FABIO RAMONI

DAPPER

.NET TORINO COMMUNITY

Chi sono

FABIO RAMONI

Full stack developer

TWITTER

@developer_fabio

GITHUB

@FabioDeveloper92

COS'E

E' una libreria micro-ORM open-source di .NET, ci consente di accedere rapidamente e facilmente ai dati di un database.

- Supporta MySQL, SQL Server, PostgreSQL, Oracle SQLite etc
- Prima release 2011 (ultima versione 2.0.123)
- Sviluppato da stack overflow

REPOSITORY LINK

https://github.com/DapperLib/Dapper

OBIETTIVI

Dapper si pone il compito di migliorare

- Perfomance
- Semplificare la gestione delle query
- Mapping dei risultati delle query
- Esecuzione stored procedures

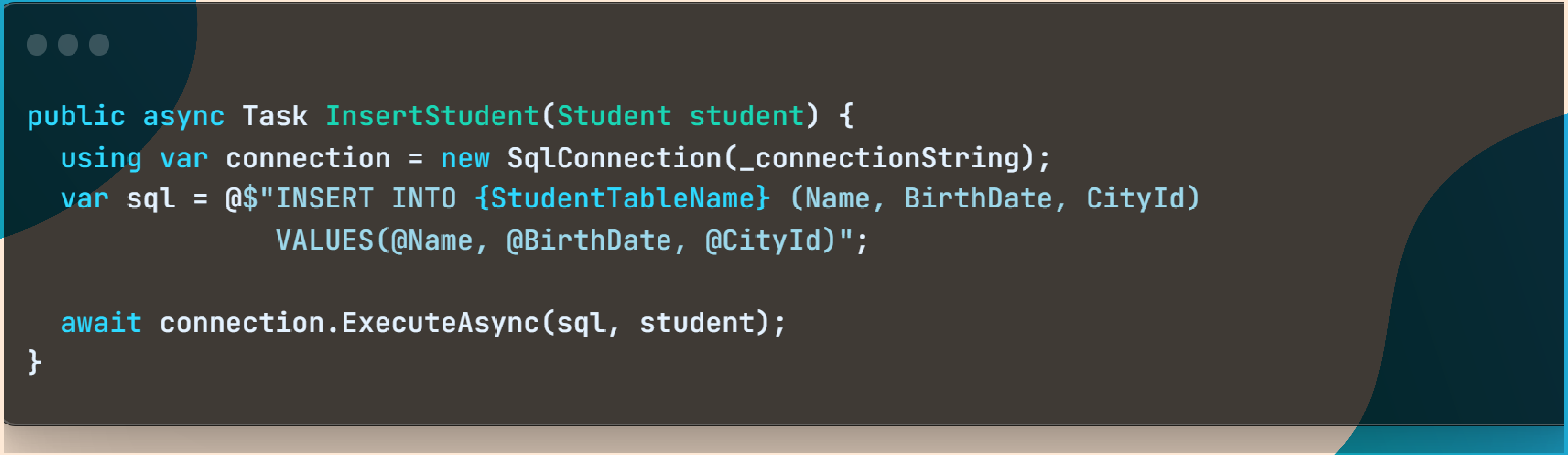
REPOSITORY LINK

https://github.com/DapperLib/Dapper

INSTALLAZIONE

```
dotnet add package Dapper

// Install the extensions for your database
dotnet add package Dapper.Extensions.MSSQL
```

EXECUTEASYNC

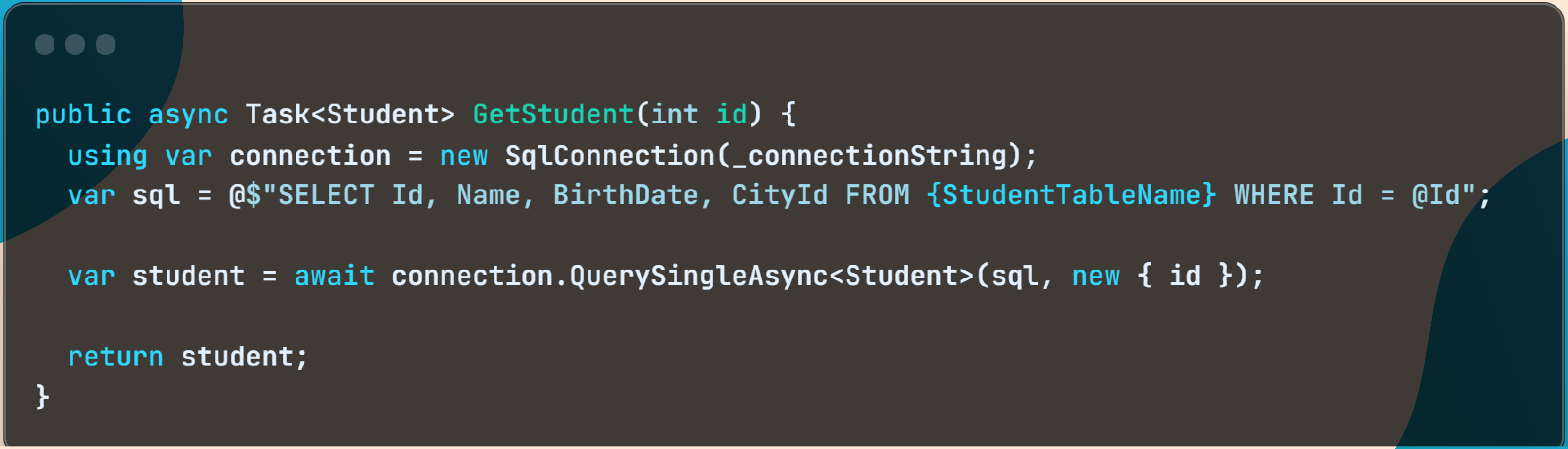


```
public async Task InsertStudent(Student student) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"INSERT INTO {StudentTableName} (Name, BirthDate, CityId)
               VALUES(@Name, @BirthDate, @CityId)";

  await connection.ExecuteAsync(sql, student);
}
```

EXECUTEASYNC (BULK)

