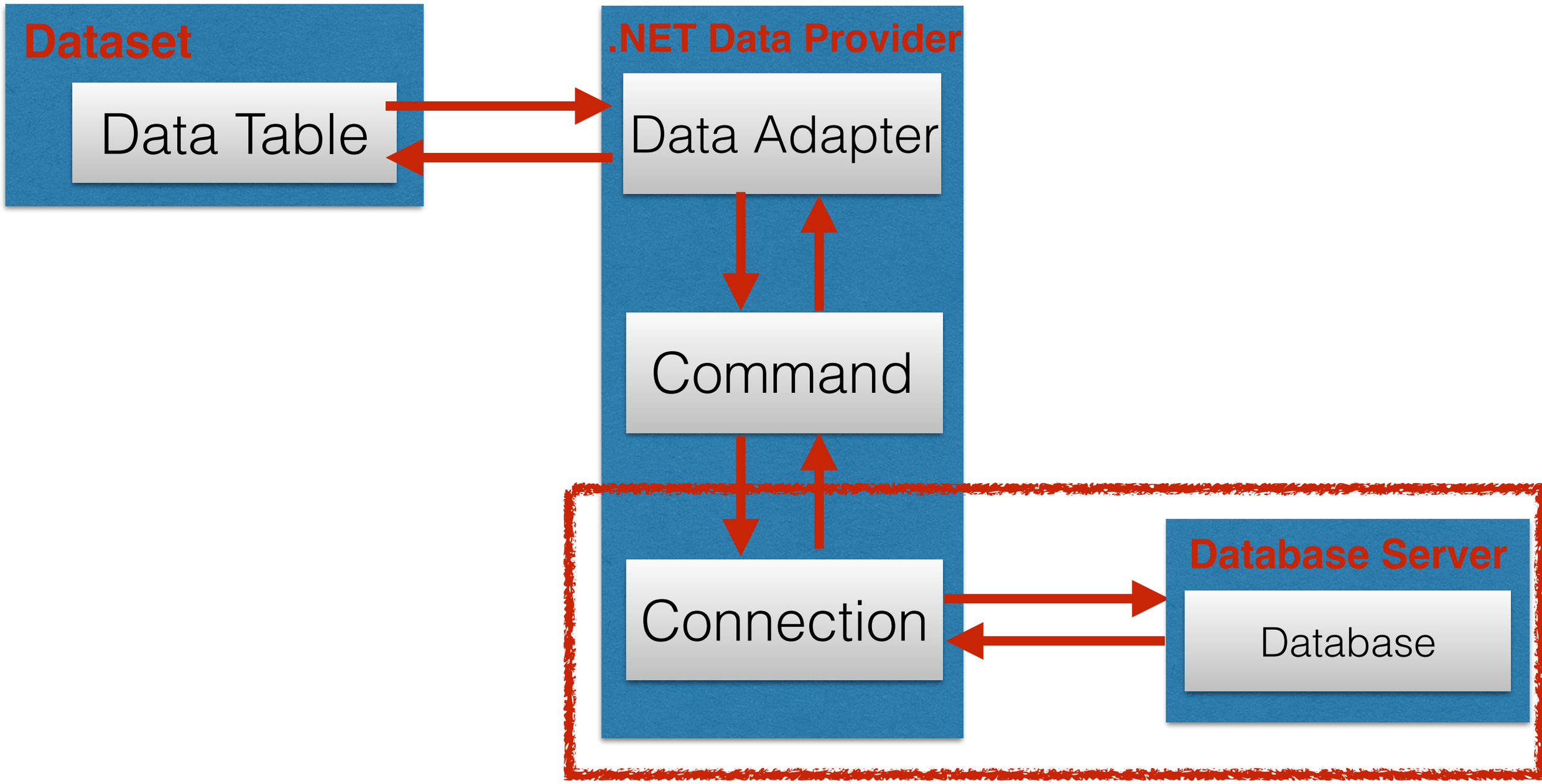
ADO.NET Transactions

ADO.NET Objects Architecture



Transactions

Frequently, some DB operations are depending on other operations.
Back to out Students DB, for example you do not want to add a new empty Course,
so first you will add the students enrolled in it.

In such common situations, we use **TRANSACTIONS**.

Transactions are groups of SQL commands which have to be executed all together as a single logical unit.

With transactions:

1) You can check that all the commands can be executed, before **committing** it
2) If one or more commands in the transactions cannot be executed, you can **roll back** to the previous state. A rollback can be complete or partial, with the use of **save points**.

Create and use Transactions

```
MySqlConnection conn;//Create the connection
conn.Open(); //you have to open first!
MySqlTransaction tr;
tr = conn.BeginTransaction() //create the transaction object

//Create a command (with a parameter in this case)
string s1 = "DELETE FROM T1 WHERE field1 = @name";
MySqlCommand cmd1 = new MySqlCommand(conn,s1);
cmd1.Parameters.Add("@name",MySqlDbType.VarChar,50);
cmd1.Connection = conn;
cmd1.Transaction = tr;//associate the transaction to the command
                      //you can associate more commands to it.
cmd1.Parameters["@name"].Value = "John";
cmd1.ExecuteNonQuery();
tr.Save("name1"); //Set a SAVE POINT

cmd1.Parameters["@name"].Value = "Jack";
cmd1.ExecuteNonQuery();
tr.Save("name2");//Set a SAVE POINT

tr.Rollback("name1"); //Rollback to the first command

tr.Commit(); //finally commit the transaction
```

Concurrency and Locking

When more than one user accesses the same DB **concurrency** problems can occur.

- Retrieving data generally do not lead to concurrency problems.
- Updating procedures are different!

EXAMPLES:
- Add table/record/field
- Delete table/record/field

Concurrency problems are usually treated with **LOCKING**.
It consists in preventing modifications to a DB while another transaction is performing modifications.

Most modern DBMS provide locking procedures.

The Concurrency Problems

Lost Updates:
Two transactions update a row at the same time and only the last update will remain in effect.

Dirty Reads:
It happens when a query selects data not yet committed by another transaction. The first transaction will read an outdated row.

Nonrepeatable Reads:
If an update occurs between two successive SELECT queries, the result of the queries will be different.

Phantom Reads:
This happens when during a DELETE or UPDATE on a group of rows when another transaction is doing INSERT or DELETE that affect at least one of the latter rows.

Isolation Levels

```
connection.BeginTransaction(IsolationLevel.LEVEL)
```

LEVEL:

`ReadUncommitted`
All concurrency problems are allowed

`ReadCommitted`
Prevents dirty reads only

`RepeatableRead`
Prevents dirty reads, lost updates, non repeatable reads.

`Snapshot`
Prevents all problems using “row versioning” (no locking)

`Serializable`
Prevents all concurrency problems.

With “Row versioning”, the data retrieved are always consistent with the data before the transaction. This is achieved by maintaining a snapshot of the original data to compare with.