

# Лекция 9

## Обработка исключений.

### Введение в БД



# try/catch

```
try{...}  
catch{...}  
finally{...}
```

- Вначале выполняются все инструкции в блоке try. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок finally. И затем конструкция try..catch..finally завершает свою работу.
- Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок catch, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.
- Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение



# Finally не срабатывает в случаях

- При завершении работы программы в блоке catch
- При отсутствии блока catch
- При возникновении (не программной генерации!) исключений
  - StackOverflowException
  - ExecutionEngineException



# Генерация исключения

```
throw [e];
```

где e — это экземпляр класса, производного от System.Exception.

```
public int GetNumberByIndex(int index){  
    if (index < 0 || index >= numbers.Length){  
        throw new IndexOutOfRangeException();  
    }  
    return numbers[index];  
}  
...  
try{  
    var res = GetNumberByIndex(10)  
}  
catch (IndexOutOfRangeException e){  
    ...  
}
```




# Класс Exception

- Класс включает ряд свойств, помогающих определить расположение кода, тип, файл справки и причину исключения:
  - StackTrace – стек вызова
  - InnerException – вызвавшее исключение
  - Message – строка сообщения
  - HelpLink – URL-ссылка
  - HRESULT - кодированное 32-разрядное значение, разделенное на три разных поля: код серьезности, код средства и код ошибки
  - Source – приложение (или объект), вызвавшее исключение
  - TargetSite – метод, вызвавший исключение
  - Data – словарь дополнительных значений
- Конструкторы
  - Exception()
  - Exception(String)
  - Exception(String, Exception)







```
LogTable log = new LogTable( 4 );
try
{
    for( int count = 1; ; count++ )
    {
        log.AddRecord(
            String.Format(
                "Log record number {0}", count ) );
    }
}
catch( Exception ex )
{
    Console.WriteLine( "\nMessage ---\n{0}", ex.Message );
    Console.WriteLine( "\nHelpLink ---\n{0}", ex.HelpLink );
    Console.WriteLine( "\nSource ---\n{0}", ex.Source );
    Console.WriteLine( "\nStackTrace ---\n{0}", ex.StackTrace );
    Console.WriteLine( "\nTargetSite ---\n{0}", ex.TargetSite );
}
```

Message ---  
The log table has overflowed. - Record "Log record number 5" was not logged.

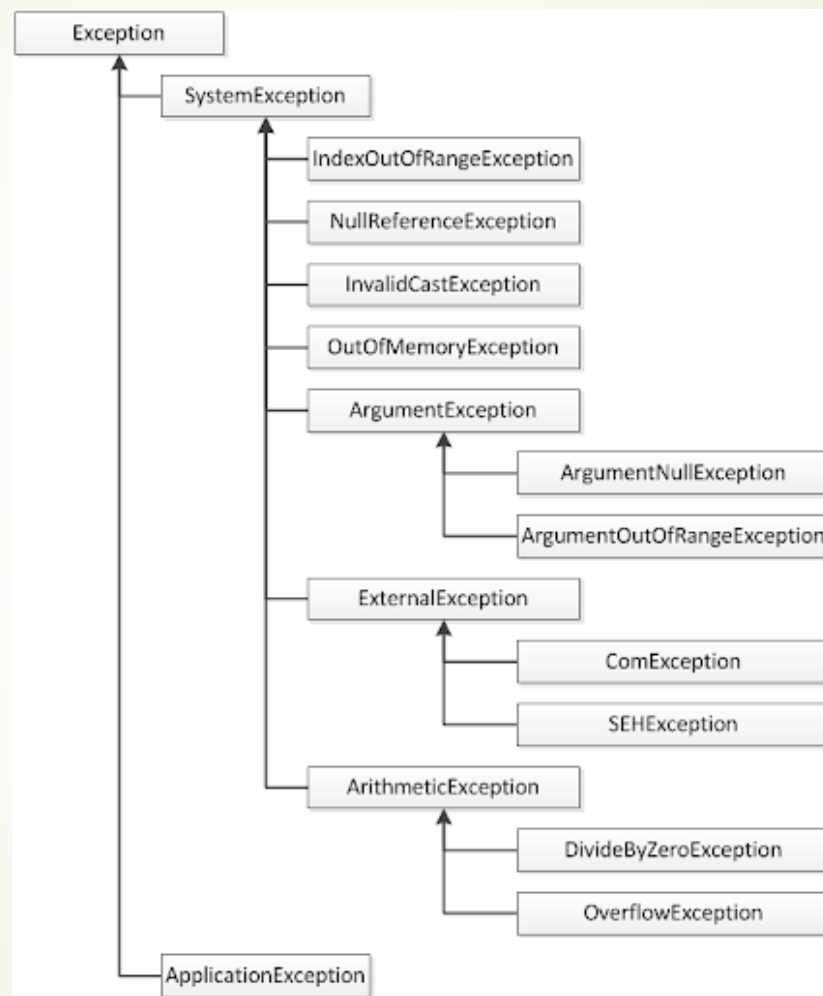
HelpLink ---  
<https://docs.microsoft.com>

