Basics of database handling, ADO.NET

Software techniques

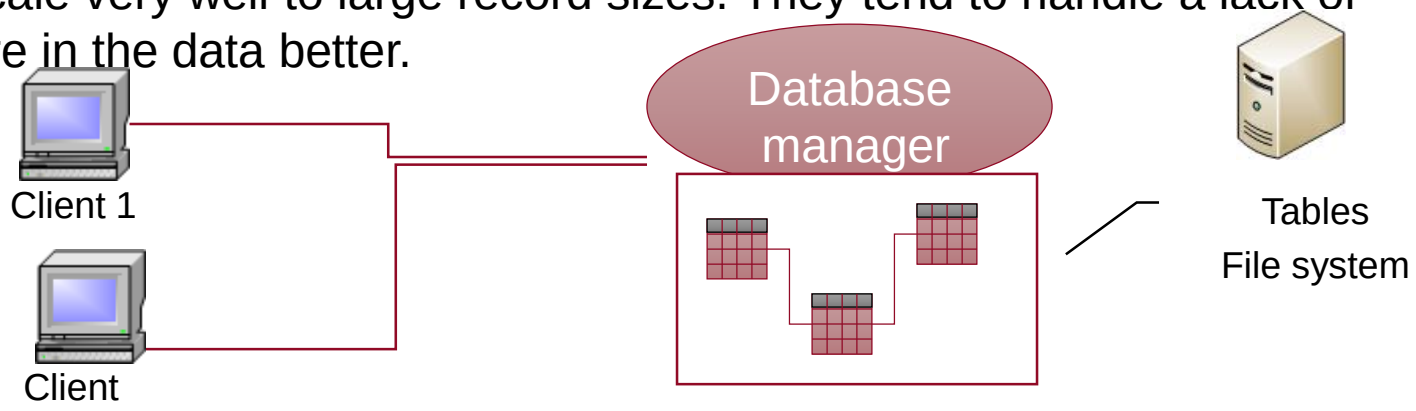
BME AUT
Department of Automation and Applied Informatics

Agenda

- Fundamentals of relational data models
- SQL
- Object relational mapping
- ADO.NET

Overview, architecture

- The scope of this lecture
  - > Next semester a whole class deals with databases
  - > This is just an overview now
  - > Goal: be able to develop applications that use database.
- Most database management systems support **relational** models
  - > Best known DBMS products: Oracle, IBM DB2, Microsoft SQL Server, MySQL
  - > Well suited for structured data
- Relationless (**NoSQL**) databases are growing in popularity
  - > Most scale very well to large record sizes. They tend to handle a lack of structure in the data better.

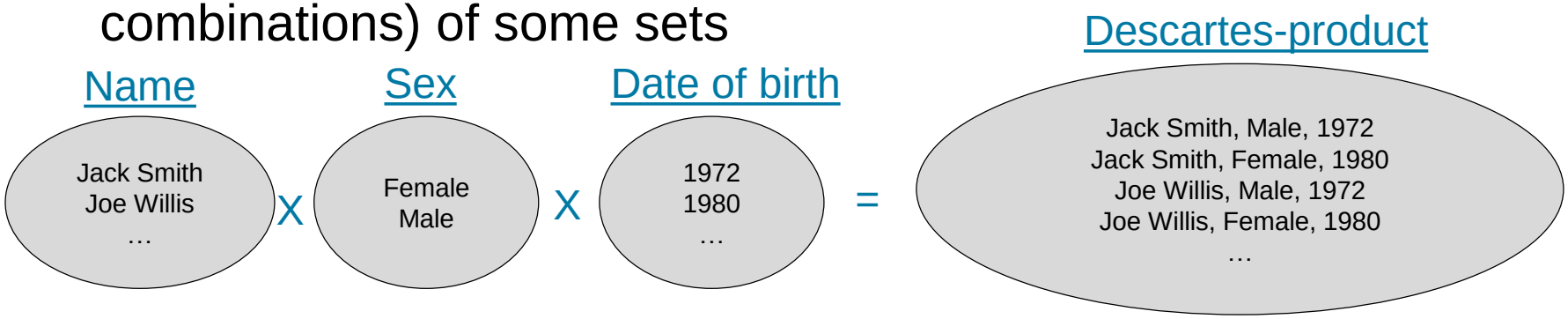


The fundamentals of relational data model

(Short excerpt from the databases course)

Fundamentals of relational data models

- Most database management systems support relational models
- Relation: a subset of the Descartes-product (all the possible combinations) of some sets



- In other words, a relation is a 2-dimensional table with named columns and any number of rows.