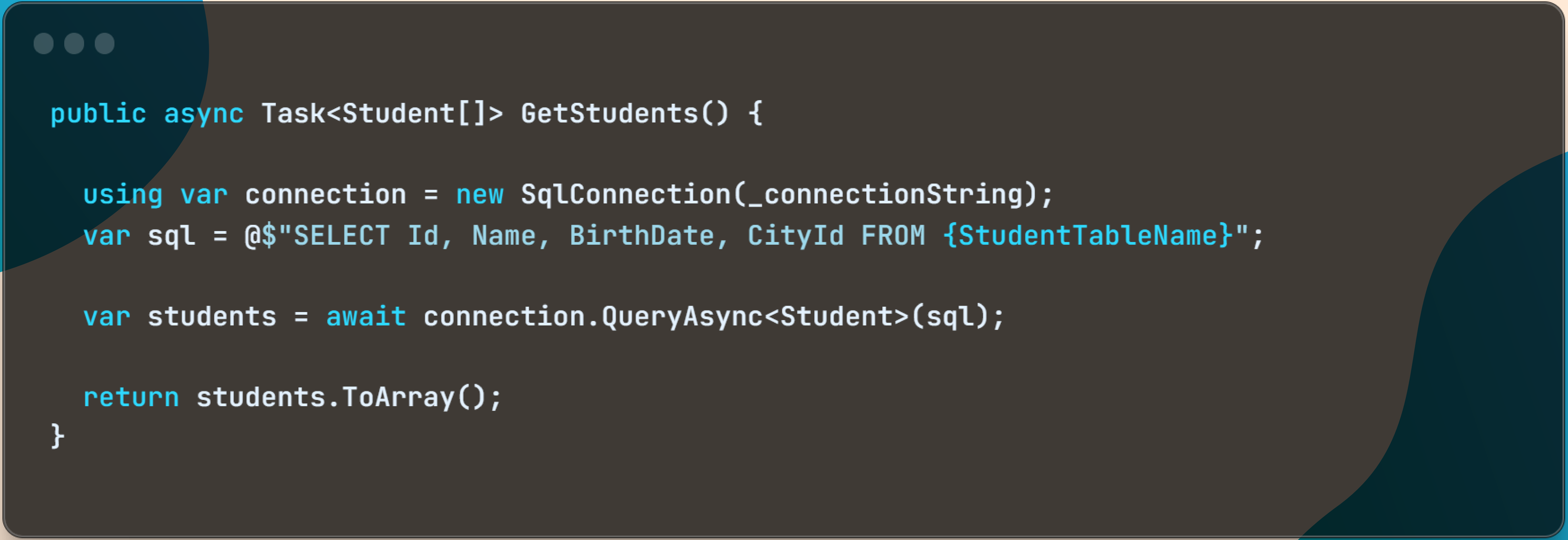
```
public async Task<int> InsertStudents(Student[] students) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"INSERT INTO {StudentTableName} (Name, BirthDate, CityId)
               VALUES (@Name, @BirthDate, @CityId)";

  var insertedRows = await connection.ExecuteAsync(sql, students);
  return insertedRows;
}
```

SELECT ONE ROW



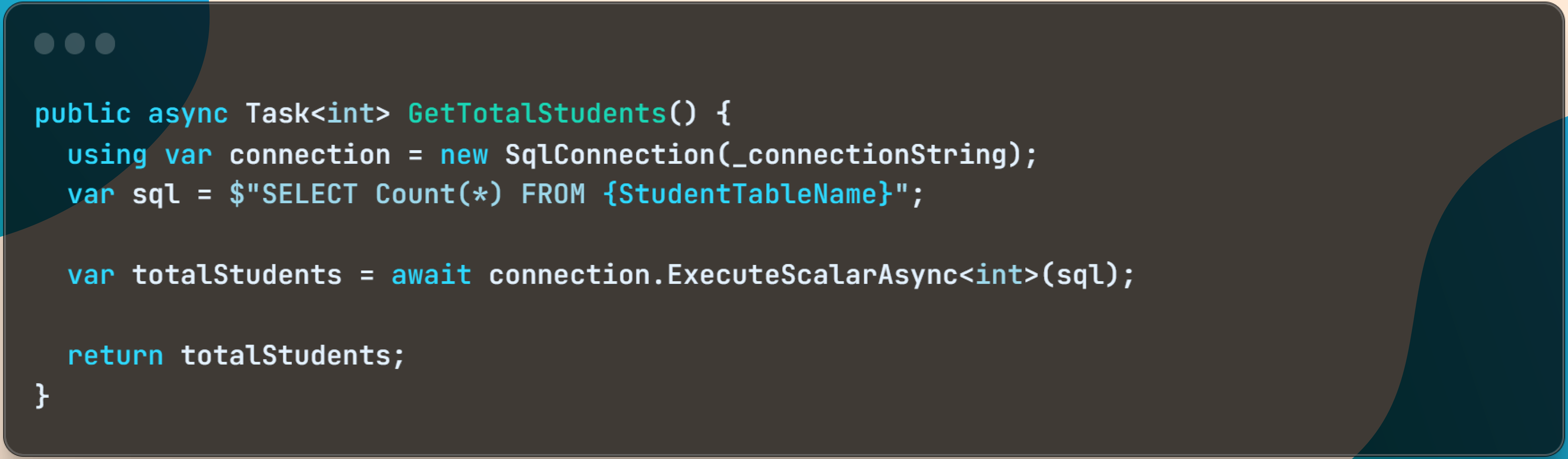
```
public async Task<Student> GetStudent(int id) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT Id, Name, BirthDate, CityId FROM {StudentTableName} WHERE Id = @Id";

  var student = await connection.QuerySingleAsync<Student>(sql, new { id });

  return student;
}
```

