Școala
informală
de IT

SQL

Școala Informală de IT

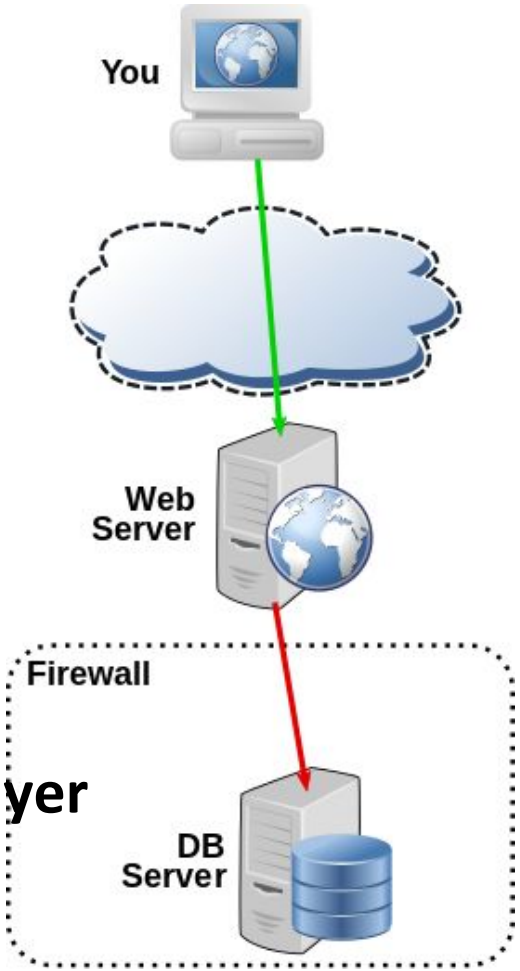
Cluj-Napoca

Agenda

- Structured Query Language (SQL)
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Transactional Control Language (TCL)

Database

- Multilayer architecture
- Data access layer
- Database



Data access layer

File system Vs Database

File system

- Low Cost
- Data recovery
- No training
- No security

Database

- Control redundancy
- Security
- Multiple user access
- Integrity constraints
- Backup and recover

Database

- **Database entities**
  - Table: collection of rows
  - Row (or record): actual data
  - Field (or cell): data member
  - Column: name of the fields

Table Name

Record

Column

Products

ProductId	Name	Brand	Price	Quantity
1	Galaxy S6	Samsung	3400	50
2	Galaxy Alpha	Samsung	1500	10
3	iPhone 6S	Apple	3600	25

Field

Școala
informală
de IT

Structured Query Language

SQL

SQL

- **Structured Query Language (SQL)**
  - **Declarative language for query and manipulation of relational data**
- **SQL consists of:**
  - **Data Manipulation Language (DML)**
    - **SELECT, INSERT, UPDATE, DELETE**
  - **Data Definition Language (DDL)**
    - **CREATE, DROP, ALTER**
  - **Transactional Control Language (TCL)**
    - **COMMIT, ROLLBACK**

SQL Commands

- SQL commands are executed through a database connection
  - DB connection is a channel between the client and the SQL server
  - DB connections take resources and should be closed when no longer used
  - Multiple clients can be connected to the SQL server at the same time
  - SQL commands can be executed in parallel
    - Transactions and isolation deal with concurrency

SQL and CRUD

- **CRUD for databases**

Operation	Data manipulation	Data definition
Create	INSERT	CREATE
Read (Retrieve)	SELECT	-
Update (Modify)	UPDATE	ALTER
Delete (Destroy)	DELETE	DROP

Data Definition Language (DDL)

Data Types

- **Numeric**
  - `bit` (1-bit), `integer` (32-bit), `bigint` (64-bit)
  - `float, real, numeric(scale, precision)`
  - `money` – for money (precise) operations
- **Strings**
  - `char(size)` – fixed size string
  - `varchar(size)` – variable size string
  - `nvarchar(size)` – Unicode variable size string
  - `text / ntext` – text data block (unlimited size)

