

Data-driven systems

Programming Microsoft SQL Server



Automatizálási és
Alkalmazott
Informatikai Tanszék

Contents

- Microsoft SQL Server platform
- Server-side programming of databases
- Transact-SQL language
 - > Cursors
 - > Stored procedures and functions
 - > Error handling
 - > Triggers

Relational databases

Database definition

- Organized collection of logically related data
- Data: known facts recorded digitally
 - > Basic: text, number, date
 - > Multimedia: images, sounds, video
 - > Structured: XML, JSON
- Organized collection: easy to find
- Related data: required for an application

Relation (table) properties

record / row {


ColumnID	ColumnA	ColumnB	ColumnC
Key1	Scalar value		
Key2			
Key3			
Key4			

}
column / attribute

Integrity requirements

OszlopID	ColumnA	ColumnB	ColumnC
Key1	123		
Key2	456		
Key3	789	Rec999	

XXX	YYY
Rec998	
Rec999	



1. Entity integrity: key \neq NULL
2. Domain integrity: column data type and domain
3. Referential integrity: foreign key

Microsoft SQL Server platform

Server components

- SQL Server Service
- SQL Server Browser
- SQL Server Agent
- SQL Server Analysis Services
- SQL Server Reporting Services
- SQL Server Integration Services
- SQL Writer

Databases

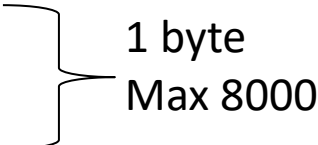
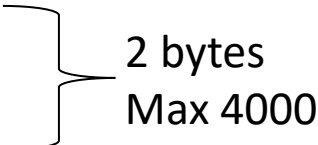
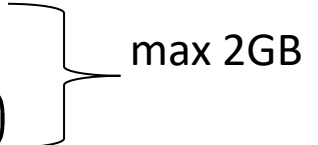
- Database
 - > Data file (.mdf)
 - Can be filegroup
 - > Transaction log (.ldf)
 - > Can contain multiple schemas
 - Default schema: dbo
 - Logical separation, access control
- Built-in databases
 - > Master, Model, Distribution, MSDB, Temp

User schema

- User schema = objects in the database
- Table
 - > Column
 - > Computed Column
 - Virtual
 - Stored
- View
 - > Indexable → Stored
- Index
- Sequence
- Programmability
 - > Procedure
 - > Method
 - > Trigger
 - > Assembly

Data types

- Text

- > Char(n) 
- > Varchar(n)
- > Nchar(n) 
- > Nvarchar(n)
- > Varchar(max) 
- > Nvarchar(max)

- Large objects

- > Image
- > TEXT / ntext
- > VARBINARY

- Numeric

- > Int
- > Float
- > Numeric(p,s)

- Date

- > Datetime
 - From Jan. 1st 1753 (!)
- > Datetime2

- Other

- > XML
- > Hierarchyid
- > Geographic data

Sidenote

- Where do we store images for a web application?
- Database?
- File system?
- External / cloud service?
- Performance, security, backup/restore, scaling, maintainability, simplicity, ...

Generating values for primary key

- Identity keyword

- > create table Status(
 ID int identity(1,1) primary key,
 Name nvarchar(20))
 - > Insert into Status values ('Ready')

- Query

- > ident_current('Status')
 - Any session, any scope, specific table
 - > @@IDENTITY
 - Current session, any scope, any table
 - > SCOPE_IDENTITY ()
 - Current session, current scope, any table

Scope vs session

- Scope
 - > Structural unit
 - > Procedure, trigger, function, batch
 - > They are in the same scope if, for example, they are in one procedure
- Session
 - > Logical database connection
 - > Either from a terminal window or programmatically
 - > A logical flow of message exchanges
 - > Typically a 1:1 physical connection
 - MARS – one physical connection, multiple sessions

Creating a new record and query the key

```
create table VAT  
(  
    ID int identity primary key,  
    Percentage int  
)
```

```
insert into VAT(Percentage)  
values (27)
```

```
select @@identity
```