Name	Sex	Date of birth
Jack Smith	Male	1972
Joe Willis	Male	1980
…	…	…

Basic concepts of relational data models

- Attribute
  - The columns of the table
  - In the previous example there are three attributes: Name, Sex and Date of birth
  - Has got a domain (e.g. bool: true/false, 32 bit int, max 200 char long string, …)
- Record
  - One row of the table, also called an element of the relation
- Relational schema
  - Describes the attributes of a relation:
    - RelationName(A_1, A_2, …A_N)
    - Example: Person(Name, Sex, Date of birth)

Concepts of relational data models

- There is a single item in every cell of the table
  - >Each row is unique
  - >Each column has a unique name
  - >The order of rows is not important
  - >The order of the columns is not important
- Relational database schema: the set of relation schemas of each table

Fundamentals of relational data models

- **Anomalies**

  Example: every student choses a specialization that belongs to a department

Neptun	Name	Year	Specialization	Dep.name	Specialization manager
SD3ER5	George Gee	4	Infocomm	HIT	Dr. Tranzisztor Béla
AD1212	Joe Smith	3	Electrical eng.	VET	Dr. Apertúra János
DHJ7M4	Stephen Willis	3	Software eng.	AUT	Dr. Reláció György
96JGTG	Jane Ewing	4	Software eng.	AUT	Dr. Reláció György
85JUHG	Peter Blacksmith	3	Software eng.	AUT	Dr. Reláció György

There is redundancy in the data

- **Update anomaly**: if the value of an attribute is changed it has to be modified in more than one places (e.g. modifying the faculty manager for AUT)
- **Delete anomaly**: if I delete Joe Smith the data about the whole faculty is lost
- **Insert anomaly**: a new specialization cannot be inserted without inserting a new a student.

Basic concepts

- Functional dependecy
  - > If the rows of relation R match on attributes A^1, A^2, …, A^n , and this also entails that they match on attribute B, then B is functionally dependent on A^1, A^2, …, A^n
  - > In simpler terms: Attribute B is funkcionally dependent on A^1, A^2, …, A^n, if knowin their value we also know B's value.
  - > Notation: A^1, A^2, …, A^n → B

- Examples
  - > NeptunCode → Name
  - > Specialization → Department, Specialization manager
  - > Name, Address, DoB → Social Security Number (SSN)

Basic concepts

- Key
  - > A set of attributes that must meet two criteria:
    - 1. All other attributes must be functionally dependent on the key: **unique**.
      - For each row, the key must be unique.
      - The key's value identifies the row.
    - 2. If we remove any of the attributes from the set, the first criteria will no longer hold: **minimal**.
- Example:
  - > Student(SSN, NeptunCode, Name, Address, DoB)
    - Func deps: SSN→ NeptunCode, Name, Address, DoB
      NeptunCode → SSN, Name, Address, DoB
      Name, Address, DoB → SSN, NeptunCode
    - Keys: (SSN), (NeptunCode), (Name, Address, DoB), because they all determine a single row (they are unique in the relation)

Basic concepts

- **Primary key (PK)**
  - > We select one of the keys for this purpose
  - > We underline it in notations: Student(NeptunCode, SSN, Name, Address, DoB)
- **Foreign key (FK)**
  - > A reference to another table's primary key
  - > Example:

    Specialization(SpecName, Department, SpecManager)

    Student(NeptunCode, Name, ***SpecName***, Year)
    - – The SpecName attribute in the Student table is a foreign key to elements in the Specialization table.
  - > We describe relations between tables with foreign keys
  - > Database management systems (DBMS) check the value of foreign keys
    - – Eg. A Student may only have a SpecName that appears in the Specialization table
    - – As such foreign keys are important to maintaining data integrity!