Data Types

- **Binary data**
  - `varbinary(size)` – a sequence of bits
  - `image` – a binary block up to 1 GB
- **Date and time**
  - `datetime` – date and time
  - `smalldatetime` – date and time (1-minute precision)
- **Other types**
  - `timestamp` – automatically generated datarow change
  - `uniqueidentifier` – GUID identifier

Data Types

- `Nullable` and `NOT NULL` types
  - All types in SQL Server may or may not allow `NULL` values
- `Primary key` columns
  - Define the primary key
- `Identity` columns
  - Automatically increased values by increment when a new row is inserted (auto-increment values) starting from identity seed
  - Used in combination with `primary key`

Primary Key

- Always define an additional column for `primary key`
  - Don't use an existing column (for example SSN)
  - Must be an integer number
  - Must be declared as a `primary key`
  - Use `identity` to implement auto-increment
  - Put the `primary key` as a first column
- Exceptions
  - Entities that have well known ID, e.g. countries (BG, DE, US) and currencies (USD, EUR, BGN)

Data Definition Language

- DDL commands for defining / editing objects
  - CREATE
    - CREATE DATABASE, CREATE TABLE, CREATE INDEX
  - ALTER
    - ALTER TABLE
  - DROP
    - DROP TABLE, DROP INDEX

CREATE

- **CREATE command**
  - CREATE TABLE <name> (<field_definitions>)
  - CREATE VIEW <name> AS <select>
  - CREATE <object> <definition>

```
CREATE TABLE Products(
  ProductID int AUTO_INCREMENT,
  Name nvarchar(100) NOT NULL,
  PRIMARY KEY(ProductID)
);

CREATE VIEW [First 10 Products] AS
SELECT TOP 10 Name FROM Products
```

ALTER

- **ALTER command**
  - ALTER TABLE <name> <command>
  - ALTER <object> <command>

```
-- Add column Country to the table Brands
ALTER TABLE Brands ADD COLUMN Country int

-- Remove column Country from the table Brands
ALTER TABLE Brands DROP COLUMN Country
```

DROP

- **DROP command**
  - DROP TABLE <name>
  - DROP TRIGGER <name>
  - DROP INDEX <name>
  - DROP <object>

Data Manipulation Language (DML)

SQL Examples

```
SELECT Name, Price, Description FROM Products
```

```
SELECT * FROM Brands
WHERE Brand = 'Samsung'
```

```
INSERT INTO Products (Name, Price)
VALUES('Galaxy Alpha', 1600)
```

```
UPDATE Products
SET Warranty = 12 WHERE BrandID = 3
```

```
DELETE FROM Products
WHERE BuyDate = '1/1/2006'
```