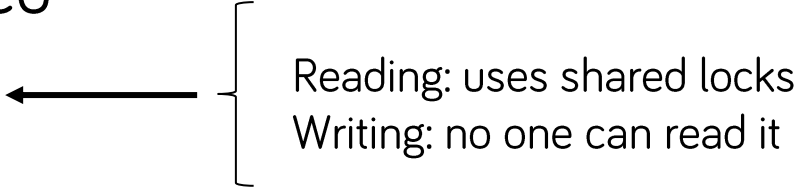
@@IDENTITY vs SCOPE_IDENTITY

- For example, a trigger runs on table T1, which creates a new row on T2 when a row is inserted
- After inserting a new row into T1:
 - > @@IDENTITY: Returns the ID of a row inserted in table T2
 - > SCOPE_IDENTITY: returns the ID of the new row added to T1

Transaction boundary

- Specific to a connection
- Starting a transaction (depends on configuration)
 - > Auto commit: All statements are independent transactions
 - This is the default mode
 - > Explicit transactions: Begin tran, for multiple statements
 - Can be nested
 - @@TRANCOUNT variable
 - > Implicit transactions: Automatically starts a transaction after the end of the previous one
 - If there is no current transaction, certain statements automatically begin one
- DML and DDL statements can be part of transactions too
 - > Few exceptions: Create Database, Backup, Restore

Supported isolation levels

- Isolation levels in the SQL standard
 - > Read uncommitted
 - > Read committed ← 
 - > Repeatable read
 - > Serializable
- Not in the standard
 - > Snapshot
 - Snapshot when the transaction starts
 - Row-level versioning

Access control

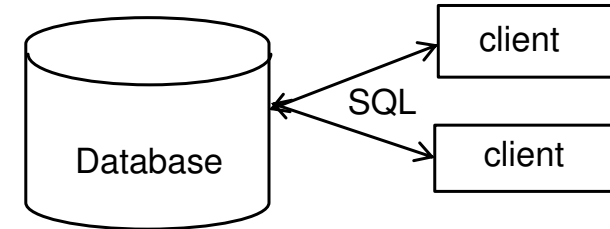
- System-level
- Database-level
- Schema-level
- Object-level
 - > Specific to an object
 - > Can be column specific in case of tables and views
- Allow and deny permissions
- No row-level access control.
 - > Some level of support since MSSQL2014

Database server-side programming



Role of the database

- Data model: tables
- Data manipulation: SQL language
- Tasks, that are outside the scope of the relational model
 - > Business Layer
 - > Data Layer → **Server-side programming**



Server-side programming: motivation

- Example: Registering for an exam in Neptun
 - > Exams table: ID, Course, Date, Limit
 - > ExamRegistration table: ExamId, StudentId
- Pseudo-code:

```
func registerForExam(examid, studentId) {  
    registered_cnt = count examregistration ID==examid  
    if registered_cnt < exam.limit  
        insert examregistration examid, studentid  
    else error  
}
```

Server-side programming: motivation

- Example: Registering for an exam in Neptun
- With T-SQL using a stored procedure

```
create procedure examregistration
@examid int, @studentid int
as
begin
    declare @registered_cnt int =
        select count(*) from examregistration where ID=@examid
    declare @examlimit int
    select @examlimit = limit from exam where ID=@examid
    if @registered_cnt < @examlimit
        insert into examregistration values(@examid, @studentid)
    else
        throw 51005, 'exam limit reached'
end
```

Server-side programming: motivation

- Example: Registering for an exam in Neptun
- With T-SQL using a stored procedure

```
create procedure examregistration
@examid int, @studentid int
as
begin
  declare @registered_cnt int =
    select count(*) from examregistration where ID=@examid

  declare @examlimit int
  select @examlimit = limit from exam where ID=@examid

  if @registerd_cnt < @examlimit
    insert into examregistration values(@examid, @studentid)
  else
    throw 51005, 'exam limit reached'
end
```

Procedural code

Stored procedure ~

Parameters

Local variable

Branching (control flow)

Errors

Advantages of server-side programming - 1

- Database is responsible for consistency
 - > Role of the database changes
 - Data source + Services
- Security
 - > Data modification through public "interface"
 - E.g., this is the stored procedure
 - > Closed execution environment
 - Data does not need to get outside of the database