Basic concepts

- NULL attributes
  - If specified, some attributes can also have NULL value.
- Details about primary keys
  - > In practice we usually generate an ‘artificial’ primary key
  - > Eg.
    - Employee (EmployeeID, Name, Address, …)
    - Comodity (ComodityID, Name, Price, Description,…)
  - > Usually one of the two:
    - AutoIncrement ed - the DBMS increases its value automatically by one each time a new row is inserted.
    - GUID (Globaly Unique Identifier) – a 128 bit randomly generated value

Normalization

- Normalization
  - > Some table schemas lead to anomalies
  - > To avoid this, we decompose the relation (table) to multiple relations, taking the shape of some **normalized form**
  - > In practice we use **BCNF** and **3NF**
    - – Out of the scope of this lecture
  - > Let's normalize our previous example
    - – Specialization(SpecializationName, Depart., SpecManager)
    - – Student(NeptunCode, Name, Year, ***SpecName***)

Student table

NeptunCode	Name	Year	SpecName
SD3ER5	Kotányi Aladár	4	Infokomm. rendszerek
AD1212	Kalapács Attila	3	Vill. energetika
DHJ7M4	Lopovszky Károly	3	Szám. rendszerek
96JGTG	Keltai János	4	Szám. rendszerek
85JUHG	Baldai Baldvin	3	

SpecName is a foreign key in the Student table

Specialization table

SpecializationName	Depart.	SpecManager
Infokomm. rendszerek	HIT	Dr. Tranzisztor Béla
Vill. energetika	VET	Dr. Apertúra János
Szám. rendszerek	AAIT	Dr. Reláció György

Normalization

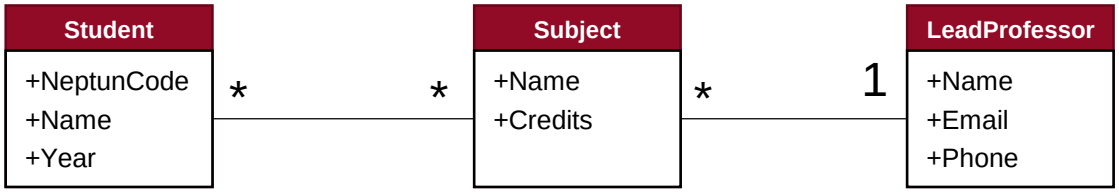
- Denormalization
  - We sometimes (rarely) ‘break’ a normalized schema
    - Eg. To store a computed value
    - Eg: the number of students could be stored in the Specialization table (to avoid counting each time)
  - Only if we must!
    - Hard to properly maintain, can easily lead to inconsistency!
- Transactions
  - Execute several operations in one go
  - Maintain ACID principles
    - Atomicity, Consistency, Isolation, Durability
    - (Later in other courses)

Object- relational mapping

ORM

Object-relational mapping

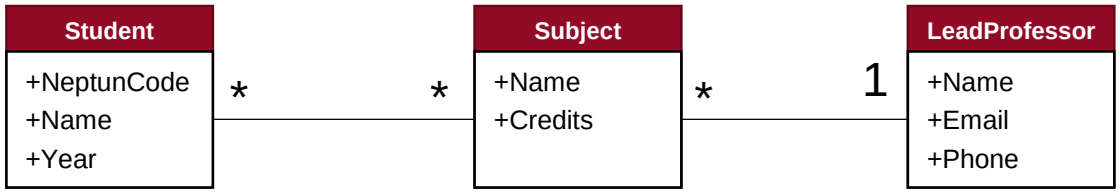
- We design databases in two main ways
  - > A) „Traditional”: we plot and decompose relations, normalizing them. This is what we saw before.
  - > B) Take a theoretical entity model (eg. UML class diagram or Entity-Relationship diagram) ➔ Classes/entities will become tables.
- Example for B)
  - > Take the following UML class diagram



  - > Looking back at the first lecture – mapping connections in code:
    - – One-to-many connection - Lead professor to Subject: The LeadProfessor class has a list of pointers of references to the Subject objects they are in charge of. The Subject class has a pointer or reference to its LeadProfessor.
    - – Many-to-many connection - Student – Subject: both classes have a list of pointers/references to the other.

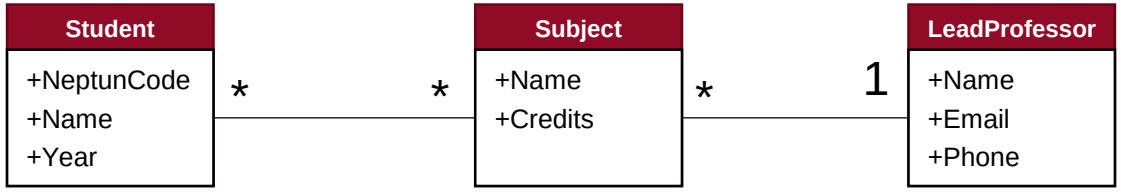
Object-relational mapping

- How do we get from the class diagram to database tables.