Școala informală de IT

SQL SELECT

Projection

Take some of the columns

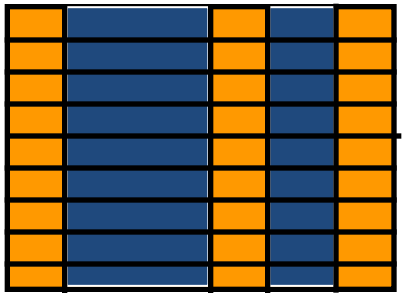


Table 1

Selection

Take some of the rows

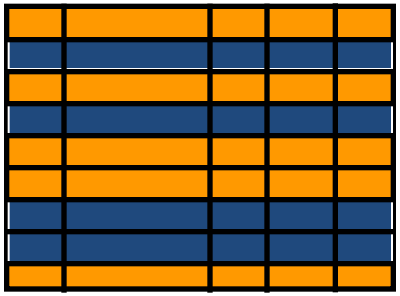


Table 1

Join

Combine tables by some column

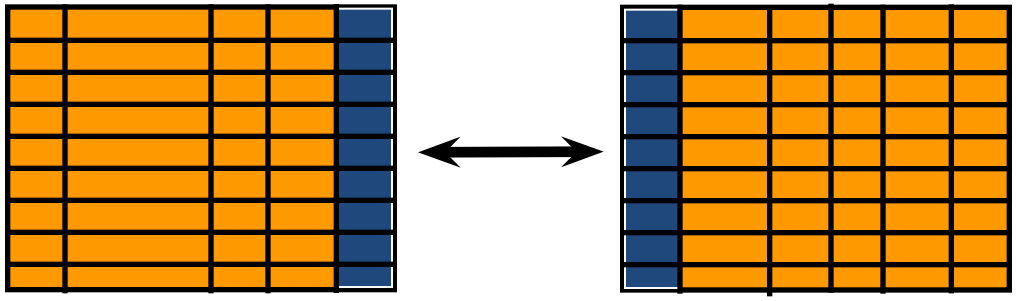


Table 1

Table 2

INSERT

- **INSERT command**
  - INSERT INTO <table> VALUES (<values>)
  - INSERT INTO <table>(<columns>) VALUES (<values>)
  - INSERT INTO <table> SELECT <values>

```
INSERT INTO Promotions
    VALUES (101, 'Xperia Z1', 2200)

INSERT INTO Promotions (ProductID, Name, Price)
    VALUES (101, 'Xperia Z1', 2200)

INSERT INTO Promotions (ProductID, Name, Price)
    SELECT ProductID, Name, 2200 FROM Products
```

SELECT

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table WHERE clause
```

- **SELECT** identifies what columns
- **FROM** identifies which table
- **DISTINCT** removes duplicates
- **WHERE** defines what clause

SELECT Example

- **Selecting all columns from brands**

```
SELECT * FROM Products
```

ProductID	Name	BrandID
1	Galaxy Alpha	12
2	iPhone 6S	4
3	HTC ONE	273

- **Selecting specific columns**

```
SELECT
   ProductID,
   Name
FROM Products
```

ProductID	Name
1	Galaxy Alpha
2	iPhone 6S
3	HTC ONE

Arithmetic Operations

- **Arithmetic operators are available: +, -, *, /**
- **Examples:**

```
SELECT (7 + 3) * 2
```

```
SELECT Name, Price, Price * 0.2 FROM Products
```

Name	Price	(No column name)
Galaxy Alpha	1600,00	320,00
iPhone 6S	3500,00	700,00
HTC ONE	2000,00	400,00

NULL Value

- A NULL is a value that is unavailable, unassigned, unknown, or inapplicable
  - Not the same as zero or a blank space
- Arithmetic expressions containing a NULL value are evaluated to NULL

```
SELECT Name, ProductID FROM Products
```

Name	ProductID
Galaxy Alpha	NULL
iPhone 6S	300
HTC ONE	1

NULL is displayed as empty space or as NULL

Column Aliases

- Aliases rename a column heading
- Useful with calculations
- Immediately follows the column name
  - There is an optional AS keyword
- Double quotation marks if contains spaces

```
SELECT ProductID, Name, Price,
Price * 0.2 AS VAT FROM Products
```

ProductID	Name	Price	VAT
433	Galaxy Alpha	1600.00	320.00
221	iPhone 6S	3500.00	700.00

Concatenation Operator

- Concatenates columns or strings to other columns
- Is represented by plus sign “+”
- Creates a resultant column that is a character expression

```
SELECT Name + ' / ' + Brand AS [Full Name],
ProductID as [No.] FROM Products
```

Full Name	No.
Galaxy Alpha / Samsung	134
iPhone 6S / Apple	253
HTC ONE / HTC	321

Literals

- A literal is a character, a number, or a date included in the SELECT list
- Date and character literal values must be enclosed within single quotation marks

```
SELECT Name + '''s brand is ' +
Brand AS [Our Products] FROM Products
```

Our Products
Galaxy Alpha's brand is Samsung
iPhone 6S's brand is Apple
HTC ONE's brand is HTC

Removing Duplicates

- The default display of queries is all rows, including duplicate rows

```
SELECT Price
FROM Products
```

ProductID
2500
3000
2500
…

- Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause

```
SELECT DISTINCT Price
FROM Products
```

ProductID
2500
3000
…

UNION, INTERSECT, MINUS

- **UNION combines the results from several SELECT statements**
  - **The columns count and types should match**

```
SELECT Name AS ProductName
FROM Products
UNION
SELECT Name AS ProductName
FROM Promotions
```

ProductName
Galaxy Alpha
iPhone 6S
HTC ONE
…

- **INTERSECT / EXCEPT perform logical intersection / difference between given two sets of records**

Limit Selected Rows

- **Restrict the rows returned by using the WHERE clause:**

```
SELECT Name, Brand
FROM Products WHERE
Brand = 'Samsung'
```

Name	Brand
Note 5	Samsung
Galaxy S5	Samsung
Galaxy Alpha	Samsung
…	…

- **More examples:**

```
SELECT Name, Price, Description FROM Products
WHERE ProductID = 123
```

```
SELECT Name, Price FROM Products
WHERE Price <= 2000
```

Other Comparison

- Using BETWEEN operator to specify a range:

```
SELECT Name, Price FROM Products
WHERE Price BETWEEN 1000 AND 2000
```

- Using IN / NOT IN to specify a set of values:

```
SELECT Name, Description, BrandID FROM Products
WHERE BrandID IN (109, 3, 16)
```

- Using LIKE operator to specify a pattern:

```
SELECT Name FROM Products
WHERE Name LIKE 'S%'
```

  - % means 0 or more chars; _ means one char

Comparing with NULL

- **Checking for NULL value:**

```
SELECT Name, BrandID FROM Products
WHERE BrandID IS NULL
```

```
SELECT Name, BrandID FROM Products
WHERE BrandID IS NOT NULL
```

- **Attention: COLUMN = NULL is always false!**

```
SELECT NAME, BrandID FROM Products
WHERE BrandID = NULL
```

This is always false!

Logical Operators

- Using NOT, OR and AND operators and brackets:

```
SELECT Name, BrandID FROM Products
WHERE Price >= 1000 AND Brand LIKE 'S%'
```

```
SELECT Name FROM Products
WHERE BrandID IS NOT NULL OR Name LIKE '%5_'
```

```
SELECT Name FROM Products
WHERE NOT (ProductID = 3 OR ProductID = 4)
```

```
SELECT Name, Description FROM Products
WHERE
    (ProductID = 3 OR ProductID = 4) AND
    (Price >= 1000 OR ProductID IS NULL)
```

ORDER BY

- Sort rows with the ORDER BY clause
  - ASC: ascending order, default
  - DESC: descending order

```
SELECT Name, Price
FROM Products
ORDER BY Price
```

Name	Price
Galaxy Alpha	1600
HTC ONE	2000
iPhone 6S	3500

```
SELECT Name, Price
FROM Products
ORDER BY Price DESC
```

Name	Price
iPhone 6S	3500
HTC ONE	2000
Galaxy Alpha	1600

UPDATE

- **UPDATE command**
  - UPDATE <table> SET <column = expression> WHERE <condition>
  - **Note: Don't forget the WHERE clause!**

```
UPDATE Products
SET Name = 'Galaxy A5'
WHERE ProductID = 1

UPDATE Products
SET Price = Price * 1.10,
    Description = 'Smartphone...'
WHERE ProductID = 3
```

DELETE

- **Deleting rows from a table**
  - `DELETE FROM <table> WHERE <condition>`

```
DELETE FROM Products WHERE ProductID = 1
DELETE FROM Products WHERE Name LIKE '%5%'
```

  - **Note: Don't forget the `WHERE` clause!**
- **Delete all rows from a table at once**
  - `TRUNCATE TABLE <table>`

```
TRUNCATE TABLE Users
```

Nested SELECT

- **SELECT can be nested in the where clause**

```
SELECT Name, BrandID, Price
FROM Products
WHERE Price =
  (SELECT MAX(Price) FROM Products)
```