SELECT MULTI ROW



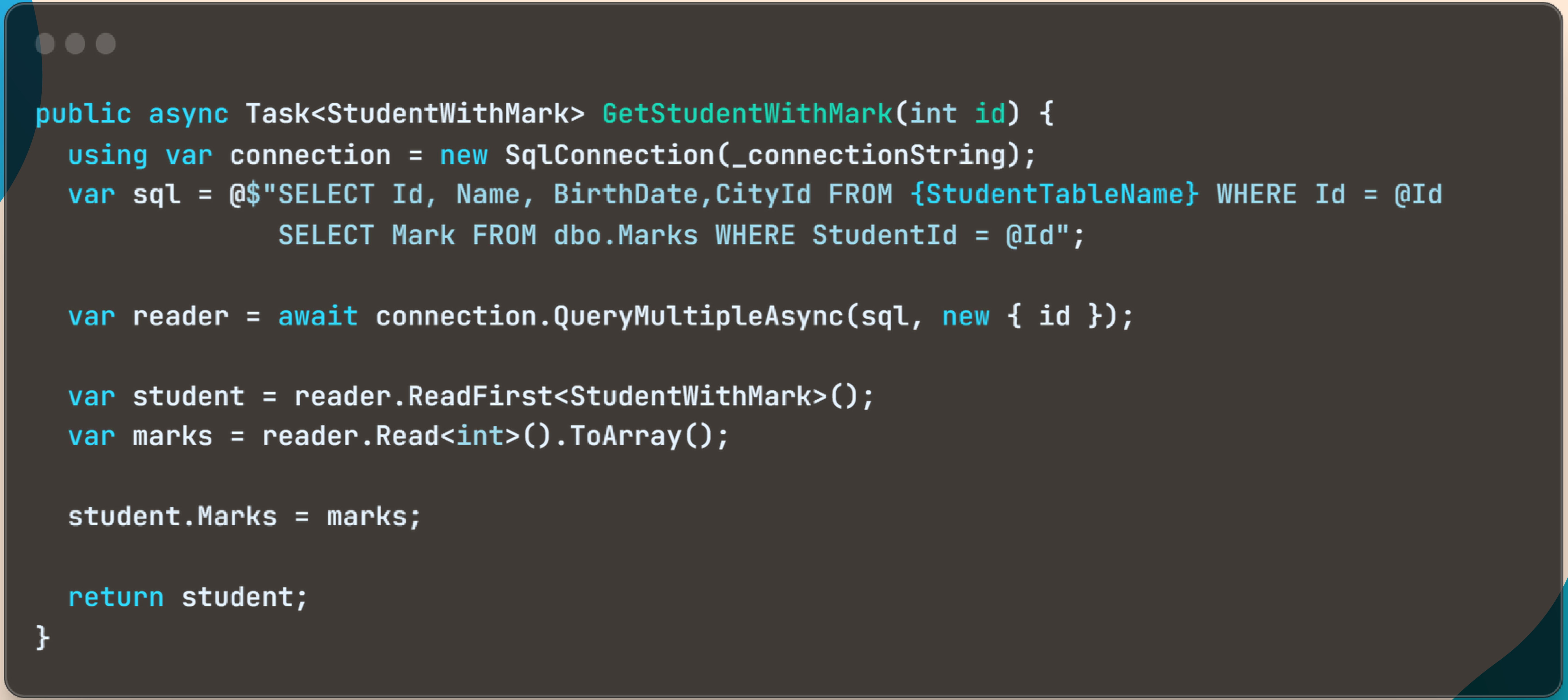
```
public async Task<Student[]> GetStudents() {

  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT Id, Name, BirthDate, CityId FROM {StudentTableName}";

  var students = await connection.QueryAsync<Student>(sql);

  return students.ToArray();
}
```

SCALAR VALUE



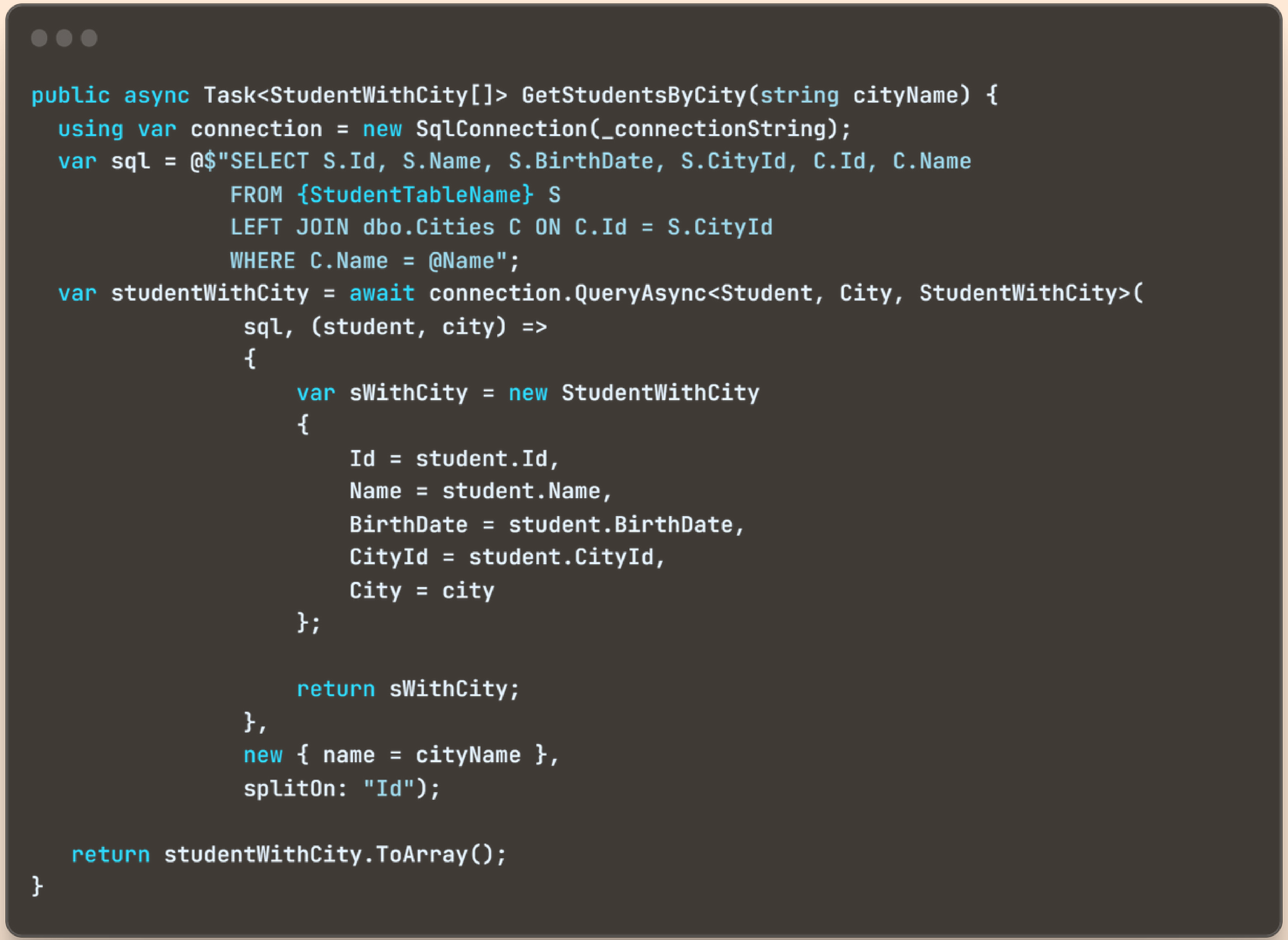
```
public async Task<int> GetTotalStudents() {
  using var connection = new SqlConnection(_connectionString);
  var sql = $"SELECT Count(*) FROM {StudentTableName}";

  var totalStudents = await connection.ExecuteScalarAsync<int>(sql);

  return totalStudents;
}
```