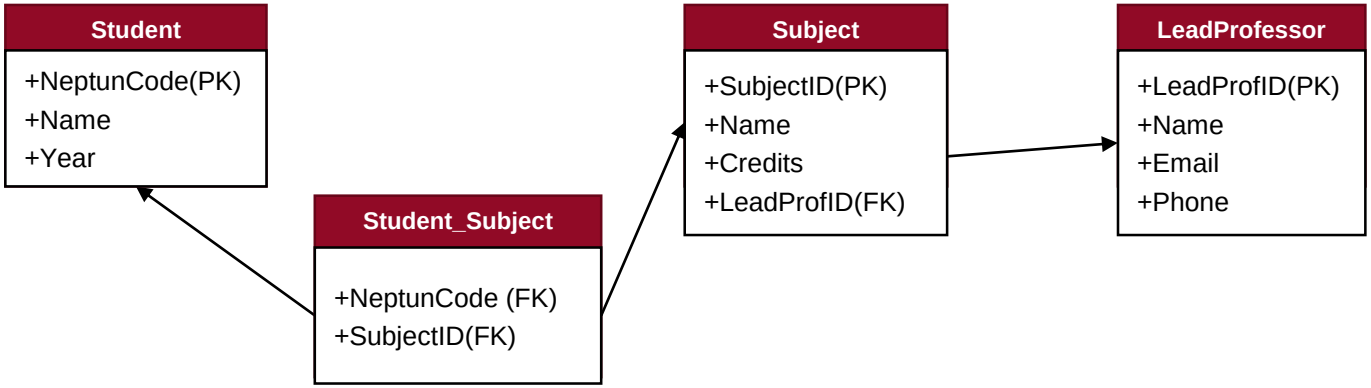


  - Classes, member variables:
    - Tables, attributes
  - One-many relationship
    - on the many side of the relation introduce a foreign key into the table
  - Many-many relationship
    - introduce a new joining table that contains a foreign key for both tables

Object-relational mapping



- This is the database schema:



> We have a fourth connecting table: Student_Subject

Object-relational mapping

- Let's look at our database with some data

Student

NeptunCode	Name	Year
SD3ER5	John Doe	4
AD1212	Kalapács Attila	3
DHJ7M4	Lopovszky Károly	3
96JGTG	Keltai János	4

Subject

SubjectID	Name	Credits	LeadProfID
1	Prob. theory	2	2
2	Software te.	4	1
3	Databases	3	1

Student_Subject

NeptunCode	SubjectID
SD3ER5	2
SD3ER5	3
AD1212	1
…	…

LeadProfessor

LeadProfessorID	Name	…
1	Dr. Reláció György	
2	Kalapács Attila	
3	Lopovszky Károly	
4	Keltai János	

John Doe is a student of Szoftech and Database subjects.

Dr. Reláció György is lead professor for Szoftech and Databases.

Check: are there anomalies?

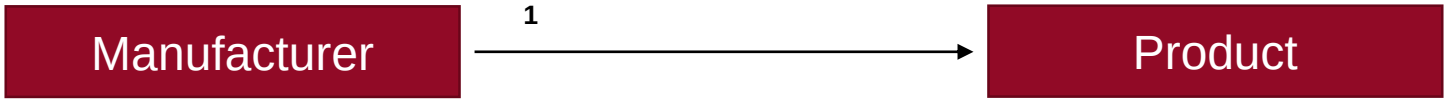
The basics of SQL

A Structure Query Language

SQL introduction

- **SQL - Structured Query Language**
- Language elements can be split into two groups
  - > **DDL - Data Definition Language**
    - – E.g. Creating and modifying tables
  - > **DML – Data Manipulation Language**
    - – E.g. Inserting, deleting, selecting, updating records

- Example
  - > Product and Manufacturer: a product is produced by one manufacturer, a manufacturer produces multiple products
  - > Manufacturer(ManufacturerID, Name, Address)
  - > Product(ProductID, Name, Price, ManufacturerID)



SQL – data definition

- **Creating the tables**

```
CREATE TABLE Manufacturer
(
      ManufacturerID int NOT NULL PRIMARY KEY,
      Name nvarchar(50) NOT NULL,
      Address nvarchar(200) NOT NULL
)

CREATE TABLE Product
(
      ProductID int NOT NULL PRIMARY KEY,
      Name nvarchar(50) NOT NULL,
      Price int NOT NULL,
      ManufacturerID int NULL REFERENCES Manufacturer(ManufacturerID)
)
```