Advantages of server-side programming - 2

- Increase performance
 - > Reduce network traffic
 - > Database caching
- Productivity
 - > The code is called by multiple components
 - > Easier maintenance -> need to fix in the DB

Disadvantages of server-side programming

- Not standardized
 - > Platform specific languages
 - > Platform specific solutions
- Interpreted
- Increases the load of the server
- Not possible or hard to scale

Transact-SQL language



T-SQL language

- Transact-SQL / T-SQL
- Language of Microsoft SQL Server
- Instructions according to the SQL language
 - + variables
 - + statement blocks
 - + cycles
 - + structured error handling
 - + new language constructs

T-SQL language

- SQL language review
- MSSQL specific syntax
- Server-side programming

> See details in the lecture notes



T-SQL nyelv

```
DECLARE @c int = 0
SELECT @c = COUNT(*) FROM Product
WHILE @c < 1000
BEGIN
    INSERT INTO Product VALUES ('Alma')
    SET @c = @c + 1
END
```

T-SQL nyelv

```
DECLARE @cid int = 123
```

```
DECLARE @name nvarchar(max)
```

```
SELECT @name=Name FROM Customer WHERE ID=@cid
```

```
IF @name IS NOT NULL
```

```
BEGIN
```

```
    PRINT 'Email cím frissítése'
```

```
    UPDATE Customer SET Email='...' WHERE ID=@cid
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
    PRINT 'Nincs ilyen vevo'
```

```
END
```


T-SQL functions

- Isnull(kif1,kif2)
- CASE *input_expression*
WHEN *when_expression* THEN
result_expression [...*n*] [ELSE
else_result_expression]
END
- CAST (*expression* AS *data_type*
[(*length*)])
- CONVERT (*data_type* [(*length*)
], *expression* [, *style*])
 - > *style* → conversion, formatting
- Try_cast / Try_convert
- Date
 - > Getdate()
 - > DateAdd(datepart , number, date)
 - > Month(date)
 - > Year(date)
 - > ...
- String
 - > Replace
 - > Trim
 - > Ltrim
 - > Rtrim
 - > Len
 - > Format
 - > ...

Some examples

```
-- Részstring csere  
SELECT REPLACE('Happy Birthday!', 'day', 'month')  
-- Happy Birthmonth!
```

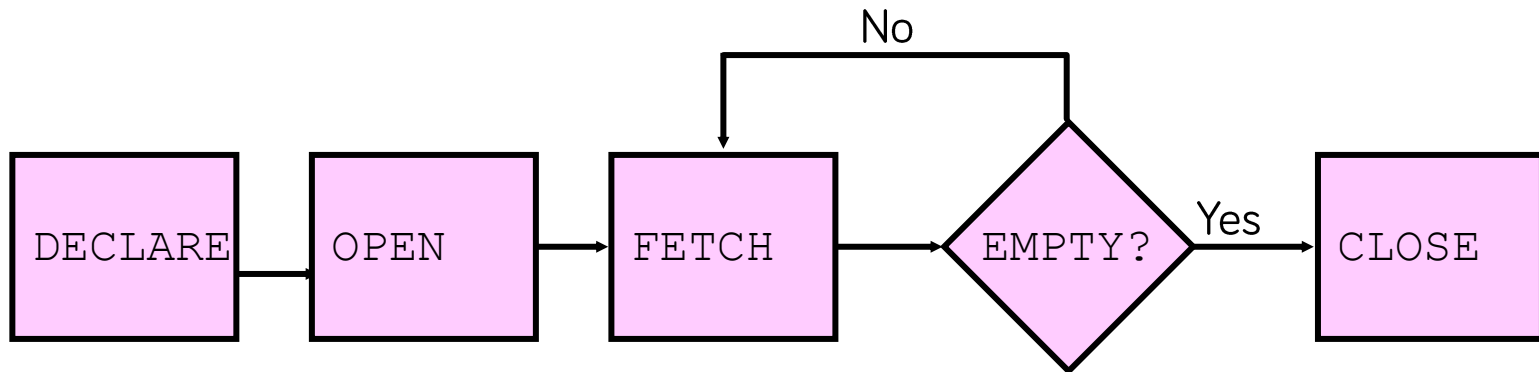
```
-- Dátumok közötti különbség adott mértékegységben (itt: nap) mérve  
SELECT DATEDIFF(day, '2021-09-28 12:10:09', '2021-11-04 13:45:09')  
-- 37
```

```
SELECT CONVERT(int, '12')  
-- 12
```

```
DECLARE @a int  
DECLARE @b int = 5  
SELECT ISNULL(@a, @b)  
-- 5
```