GET MORE TABLE



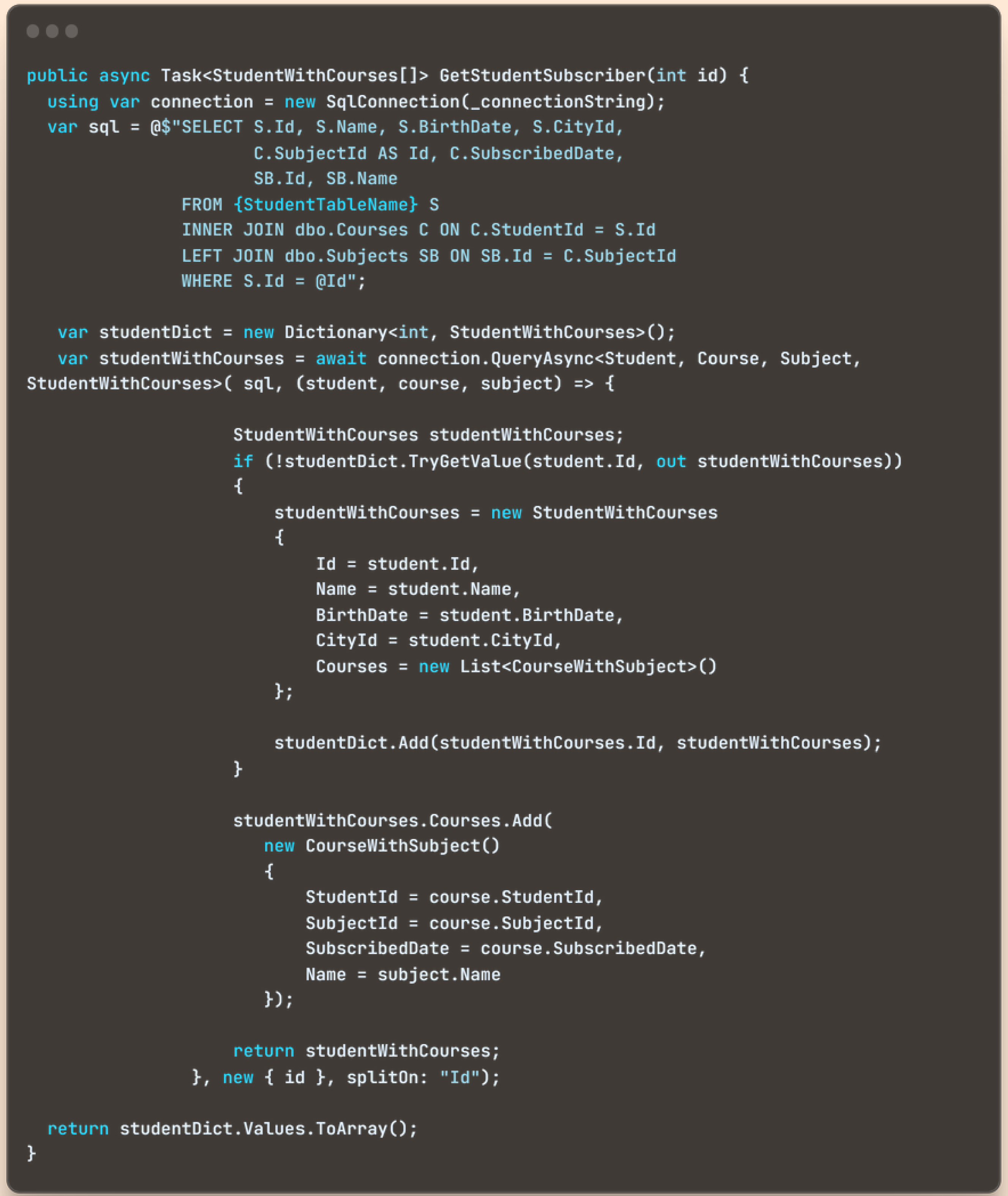
```
public async Task<StudentWithMark> GetStudentWithMark(int id) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT Id, Name, BirthDate,CityId FROM {StudentTableName} WHERE Id = @Id
               SELECT Mark FROM dbo.Marks WHERE StudentId = @Id";

  var reader = await connection.QueryMultipleAsync(sql, new { id });

  var student = reader.ReadFirst<StudentWithMark>();
  var marks = reader.Read<int>().ToArray();

  student.Marks = marks;

  return student;
}
```

