6,10)
INSERT test (col1, col2) VALUES (6,51)
UPDATE test SET col2 = 4
SELECT * FROM test
```

```
(1 row(s) affected)

Msg 2627, Level 14, State 1, Line 2
Violation of PRIMARY KEY constraint 'PK__test__0BC6C43E'.
Cannot insert duplicate key in object 'dbo.test'.
The statement has been terminated.

(1 row(s) affected)

col1        col2
----------- -----------
6           4
(1 row(s) affected)
```

T-SQL Local Variables

```
CREATE TABLE test2 (col INT NOT NULL)

INSERT test2 (col) VALUES (1)

DECLARE @maxcol INT
SELECT @maxcol = MAX(col) FROM test2
INSERT INTO test2 (col) VALUES (@maxcol + 1)
```

} execute as *one* batch

After execution of this batch:

- the variable **`@maxcol`** is no longer defined

GO is NOT an SQL-stmt

If you want to send *two* batches to the server, you will have to do so one by one

Another way is to use a separator that instructs the *CLIENT* to send two batches

- A lot of front-end tools use the keyword `go`

T-SQL Conditional Execution

```
IF Boolean_expression
   {Transact-sql_statement | statement_block}
[ELSE [Boolean_expression]
   {Transact-sql_statement | statement_block}]


statement_block:
   BEGIN
      Transact-SQL_statements
   END
```

T-SQL Conditional Execution

```
CREATE TABLE product
( prod_nr      int             NOT NULL
       CONSTRAINT pk_product PRIMARY KEY (prod_nr),
  name         varchar(30)     NOT NULL,
  price        money           NOT NULL,
  type         varchar(30)     NOT NULL
)

INSERT product (prod_nr, name, price, type)
    VALUES (1, 'tv', 500, 'electronics')

INSERT product (prod_nr, name, price, type)
    VALUES (2, 'radio', 100, 'electronics')

INSERT product (prod_nr, name, price, type)
    VALUES (3, 'ball', 100, 'sport')
```

Conditional Execution Exercise 1

```
IF EXISTS (SELECT 1 FROM product WHERE name = 'radio')
  BEGIN
        PRINT 'There are radio''s in stock'
  END
ELSE
  BEGIN
        PRINT 'No radio''s in stock'
  END
```

Conditional Execution Exercise 2

```
DECLARE @number INT
SET @number = 100
IF (SELECT AVG(price)
    FROM product WHERE type = 'electronics') > @number
 BEGIN
    PRINT 'The average price of electronic products is greater than '
           + CONVERT(varchar(10), @number)
 END
ELSE
 BEGIN
   RAISERROR ('The average price of electronic products
                  is less than %d.', 16, 1, @number)
 END
```

T-SQL Repeated Execution

```
WHILE Boolean_expression
   {Transact-sql_statement | statement_block}
```

```
 WHILE (SELECT AVG(price) FROM product) < 500
   BEGIN
     UPDATE product
     SET price = price * 1.05
   END
```

Exercise

Modulo11-check:

972428577 is a valid bankaccountnumber, because:

(9*9 + 8*7 + 7*2 + 6*4 + 5*2 + 4*8 + 3* 5 + 2*7 + 1*7) % 11 = 0

Make a script that checks this for a fixed accountnumber

CASE, with *input expression*

```
USE AdventureWorks
GO

SELECT TOP 10 SalesOrderID%10 AS 'OrderLastDigit',
       ProductID%10 AS 'ProductLastDigit',
       'How Close?' =
           CASE SalesOrderID%10    -- input expression
             WHEN ProductID%10     THEN 'Exact Match!'
             WHEN ProductID%10 - 1 THEN 'Within 1'
             WHEN ProductID%10 + 1 THEN 'Within 1'
             ELSE 'More Than One Apart'
           END
FROM Sales.SalesOrderDetail
ORDER BY SalesOrderID DESC
```

A searched CASE, without *input expression*

```
SELECT TOP 10 SalesOrderID%10 AS 'OrderLastDigit',
       ProductID%10 AS 'ProductLastDigit',
       'How Close?' =
       CASE
       WHEN (SalesOrderID%10) < 3 THEN 'Ends with less than three'
       WHEN ProductID = 6     THEN 'ProductID is 6'
       WHEN ABS(SalesOrderID%10 - ProductID) <= 1 THEN 'Within 1'
       ELSE 'More Than One Apart'
       END
FROM Sales.SalesOrderDetail
ORDER BY SalesOrderID DESC
```

Evaluates to a Boolean

Exercise

Use a searched CASE in a script that gives the name of the current day translated in Finnish.