```
SELECT Name, BrandID, Price
FROM Products
WHERE BrandID IN
  (SELECT BrandID FROM Brands
      WHERE Brand = 'HTC')
```

  - **Note: always prefer joins to nested SELECT statements for better performance**

Nested SELECT with Aliases

- Tables from the main SELECT can be referred in the nested SELECT by aliases
- Example:
  - Find the maximal price for each Brand and the name of the product that gets it

```
SELECT Name, Price
FROM Products p
WHERE Price =
  (SELECT MAX(Price) FROM Products
   WHERE BrandID = p.BrandID)
ORDER BY ProductID
```

EXISTS

- Using the EXISTS operator in SELECT statements
  - Find all products with Brand from the first Brand

```
SELECT Name, Price, ProductID, BrandID
FROM Products p
WHERE EXISTS
  (SELECT ProductID
   FROM Products m
   WHERE m.ProductID = p.ProductID AND m.BrandID = 1)
```

Multiple Tables

- **Sometimes you need data from more than one table:**

Name	ProductID
Galaxy Alpha	1
iPhone 6S	3
HTC ONE	2

BrandID	Brand
21	Samsung
22	HTC
23	Apple

Name	Brand
Galaxy Alpha	Samsung
iPhone 6S	HTC
HTC ONE	Apple

Cartesian Product

- **This will produce Cartesian product:**

```
SELECT Name, Brand
FROM Products, Brands
```

- **The result:**

Name	Brand
Galaxy Alpha	Samsung
iPhone 6S	Samsung
HTC ONE	Samsung
Galaxy Alpha	HTC
iPhone 6S	HTC
HTC ONE	HTC
..	..

Cartesian Product

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition

Equijoins

- Inner joins with join conditions pushed down to the WHERE clause

```
SELECT p.ProductID, p.Name, p.BrandID, s.BrandID, s.Brand
FROM Products p, Brands s
  WHERE p.BrandID = s.BrandID
```

ProductID	Name	BrandID	BrandID	Brand
1	Galaxy Alpha	7	7	Samsung
2	HTC ONE	4	4	HTC
3	iPhone 6S	1	1	Apple

INNER JOIN

- To specify arbitrary conditions or specify columns to join, the ON clause is used
  - Such JOIN is called also INNER JOIN

```
SELECT p.ProductID, p.Name, p.BrandID, b.BrandID, b.Brand
FROM Products p
  INNER JOIN Brands b ON p.BrandID = b.BrandID
```

ProductID	Name	BrandID	BrandID	Brand
1	Galaxy Alpha	7	7	Samsung
2	HTC ONE	4	4	HTC
3	iPhone 6S	1	1	Apple

Additional Conditions

- You can apply additional conditions in the WHERE clause:

```
SELECT p.ProductID, p.Name, p.BrandID, b.BrandID, b.Brand
FROM Products p
  INNER JOIN Brands b ON p.BrandID = b.BrandID
WHERE p.Price >= 1000
```

ProductID	Name	BrandID	BrandID	Brand
1	Galaxy Alpha	7	7	Samsung
2	HTC ONE	4	4	HTC
3	iPhone 6S	1	1	Apple

Group Functions

- **Group functions operate over sets of rows to give one single result (per group)**

ProductID	Price
1	12500,00
2	13500,00
3	43300,00
4	29800,00
5	25000,00
…	…

MAX(Price)
125500,00

Group Functions

- `COUNT(*)` – count of the selected rows
- `SUM(column)` – sum of the values in given column from the selected rows
- `AVG(column)` – average of the values in given column
- `MAX(column)` – the maximal value in given column
- `MIN(column)` – the minimal value in given column

AVG and SUM

- **You can use AVG and SUM only for numeric data types**

```
SELECT
  AVG(Price) [Average Price],
  MAX(Price) [Max Price],
  MIN(Price) [Min Price],
  SUM(Price) [Price Sum]
FROM Products
WHERE Brand = 'Samsung'
```

Average Price	Max Price	Min Price	Price Sum
32700.00	32700.00	32700.00	98100.00

MIN and MAX

- You can use MIN and MAX for almost any data type (int, datetime, varchar, ...)