  - CREATE TABLE <table name>, and then comes the list of columns. For every column:
    - Name, type, null is allowed or not (optional)
    - optional: PRIMARY KEY (the column is the primary key of the table), REFERENCES <referenced table name>(<referenced column name>) – for defining foreign keys

SQL – data manipulation

Insert a row into a table

```
INSERT INTO Manufacturer(ManufacturerID,Name,Address) VALUES(1,'Nike','Beaverton')
INSERT INTO Product(ProductID, Name, Price, ManufacturerID) VALUES (1,
'Nike Flyknit Air Max', 250, 1)
```

INSERT INTO <table name>(list of columns) VALUES (list of values)

Delete (some) row(s)

```
-- Delete all products
DELETE FROM Product
-- Delete all products that cost 400
DELETE FROM Product WHERE Price = 400
```

DELETE FROM <table name> (column list) WHERE <condition>

Update (some) row(s)

```
-- update product with id = 2
UPDATE Product SET Name = 'Nike Air Max 2014', Price = 220 WHERE ProductID = 1
-- Increas the price of all product that cost less then 300
UPDATE Product SET Price = Price*1.2 WHERE Price < 300
```

UPDATE <table name> columnname1=value1, columnname2=value2

WHERE <condition>

SQL queries

- **Query**

  SELECT list of columns FROM table name WHERE <condition>

- **Conditions**
  - =, <, >, >=, <=, <>
  - epx1 BETWEEN exp2 AND exp3
  - exp1 LIKE <string pattern>, the string pattern may contain the following joker characters:
    - _: any character
    - %: any string

    Examples:
    - Strings starting with „bob” : LIKE ‘bob%’
    - Strings containing „bob” : LIKE ‘%bob%’
  - exp IS NULL
  - exp IS NOT NULL
  - AND, OR, NOT
  - EXISTS, NOT EXISTS: a result set contains any record or not

SQL queries

- **Examples**

```
-- Query every record of
-- the product table
SELECT *
FROM Product
```

	TermekID	Nev	Ar	GyartoID
1	1	Termék1	100	1
2	2	Termék2	200	1
3	3	Termék3	400	2
4	4	Termék4	400	NULL

```
-- Query every product
SELECT Name, Price
FROM Product
```

```
SELECT *
FROM Product
WHERE Price > 400
```

```
SELECT *
FROM Product
WHERE Price > 400 OR Price < 200
```

```
SELECT *
FROM Product
WHERE Price BETWEEN 200 AND 300
```

```
SELECT *
FROM Product
WHERE ManufacturerID IS NULL
```

```
SELECT *
FROM Product
WHERE Name LIKE '%unar%'
```

SQL queries

- Example 1: List all manufacturers with all their products

```
SELECT Manufacturer.*, Product.Name AS ProductName
FROM Manufacturer JOIN Product ON
   Manufacturer.ManufacturerID = Product.ManufacturerID
```

	GyartoID	Nev	Cim	TermekNev
1	1	Gyártó1	Cím1	Termék1
2	1	Gyártó1	Cím1	Termék2
3	2	Gyártó2	Cím2	Termék3

> JOIN, or INNER JOIN: joins two tables using the condition after the ON keyword
  - For every record in the manufacturer table it searches for the products where `Manufacturer.ManufacturerID = Product.ManufacturerID` is true.

SQL queries

- Example 2: List all the products with the corresponding manufacturer name

```
SELECT Product.*, Manufacturer.Name AS ManufacturerName
FROM Product JOIN Manufacturer
   ON Product.ManufacturerID = Manufacturer.ManufacturerID
```

- Example 3: List all the products with the corresponding manufacturer name, but also list the products where manufacturer is not specified (ManufacturerId is null)

```
SELECT Product.*, Manufacturer.Name AS ManufacturerName
FROM Manufacturer RIGHT OUTER JOIN Product
ON Product.ManufacturerID = Manufacturer. ManufacturerID
```

  - A LEFT OUTER JOIN takes the left side record into the result set even if there is no matching rights side record
  - A RIGHT OUTER JOIN takes the right side record into the result set even if there is no matching left side record

SQL queries

- List all manufacturers that have got at least two products with a price greater that 200

```
select p.ManufacturerId, count(*) as prodCount
from Manufacturer m
 RIGHT OUTER JOIN Product p ON p.ManufacturerID = m.ManufacturerID
where p.price > 200
group by p.ManufacturerId
having count(*) > 1
```

Group by

- Groups are created so that in each group the group by parameters have got the same value
- The select list can only contain columns that are either in the group by clause or in an aggregate function
- Having is used to filter for groups

ADO.NET

ADO.NET

- What is ADO.NET?
  - > Relational database access from .NET environment
    - – E.g. Oracle, MSSQL, Access, …
- What can it be used for?
  - > Execute SQL commands and read the result set
- Other DB access solutions in .NET
  - > Entity Framework is an ORM, much newer technology
  - > ADO.NET is the ‘traditional’ way of accessing a database, we’ll be looking at this
- Approach
  - > The approach of ADO.NET is quite similar to many non-.NET solutions.