RELAZIONE 1to1



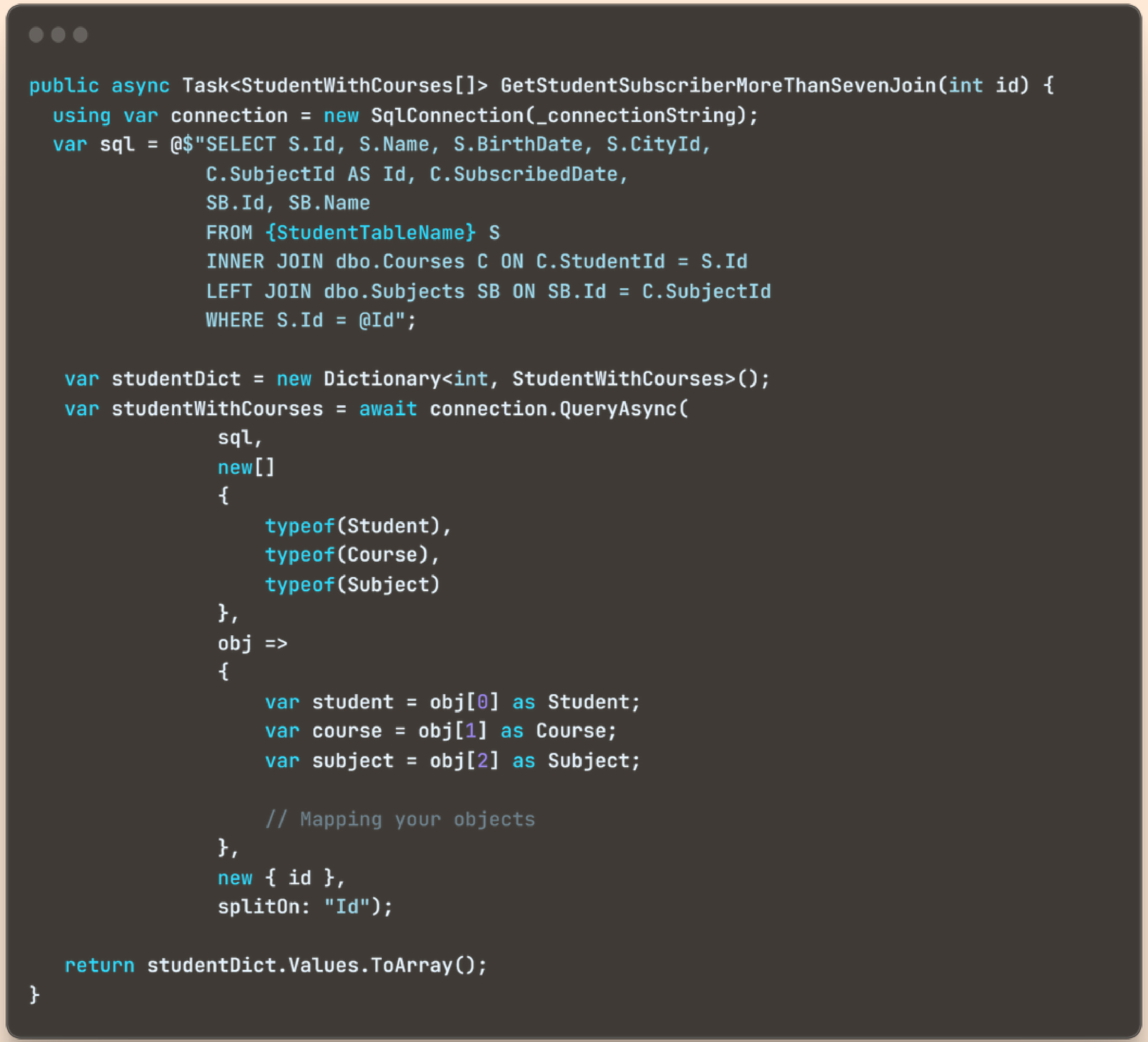
```
public async Task<StudentWithCity[]> GetStudentsByCity(string cityName) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT S.Id, S.Name, S.BirthDate, S.CityId, C.Id, C.Name
               FROM {StudentTableName} S
               LEFT JOIN dbo.Cities C ON C.Id = S.CityId
               WHERE C.Name = @Name";
  var studentWithCity = await connection.QueryAsync<Student, City, StudentWithCity>(
                sql, (student, city) =>
                {
                    var sWithCity = new StudentWithCity
                    {
                        Id = student.Id,
                        Name = student.Name,
                        BirthDate = student.BirthDate,
                        CityId = student.CityId,
                        City = city
                    };

                    return sWithCity;
                },
                new { name = cityName },
                splitOn: "Id");

   return studentWithCity.ToArray();
}
```