```
SELECT MIN(Price) MinPrice, MAX(Price) MaxPrice
FROM Products
```

MinPrice	MaxPrice
1600	3500

- Displaying the product's name in alphabetical order:

```
SELECT MIN(Name), MAX(Description)
FROM Products
```

COUNT

- **COUNT(*) returns the number of rows in the result record set**

```
SELECT COUNT(*) Cnt FROM Products
WHERE BrandID = 3
```

Cnt
18

- **COUNT(expr) returns the number of rows with non-null values for the expr**

```
SELECT COUNT(BrandID) BrCount,
  COUNT(*) AllCount
FROM Products
WHERE BrandID = 16
```

BrCount	AllCount
1	2

Group Functions and NULL

- Group functions ignore NULL values in the target column

```
SELECT AVG(BrandID) Avg,
  SUM(BrandID) / COUNT(*) AvgAll
FROM Products
```

Avg	AvgAll
108	106

- If each NULL value in the BrandID column were considered as 0 in the calculation, the result would be 106

Creating Groups of Data

Products

BrandID	Price
12	10300
12	16800
12	16800
12	10300
2	28800
2	25000
2	29800
16	125500
16	60100
...	...

54200

83600

185600

BrandID	SUM (Price)
12	54200
2	83600
16	185600
...	...

GROUP BY

- We can divide rows in a table into smaller groups by using the GROUP BY clause
- The SELECT + GROUP BY syntax:

```
SELECT <columns>, <group_function(column)>
FROM   <table>
[WHERE <condition>]
[GROUP BY <group_by_expression> ]
[HAVING   <filtering_expression>]
[ORDER BY <columns>
```

- The <group_by_expression> is a list of columns

GROUP BY

- **Example of grouping data:**

```
SELECT BrandID, SUM(Price) as TotalPrice
FROM Products
GROUP BY BrandID
```

BrandID	TotalPrice
12	72000
2	108600
16	185600
…	…

- **The GROUP BY column is not necessary needed to be in the SELECT list**

Grouping by More Columns

BrandID	Product	Price
Sony	Sony Xperia	1500
Samsung	Galaxy Alpha	1600
Samsung	Galaxy 5S	3500
Apple	iPhone 6S	4000
Apple	iPhone 5S	2500
HTC	HTC ONE	2100
HTC	HTC Desire	1700
LG	LG Optimus	2200
...	...	...

1500

5100

6500

3800

2200

Brand	Price
Sony	1500
Samsung	5100
Apple	6500
HTC	3800
LG	2200
...	...

Grouping by More Columns

- **Example of grouping data by several columns:**

```
SELECT BrandID, Name,
  SUM(Price) as Prices, COUNT(*) as Count
FROM Products
GROUP BY BrandID, Name
```

BrandID	Name	Prices	Count
2	Galaxy Alpha	58600	2
2	Galaxy S5	50000	2
7	iPhone 5S	525000	21
7	iPhone 6S	1926000	157
…	…	…	…

HAVING

- HAVING works like WHERE but is used for the grouping functions

```
SELECT BrandID, COUNT(ProductID) as
  Count, AVG(Price) AveragePrice
FROM Products
GROUP BY BrandID
HAVING COUNT(ProductID) BETWEEN 3 AND 5
```

BrandID	Count	AveragePrice
2	4	27150
12	5	14400
…	…	…

Standard Functions

- **Single-row functions**
  - **String functions**
  - **Mathematical functions**
  - **Date functions**
  - **Conversion functions**
- **Multiple-row functions**
  - **Aggregate functions**

String Functions

- Changing the casing – LOWER, UPPER
- Manipulating characters – SUBSTRING, LEN, LEFT, RIGHT, LTRIM, REPLACE

```
SELECT Name, LEN(Name) AS NameLength,
  UPPER(Name) AS UpperName FROM Products
WHERE RIGHT(Name, 2) = 'Ga'
```

Name	NameLength	UpperName
Galaxy Alpha	12	GALAXY ALPHA
Galaxy S6	9	GALAXY S6
Galaxy Note	11	GALAXY NOTE
…	…	…

Other Functions

- **Mathematical Functions – ROUND, FLOOR, POWER, ABS, SQRT, …**