ADO.NET – object model

- Most important objects for data access
  - Connection:
    - Represents a database connection to a database manager
    - Before executing any command a connection has to be created.
  - Command:
    - Can execute a SQL command (or stored procedure)
  - DataReader
    - Can be used to iterate through the result set of a query (e.g. SELECT).

ADO.NET – Provider-based architecture

- Provider-independent interfaces and abstract base classes
  - > IConnection, DbConnection
  - > ICommand, DbCommand
  - > IDataReader, DbDataReader

- Provider-specific implementations
  - > Derive from the above mentioned base classes
  - > E.g..:
    - Oracle: OracleConnection, OracleCommand, OracleDataReader
    - MSSQL: SqlConnection, SqlCommand, SqlDataReader
    - …

ADO.NET – Connection oriented data access

- Steps(using MS SQL provider)
  - > Instantiate SqlConnection instance with the parameters of the connection (connection string)
  - > Create an SqlCommand instance
    - – Specify the SQL command text or stored procedure name as parameter
    - – Set the SqlConnection instance to use when executing the command
  - > Execute the command (SqlCommand)
  - > If the command has got a result set (e.g. SELECT), an instance of SqlDataReader is used to iterate through the result set.
  - > Close the connection
- Methods of a command
  - > ExecuteNonQuery – for SQL commands that doesn't return anything (e.g. INSERT, UPDATE)
  - > ExecuteScalar – for SQL commands that return a scalar value.
  - > ExecuteReader – for SQL commands that return one or more result sets.

ADO.NET – Connection oriented data access

Example for a query

```
try
{
    // Create connection object
    conn = new SqlConnection("Data Source=(local); Initial Catalog=SzoftechDB; Integrated security = true");

    // Create database command
    SqlCommand command = new SqlCommand("SELECT ProductID, Name, Price FROM Product", conn);

    // Open connection
    conn.Open();

    // Query and print results
    reader = command.ExecuteReader();
    while (reader.Read())
    {
        Console.WriteLine("{0}\t{1}\t{2}", reader[“ProductID"].ToString(),
            reader["Name"].ToString(), reader[“Price"].ToString());
    }
}
finally
{
    if (reader != null) reader.Close();
    if (conn != null) conn.Close(); // We must make sure we close the connection!!!
}
```

ADO.NET – Connection oriented data access

- Key features
  - > DataReader can read the result set only once, forward, record by record (e.g. cannot step back to a previous record).
  - > reader["column name"] returns the value of a column of the current record
  - > The database connection is a heavy resource!!!
    - – Close it as soon as possible!
      - For non query operations right after executing the command
      - For queries after reading the result set with DataReader.
    - – Why is it an expensive resource?
      - Usually the number of connections that can be opened in parallel is limited: if many clients connect to a DB management system it can run out of free connections.
      - The max concurrent connection number is usually also limited by product licenses. You have to pay for more connections