Source ---  
Exception\_Class\_Samples

StackTrace ---  
at NDP\_UE\_CS.LogTable.AddRecord(String newRecord)  
at NDP\_UE\_CS.OverflowDemo.Main()

TargetSite ---  
Int32 AddRecord(System.String)

# Классы исключений








# Фильтрация исключений

```
catch (ArgumentException e) when (e.ParamName == "..."){...}
```

Фильтры исключений предпочтительнее перехвата и повторного вызова, поскольку фильтры оставляют стек в целости и сохранности. Если последующий обработчик обращается к стеку, вы можете увидеть, откуда изначально произошло исключение, а не только последнее место, в котором оно было повторно вызвано. Обычно выражения фильтра исключений используются для ведения журнала. Вы можете создать фильтр, который всегда возвращает значение false, а также записывает выходной результат в журнал, чтобы регистрировать исключения в журнале по мере их поступления без необходимости их обработки и повторного вызова

# ИСКЛЮЧЕНИЯ И МНОГОПОТОЧНОСТЬ

```
private static void ProcessDataInParallel(byte[] data) {  
    var exceptions = new ConcurrentQueue<Exception>();  
    Parallel.ForEach(data, d => {  
        try{  
            if (d < 3) throw new ArgumentException($"Value is {d}. Value must be greater than or equal to 3.");  
            else Console.Write(d + " ");  
        }  
        catch (Exception e){  
            exceptions.Enqueue(e);  
        }  
    });  
    if (exceptions.Count > 0) throw new AggregateException(exceptions);  
}  
...  
try{  
    ProcessDataInParallel(data);  
}  
catch (AggregateException ae){  
    foreach (var ex in ae.Flatten().InnerExceptions)  
        Console.WriteLine(ex.Message);  
}
```



```
public async Task DoMultipleAsync(){
    var theTask1 = ExcAsync(info: "First Task");
    var theTask2 = ExcAsync(info: "Second Task");
    var theTask3 = ExcAsync(info: "Third Task");
    var allTasks = Task.WhenAll(theTask1, theTask2, theTask3);
    try{
        await allTasks;
    }
    catch (Exception ex){
        Debug.WriteLine("Exception: " + ex.Message);
        Debug.WriteLine("Task IsFaulted: " + allTasks.IsFaulted);
        foreach (var inEx in allTasks.Exception.InnerExceptions){
            Debug.WriteLine("Task Inner Exception: " + inEx.Message);
        }
    }
}

private async Task ExcAsync(string info){
    await Task.Delay(100);
    throw new Exception("Error-" + info);
}
```



# УСЛОВНЫЕ КОНСТРУКЦИИ

```
if (Int32.TryParse(input, out x)){  
    x *= x;  
    Console.WriteLine("Квадрат числа: " + x);  
}  
else{  
    Console.WriteLine("Некорректный ввод");  
}
```

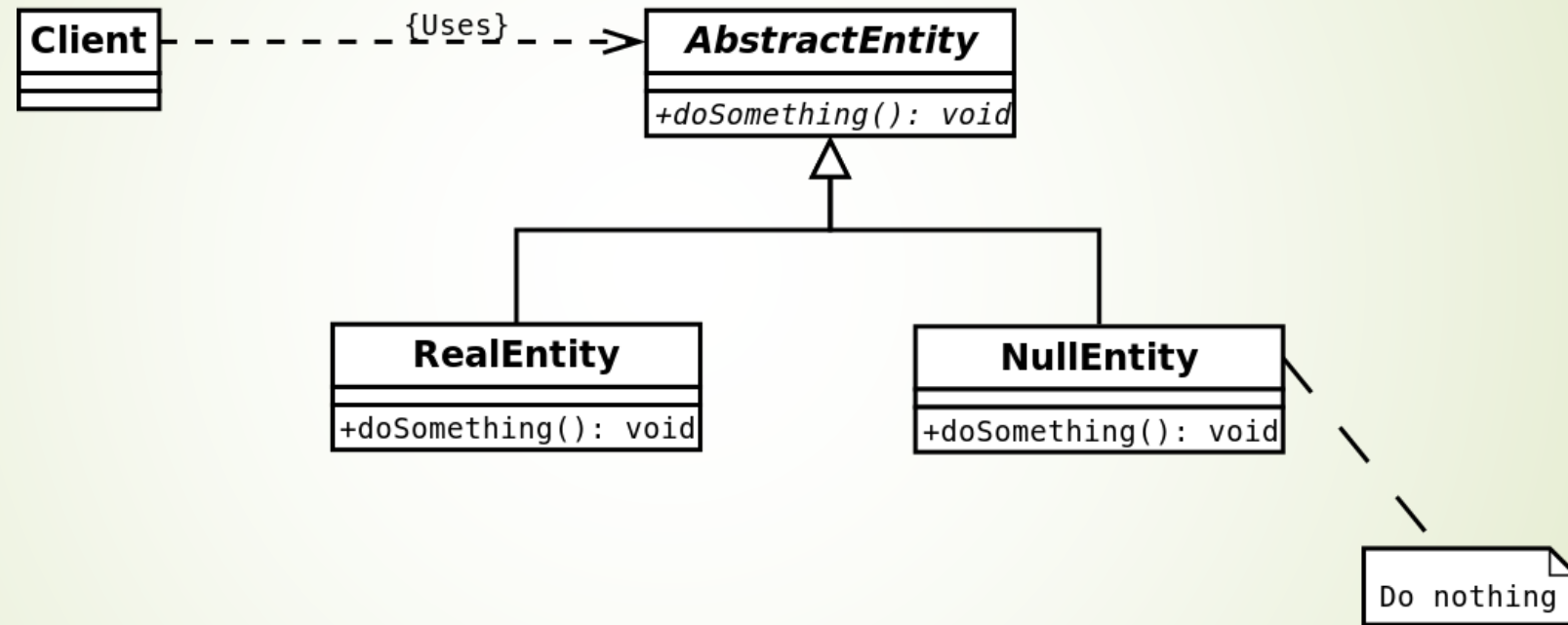
Аналогично работают методы Double.TryParse, Byte.TryParse, SByte.TryParse, Enum.TryParse, Guid. TryParse, Half. TryParse, Int16. TryParse, Int32. TryParse, Int64.TryParse, UInt32.TryParse, UInt32.TryParse UInt32.TryParse, Single.TryParse, IntPtr.TryParse, Decimal. TryParse