```
SELECT FLOOR(3.14) → 3
SELECT ROUND(5.86, 0) → 6.00
```

- **Date Functions – GETDATE, DATEADD, DAY, MONTH, YEAR, …**
- **Conversion Functions – CONVERT, CAST**

```
SELECT CONVERT(DATETIME, '20151231', 112)
→ 2015-12-31 00:00:00.000
-- 112 is the ISO formatting style YYYYMMDD
```

Transactional Control Language (TCL)

ACID Principles

- **Atomicity: ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.**
- **Consistency: ensures that the database properly changes states upon a successfully committed transaction.**
- **Isolation: enables transactions to operate independently of and transparent to each other.**
- **Durability: ensures that the result or effect of a committed transaction persists in case of a system failure.**

T-SQL

- T-SQL (Transact SQL) is an extension to the standard SQL language
  - T-SQL is the standard language used in MS SQL Server
  - Supports `if` statements, loops, exceptions
    - Constructions used in the high-level procedural programming languages
  - T-SQL is used for writing stored procedures, functions, triggers, etc.

T-SQL Example

```
CREATE PROCEDURE Products AS
  DECLARE @ProductID INT, @Name NVARCHAR(100),
    @Price INT(11,2)
  DECLARE prods CURSOR FOR
    SELECT ProductID, Name, Price FROM Products

  OPEN prods
  FETCH NEXT FROM prods INTO @ProductdID, @Name, @Price
  WHILE (@@FETCH_STATUS = 0) BEGIN
    PRINT CAST(@ProductID AS VARCHAR(10)) + ' ' + @Name
    FETCH NEXT FROM prods INTO @ProductID, @Name, @Price
  END
  CLOSE prods
  DEALLOCATE prods
GO
```

Transactions

- Transactions start by executing BEGIN TRANSACTION (or just BEGIN TRAN)
- Use COMMIT to confirm changes and finish the transaction
- Use ROLLBACK to cancel changes and abort the transaction

```
BEGIN TRANSACTION
    DELETE FROM Products;
    DELETE FROM Brands;
ROLLBACK TRANSACTION -- or COMMIT TRANSACTION
```

Stored Procedures

- T-SQL is the standard language used in MS SQL Server
- Stored Procedures are reusable T-SQL code
- CREATE, ALTER, DROP PROCEDURE

```
CREATE PROCEDURE uspGetProducts AS
    SELECT * FROM Products
GO
```

- Input/output parameters

```
CREATE PROCEDURE uspGetProducts @BrandID INT = NULL AS
    SET NOCOUNT ON
    SELECT * FROM Products WHERE BrandID = @BrandID
GO
```

Stored Procedures

- **Call Stored Procedures**

```
EXEC uspGetProducts 3
-- OR
EXEC uspGetProducts @BrandID = 3
```

- **Example**

```
CREATE PROCEDURE uspGetProducts @BrandID INT = NULL
AS
    SELECT * FROM Products WHERE BrandID = @BrandID
GO
```

TRY CATCH

- **BEGIN TRY / BEGIN CATCH**
- **RAISEERROR raises an exception**

```
BEGIN TRY
    -- Generate a divide-by-zero error.
    SELECT 1/0
END TRY
BEGIN CATCH
    RAISERROR ('Error raised in TRY block.', 16, 1);
END CATCH
GO
```

TRY CATCH with TRAN

```
CREATE PROCEDURE uspDeleteBrand @BrandID INT AS
BEGIN TRY
    BEGIN TRANSACTION;
        DELETE FROM Products WHERE BrandID = @BrandID;
        RAISERROR('Delete of products failed', 16, 1);
        DELETE FROM Brands WHERE BrandID = @BrandID;
    -- If the DELETE statement succeeds, commit the
    -- transaction.
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
END CATCH
```