ADO.NET – Connection oriented data access

Example for insert

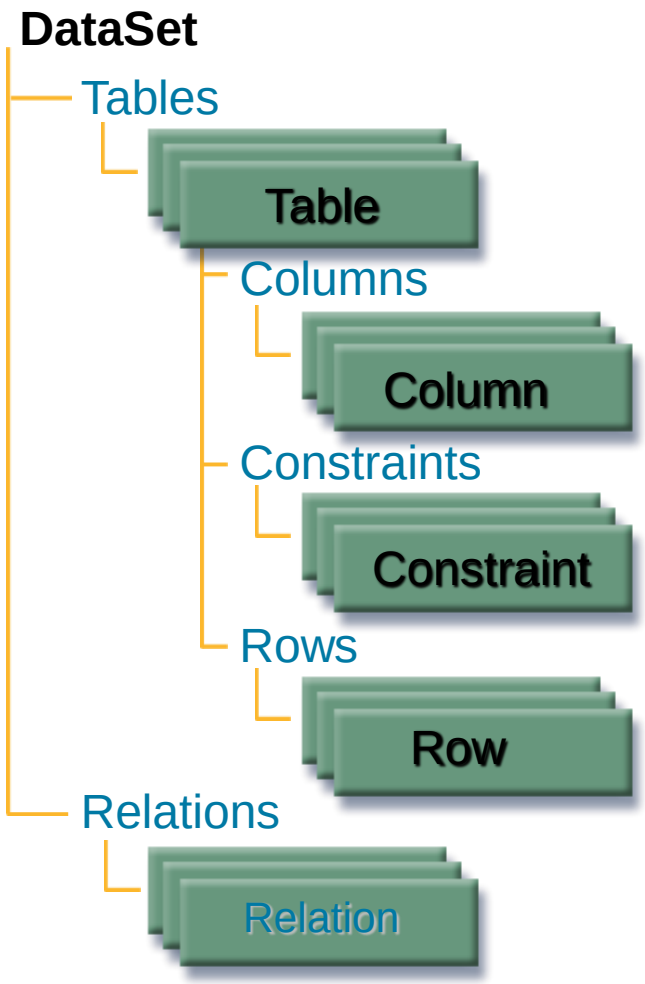
```
SqlConnection conn = null;
try
{
    // Connect to the database
    conn = new SqlConnection("Data Source=.\\SQLEXPRESS; Initial Catalog = SzoftechDB; Integrated Security = True");

    conn.Open(); // Open the connection

    //Watch out for SQL-INJECTION by passing parameters appropriately!!!
    SqlParameter pProductId = new SqlParameter("@ProductId", productId);
    SqlParameter pName = new SqlParameter("@Name", name);
    SqlParameter pPrice = new SqlParameter("@Price", price);

    // Create the command
    SqlCommand command = new SqlCommand(
        "INSERT INTO Product(ProductId, Name, Price) VALUES(@ProductId, @Name, @Price)");

    command.Parameters.Add(pProductId );
    command.Parameters.Add(pName);
    command.Parameters.Add(pPrice);

    // Provide the connection
    command.Connection = conn;
    int res = command.ExecuteNonQuery();
}
finally { if (conn != null) conn.Close(); }
```

ADO.NET – ‘Connectionless’ data access

- ‘Connectionless’ data access
  - Aka „DataSet-based” data access
  - Usual scenario
    - Query data (SELECT) from database and close the connection
    - Display data for the user (e.g. in a datagrid)
    - User edits data
    - User synchronizes the modifications: connects to the database and performs the modifications (INSERT, UPDATE, DELETE)
  - Problems to solve
    - After querying the data it has to be stored in memory
    - User modifications have to be stored (new rows, edited rows, deleted rows).
  - Solutions for these problems
    - Custom solutions
    - Using DataSet

ADO.NET – DataSet-based data access

- What is a DataSet?
  - > A temporary in-memory database(tables, columns, rows, relations)
- Key features
  - > The table stores the retrieved data
  - > Keeps track of all the modifications