How the cursor works

- „Foreach iterator”
- Process record sets
- Enumerate a record set
 - > Open
 - > Process the record
 - > Fetch next record



Cursor: declaration

```
DECLARE cur_name CURSOR  
[ FORWARD_ONLY | SCROLL ]  
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]  
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
FOR query_string  
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

Cursor: opening and navigating

- OPEN
OPEN cur_name
- FETCH
FETCH
 [[NEXT | PRIOR | FIRST | LAST
 | ABSOLUTE { n | @nvar }
 | RELATIVE { n | @nvar }
]
 FROM
]
cursor_name [INTO @variable_name [,...n]]
- @@FETCH_STATUS
 - > Returns the status of the last FETCH
 - 0 – FETCH succeeded
 - -1 – FETCH was unsuccessful
 - -2 – row cannot be found

Cursor: closing and releasing

- CLOSE
 - > Release the record set
 - > Release locks
 - > Data structure is kept, can be opened later

CLOSE cur_name
- DEALLOCATE
 - > Decrements the reference counter of the cursor
 - > If all references are removed, the structure is disposed

DEALLOCATE cur_name

Cursor sample

```
DECLARE products_cur CURSOR SCROLL SCROLL_LOCKS
FOR
    SELECT Id, Name FROM Product WHERE Stock < 3
FOR UPDATE OF Price -- Szeretnénk frissíteni is
```

```
-- Tipikus megnyitás, fetch, ciklus
OPEN products_cur
FETCH NEXT FROM products_cur INTO @ProductID, @ProductName
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
-- Következő rekord lekérdezése, majd ugrás a WHILE ciklushoz
    FETCH NEXT FROM products_cur INTO @ProductID, @ProductName
END
```

```
-- Kurzor használatának befejezése
CLOSE products_cur
DEALLOCATE products_cur
```

Stored procedures and functions

Stored procedure

- Motivation: write commonly used logic
- Object-oriented solution: function/method
- Stored procedure: a code stored in the database that we can call from T-SQL code
 - Procedure: typically does not have return value (but can have)
- Stored function: has return value, but can only read the database (cannot write)

Stored procedure types

- Internal
 - > T-SQL language
 - > Interpreted
 - > Closed environment
- External: .NET Assembly
 - > Tasks that fall out of the scope of the database

Stored procedures

- Creation

- > CREATE PROCEDURE

```
CREATE OR ALTER PROC [EDURE] proc_name  
    [ { @param data_type }  
    ] [ ,...n ]
```

```
AS sql_statements [ ...n ]
```

- Modify

- > ALTER PROCEDURE

- The whole new procedure body is specified.

- Delete

- > DROP PROCEDURE

Stored procedure sample

```
create or alter procedure InsertNewVAT -- tárolt eljárás létrehozása, neve
    @Percentage int                    -- tárolt eljárás paraméterei
as
begin
    -- innen kezdődik a kód, amit az eljárás meghívásakor végrehajt a rendszer
    begin tran                        -- nem megismételhető olvasás elkerülése
    set transaction isolation level repeatable read

    declare @Count int

    select @Count = count(*)
    from VAT
    where Percentage = @Percentage

    if @Count = 0
        insert into VAT values (@Percentage)
    else
        print 'error';

commit
end
```

```
exec InsertNewVAT 27
```

Stored function types

- Scalar-valued
 - > Calculates a single scalar result
- Table-valued
 - > Returns with a result set
 - > Can be used as a table in queries
 - > Can collect records using a complex logic (abstraction)
- Aggregate
 - > Custom aggregation functions
 - > .NET Assembly

Function sample

```
CREATE OR ALTER FUNCTION LargestVATPercentage()  
RETURNS int  
BEGIN  
RETURN (SELECT MAX(Percentage) FROM VAT)  
END
```