RELAZIONE 1toN



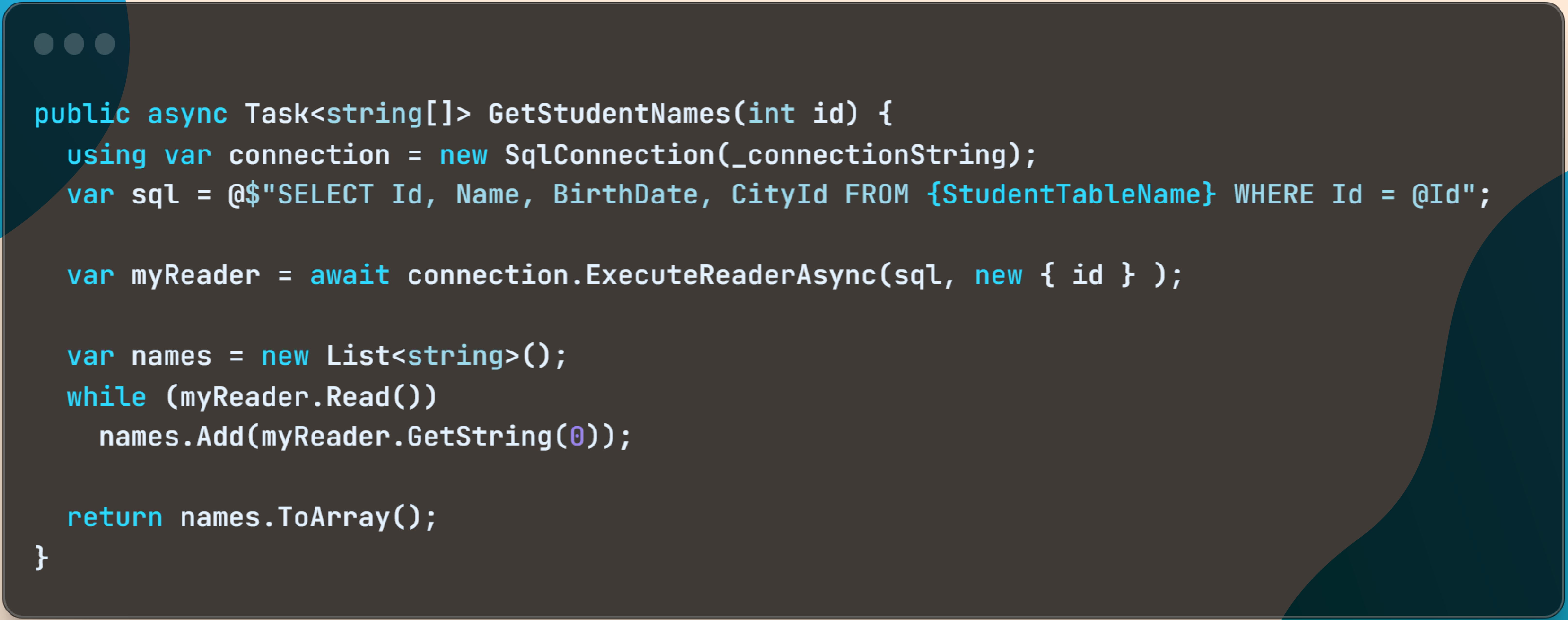
```
public async Task<StudentWithCourses[]> GetStudentSubscriber(int id) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT S.Id, S.Name, S.BirthDate, S.CityId,
                      C.SubjectId AS Id, C.SubscribedDate,
                      SB.Id, SB.Name
               FROM {StudentTableName} S
               INNER JOIN dbo.Courses C ON C.StudentId = S.Id
               LEFT JOIN dbo.Subjects SB ON SB.Id = C.SubjectId
               WHERE S.Id = @Id";

   var studentDict = new Dictionary<int, StudentWithCourses>();
   var studentWithCourses = await connection.QueryAsync<Student, Course, Subject,
StudentWithCourses>( sql, (student, course, subject) => {

                    StudentWithCourses studentWithCourses;
                    if (!studentDict.TryGetValue(student.Id, out studentWithCourses))
                    {
                        studentWithCourses = new StudentWithCourses
                        {
                            Id = student.Id,
                            Name = student.Name,
                            BirthDate = student.BirthDate,
                            CityId = student.CityId,
                            Courses = new List<CourseWithSubject>()
                        };

                        studentDict.Add(studentWithCourses.Id, studentWithCourses);
                    }

                    studentWithCourses.Courses.Add(
                       new CourseWithSubject()
                       {
                           StudentId = course.StudentId,
                           SubjectId = course.SubjectId,
                           SubscribedDate = course.SubscribedDate,
                           Name = subject.Name
                       });

                    return studentWithCourses;
                }, new { id }, splitOn: "Id");

  return studentDict.Values.ToArray();
}
```

MAPPING PIU DI 7 JOIN



```csharp
public async Task<StudentWithCourses[]> GetStudentSubscriberMoreThanSevenJoin(int id) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT S.Id, S.Name, S.BirthDate, S.CityId,
             C.SubjectId AS Id, C.SubscribedDate,
             SB.Id, SB.Name
             FROM {StudentTableName} S
             INNER JOIN dbo.Courses C ON C.StudentId = S.Id
             LEFT JOIN dbo.Subjects SB ON SB.Id = C.SubjectId
             WHERE S.Id = @Id";

   var studentDict = new Dictionary<int, StudentWithCourses>();
   var studentWithCourses = await connection.QueryAsync(
              sql,
              new[]
              {
                  typeof(Student),
                  typeof(Course),
                  typeof(Subject)
              },
              obj =>
              {
                  var student = obj[0] as Student;
                  var course = obj[1] as Course;
                  var subject = obj[2] as Subject;

                  // Mapping your objects
              },
              new { id },
              splitOn: "Id");

   return studentDict.Values.ToArray();
}
```

EXECUTE READER



```csharp
public async Task<string[]> GetStudentNames(int id) {
  using var connection = new SqlConnection(_connectionString);
  var sql = @$"SELECT Id, Name, BirthDate, CityId FROM {StudentTableName} WHERE Id = @Id";

  var myReader = await connection.ExecuteReaderAsync(sql, new { id } );

  var names = new List<string>();
  while (myReader.Read())
    names.Add(myReader.GetString(0));

  return names.ToArray();
}
```