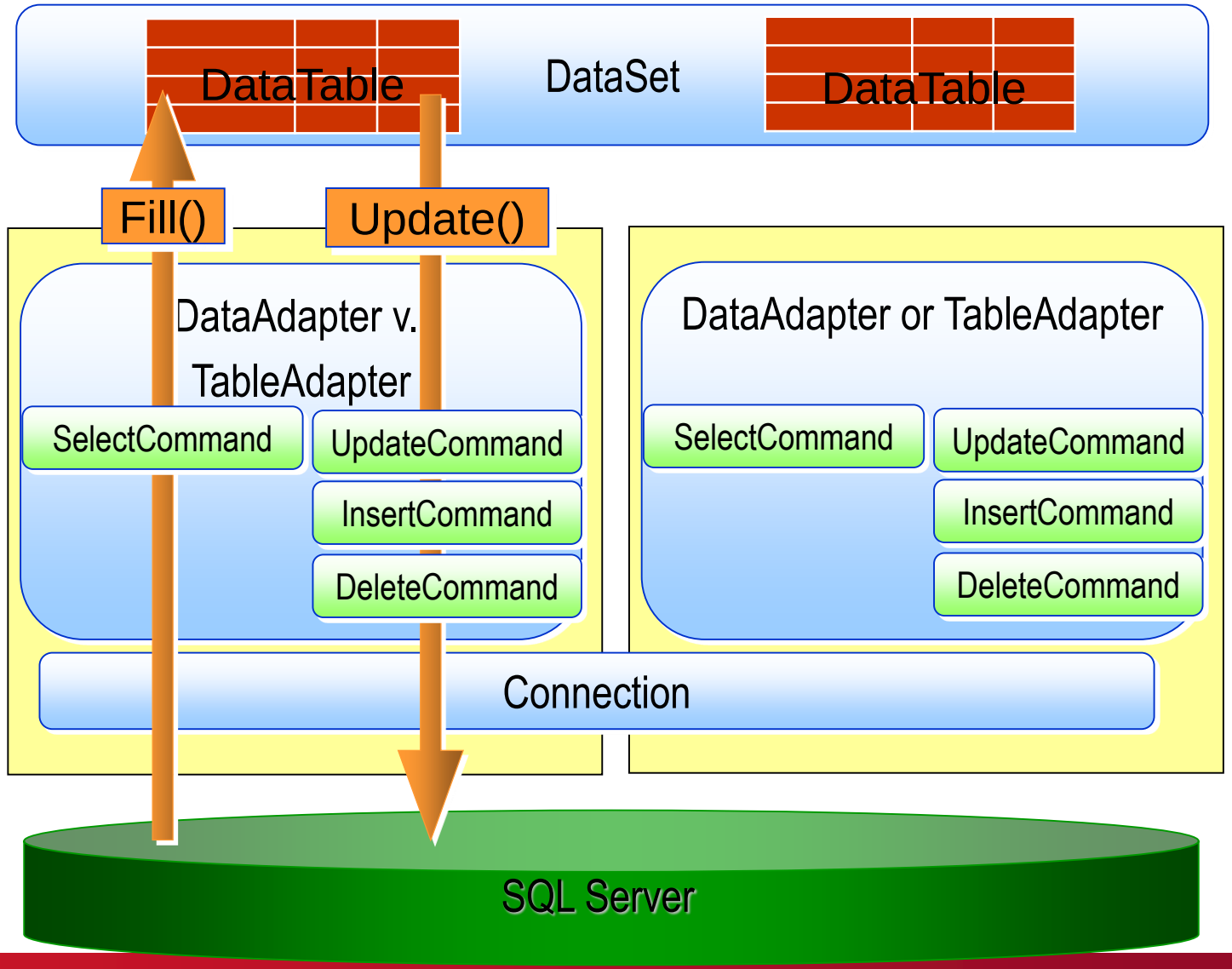
ADO.NET – DataSet-based data access

- Steps
  1) Query
     - Open the connection
     - Fill the tables of the DataSet from the result of a database query
     - Close the connection

  2) Display data from the DataSet & modify data (in memory)

  3) Sync modifications back to database
     - Open the connection
     - Save the modified rows of the DataSet
       - INSERT for every new row
       - UPDATE for every modified row
       - DELETE for every deleted row
     - Close the connection

ADO.NET – DataSet based data access



ADO.NET – DataSet based data access

```
// Create connection object
SqlConnection conn = new SqlConnection(
    "Data Source=(local); Initial Catalog=SzoftechDB;…");

// Create DataSet object
DataSet dsSzoftech = new DataSet();

// Create Data adapter: this will fill the DataSet, and save changes to the database.
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Product", conn);
adapter.InsertCommand = new SqlCommand(< an INSERT command >);
adapter.UpdateCommand = new SqlCommand(< an UPDATE command >);
adapter.DeleteCommand = new SqlCommand(< an DELETE command >);

// Filling the DataSet “Product" table: runs the adapter’s SELECT command
// The connection is opened and closed automatically
adapter.Fill(dsSzoftech, “Product");

// Print the Name of the first record of Product table
Console.WriteLine(dsSzoftech.Tables[“Product"].Rows[0][“Name"].ToString());

// Set the first record’s price to 999
dsSzoftech.Tables[“Product"].Rows[0][“Price"] = 999;

// Save changes to the database: the changed row is updated using the Update command
// The connection is opened and closed automatically
adapter.Update(dsSzoftech, “Product");
```

Looking beyond ADO.NET

- Problems
  - >A DataSet still deals with ‘raw’ SQL and relations
  - >Wouldn’t it be nice to write queries in C#, using C# classes…
- ORM: Entity Framework
  - >The most popular high level ORM for .NET (Microsoft)
- Query language: LINQ
  - >Also works on collections and other iterable types