```
select dbo.LargestVATPercentage()  
-- A dbo előtag a séma azonosítása, amivel azt jelöljük, hogy ez nem egy beépített  
-- Enélkül a függvényt nem találja meg a rendszer  
  
-- avagy például  
DECLARE @maxvat int = dbo.LargestVATPercentage()  
select @maxvat
```

Error handling

Error handling

- @@Error function
 - > Can be queried after each statement
 - > If there are no errors, value is 0
 - > sys.messages table contains the error codes

- Structured error handling

BEGIN TRY

Statements

END TRY

BEGIN CATCH

Statements

END CATCH

```
USE AdventureWorks2012;
GO
UPDATE HumanResources.EmployeePayHistory
    SET PayFrequency = 4
    WHERE BusinessEntityID = 1;
IF @@ERROR = 547
    BEGIN
        PRINT N'A check constraint violation occurred.';
    END
GO
```


Raising errors - Throw

- Since SQL Server 2012
 - > Earlier: RaisError – don't use it!
- THROW error_number, message, state
- Peculiarities
 - > Error is not string-based
 - FORMATMESSAGE
 - > Can be thrown again
 - Throw in the catch block
 - > error_number > 50000

Error handling sample

```
create or alter procedure InsertNewVAT
    @Percentage int
as
begin

    begin tran
    set transaction isolation level repeatable read

    declare @Count int

    select @Count = count(*)
    from VAT
    where Percentage = @Percentage

    if @Count = 0
        insert into VAT values (@Percentage)
    else
        throw 51000, 'error', 1;

    commit
end
```

Triggers

Trigger: motivation

- Make sure that any deletion is logged (audit logging)
 - > Solution 1: delete from DB then log in the business logic
 - Problem: can be forgotten, can be circumvented
 - > Solution 2: "subscribe" to the deletion event in the database
 - Trigger
- The order should have a cumulative total value
 - > Can be calculated from the order items, but let us save this as a separate value (denormalization)
 - > If any item of the order changes, the value must be updated

Create DML triggers

```
CREATE TRIGGER trigger_name  
ON { Table | View }  
FOR { [ DELETE ] [, ] [ INSERT ]  
      [, ] [ UPDATE ] }  
AS  
sql_statement [ ...n ]
```

Triggers

- Event handler procedures
- Maintenance of derived values
 - > Denormalization
- Logging
- Collect statistics
- For managing special referential integrity

Events

- **DML events**
 - > Insert, update, delete
 - > For tables
- **DDL trigger**
 - > Create, alter, drop, ...
 - > For schema
- System events
 - > Logon, logoff, SysError, ...
- Instead of triggers
 - > Special DML triggers
 - > Updating views

Trigger types

- DML triggers
 - > Executed after statements
 - > Works on a table
 - > Executed after data modification
 - Executed within the same transaction
- Instead of trigger
 - > Executed instead of the original statement
 - > Updating views
 - > Valid for tables too, works as a trigger before data modification
 - Can reference the table itself within the trigger, causes no recursion

Accessing modified records

- Through log tables
 - > Structure is the same as the table that is being used
 - > Table is accessible only within the trigger
 - > Specific to a session

	Insert	Delete	Update
Inserted	New records	Empty	New values of the records
Deleted	Empty	Deleted records	Previous values of the records

Trigger sample

- The AuditLog table has one field: Description
- Insert the text formatted with the name of the deleted product from the special "deleted" table

```
-- Napló tábla létrehozása
create table AuditLog([Description] [nvarchar](max) NULL)
go

-- Naplózó trigger
create or alter trigger ProductDeleteLog
on Product
for delete
as
insert into AuditLog(Description)
select 'Product deleted: ' + convert(nvarchar, d.Name) from deleted d
```

Worth to know: batch

Batch

```
USE AdventureWorks2012;
```

```
GO
```

```
DECLARE @MyMsg VARCHAR(50)
```

```
SELECT @MyMsg = 'Hello, World.'
```

```
GO
```

```
PRINT @MyMsg
```

- Collection of statements sent to the server at once
 - > *Not part of the T-SQL language*