STORE PROCEDURE

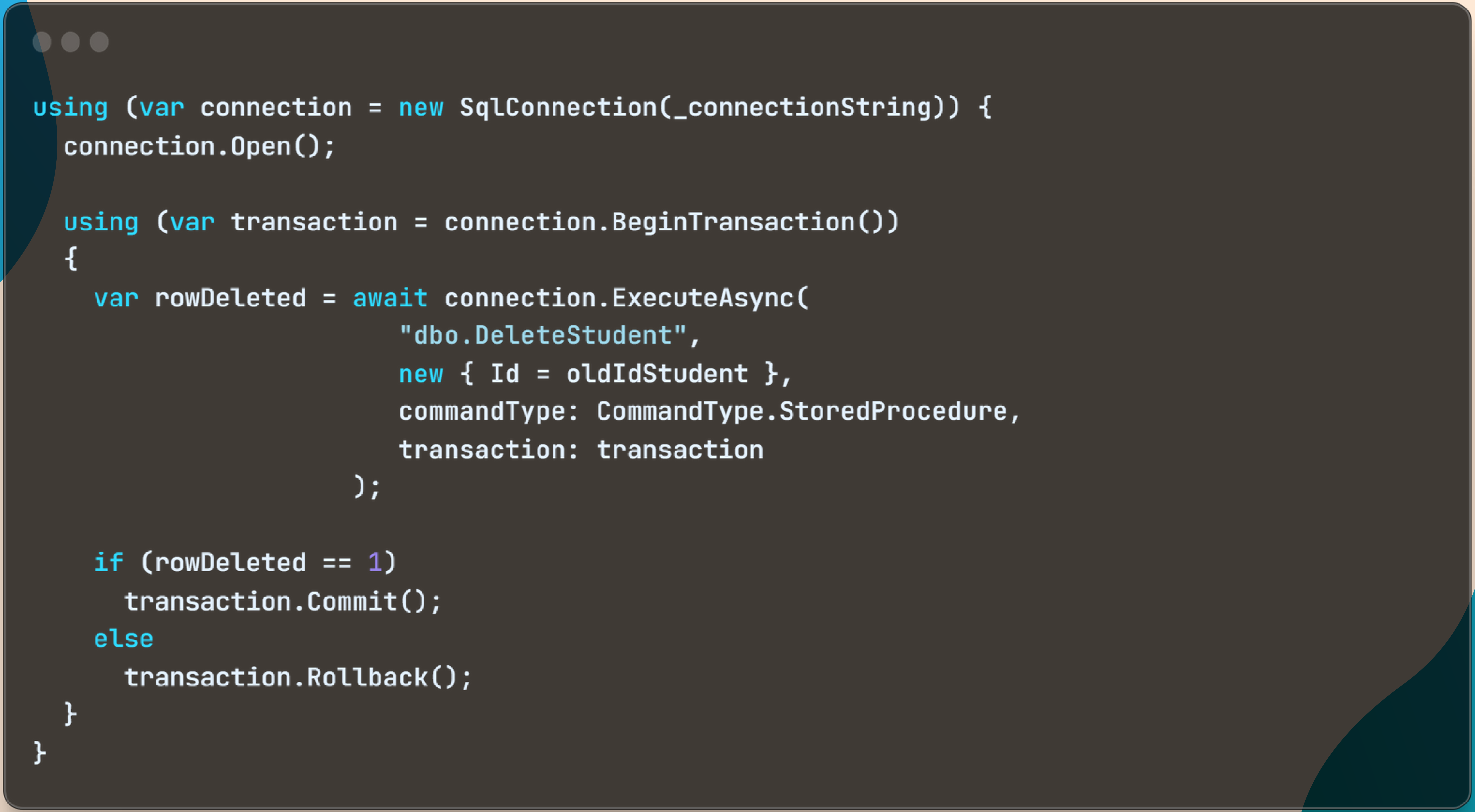
Per l'esecuzione di una store procedure, si possono usare i metodi precendenti, in questo modo:

- Anzichè scrivere la query, inserire il nome della SP
- Specificare il parametro: commandType: CommandType.StoredProcedure

PARAMETRI EXTA

- **commandTimeout** (default 30 secondi oppure use quello specificato nella stringa di connession)
- **commandType** si può specificare se stiamo eseguendo una query scritta come testo (default), store procedure o tabella (supportata solo da connessioni OLE DB)
- **transaction**
- **buffered**

TRANSACTION



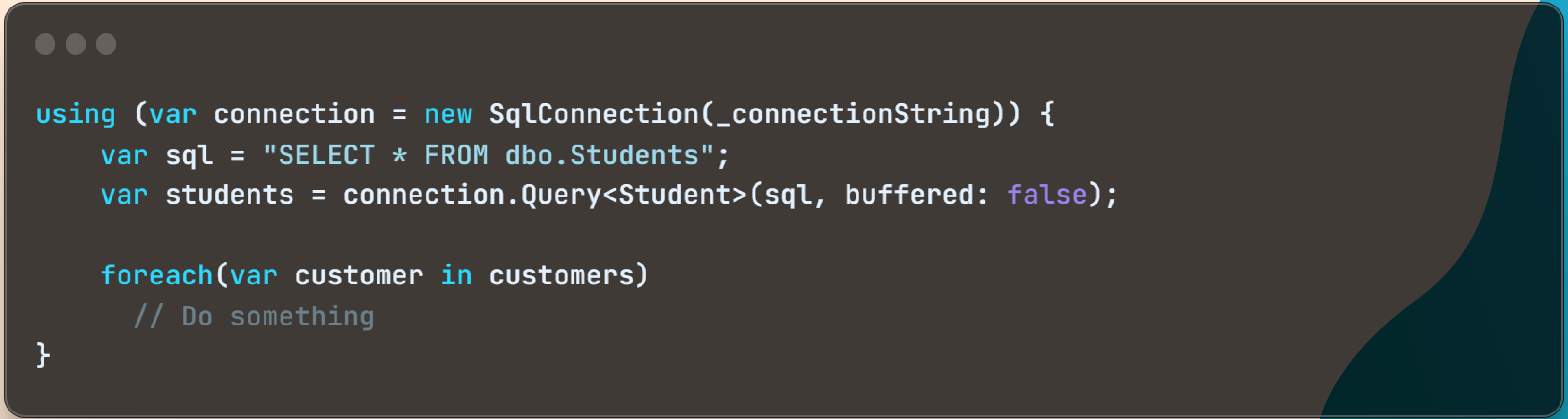
```
using (var connection = new SqlConnection(_connectionString)) {
  connection.Open();

  using (var transaction = connection.BeginTransaction())
  {
    var rowDeleted = await connection.ExecuteAsync(
                        "dbo.DeleteStudent",
                        new { Id = oldIdStudent },
                        commandType: CommandType.StoredProcedure,
                        transaction: transaction
                     );

    if (rowDeleted == 1)
      transaction.Commit();
    else
      transaction.Rollback();
  }
}
```

BUFFER / UNBUFFERED

Specificando come pametro

- buffer a true recupera tutte le righe contemporaneamente nella memoria.
  Il vantaggio è quello che recupero i dati molto più velocemente, ma consumo più memoria

- se è a false recuperiamo i risultati uno per uno quando vengono richiesti.



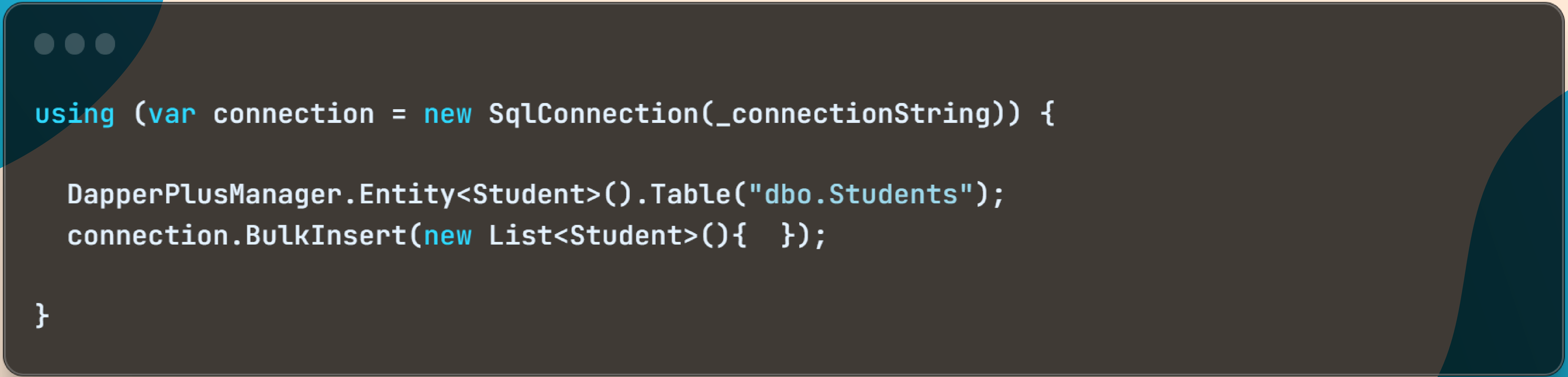
```
using (var connection = new SqlConnection(_connectionString)) {
    var sql = "SELECT * FROM dbo.Students";
    var students = connection.Query<Student>(sql, buffered: false);

    foreach(var customer in customers)
      // Do something
}
```

ESTENSIONI DAPPER

- **DAPPERPLUS** (https://dapper-plus.net/), estende IDbConnection per migliorare le prestazione per grandi quantità di dati. (Serve la licenza)

- **CONTRIB** (https://github.com/DapperLib/Dapper.Contrib) contiene una serie di metodi di supporto per l'insert, select, update e delete

DAPPERPLUS



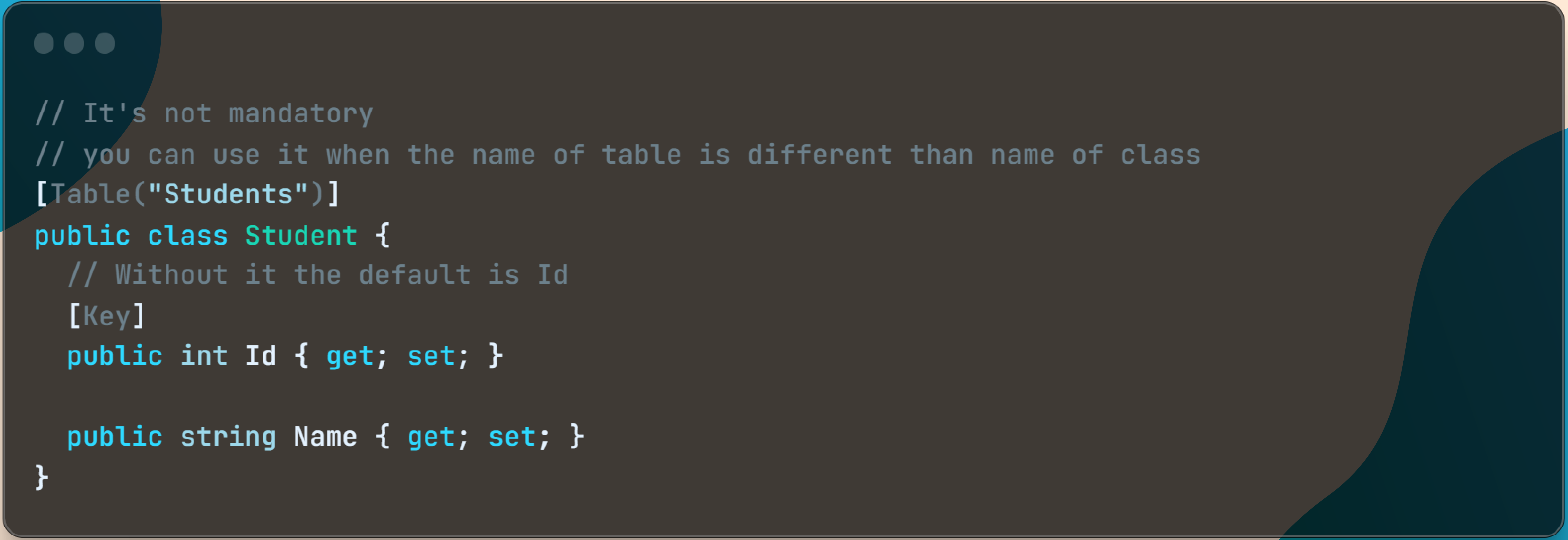
```
using (var connection = new SqlConnection(_connectionString)) {

  DapperPlusManager.Entity<Student>().Table("dbo.Students");
  connection.BulkInsert(new List<Student>(){  });

}
```

CONTRIB

Metodi disponibili che vanno ad estendere la IDBConnection

- **GET** possiamo specificare l'id per filtrare oppure avere tutti i dati
- **INSERT** al metodo posso passarli un solo oggetto o un Enumerable
- **UPDATE** al metodo posso passarli un solo oggetto o un Enumerable, usa l'id per sapere chi deve modificare
- **DELETE** al metodo posso passarli un solo oggetto o un Enumerable, usa l'id per sapere chi deve cancellare

CONTRIB (Classe)



```
// It's not mandatory
// you can use it when the name of table is different than name of class
[Table("Students")]
public class Student {
  // Without it the default is Id
  [Key]
  public int Id { get; set; }

  public string Name { get; set; }
}
```

CONCLUSIONI

- Uso per le applicazioni che richiedono un accesso ai dati semplici
- Facile da usere per tutti
- Performante
- Utilizzabile con tutto il mondo .NET

QUESTE LE SLIDE LE PUOI TROVARE SU

https://github.com/FabioDeveloper92/Talk/tree/main/NETCommunityTorino/DapperIntro