# Try-pattern

```
public bool TryGetUser(Guid id, out User user){  
    user = users.Find(u => u.Id == id)  
    if (user){  
        return true  
    }  
    return false;  
}
```

# Null object-pattern





# Maybe монада

```
public Maybe<User> TryGetUser(Guid id){  
    user = users.Find(u => u.Id == id)  
    if (user){  
        return Maybe<User>.CreateSuccess(user)  
    }  
    return Maybe<User>.CreateFailure();  
}
```

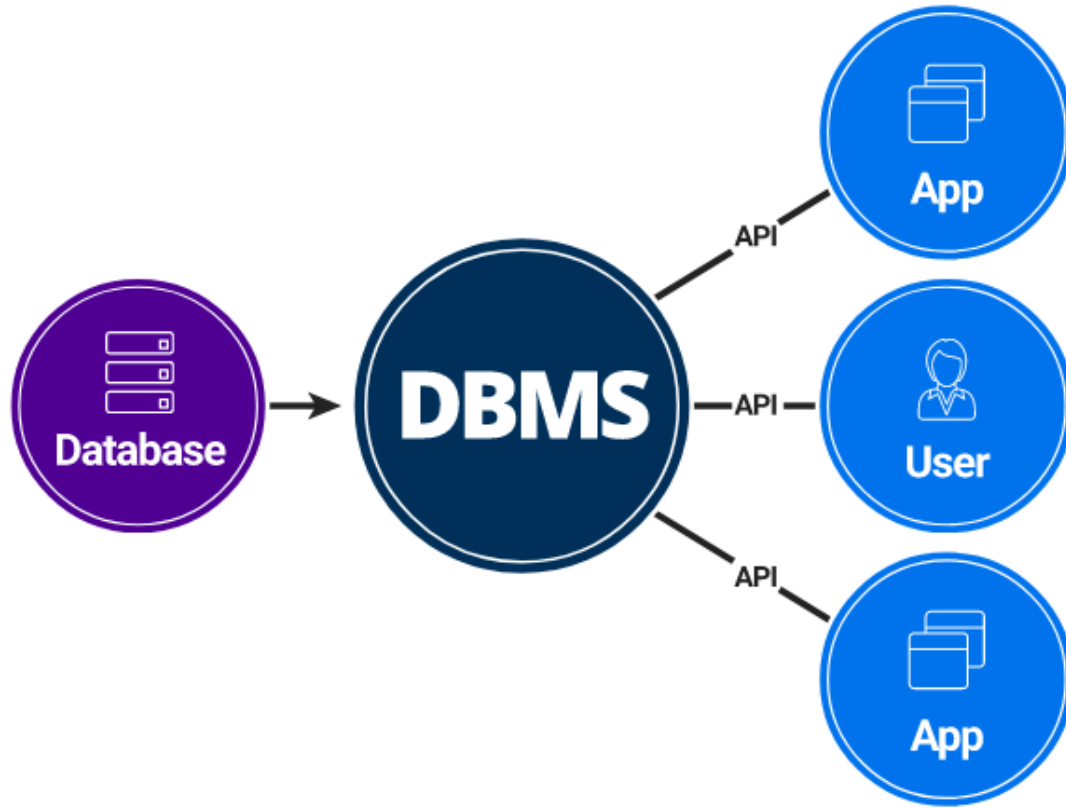




# Недостатки использования файловых хранилищ в прикладных приложениях

- Обеспечение многопользовательского доступа к данным
- Стандартизированный доступ из нескольких клиентских приложений
- Разграничение прав доступа различным пользователям
- Контроль целостности и непротиворечивости данных
- Отказоустойчивость хранилища
- Быстрый поиск данных

# Database Management Systems



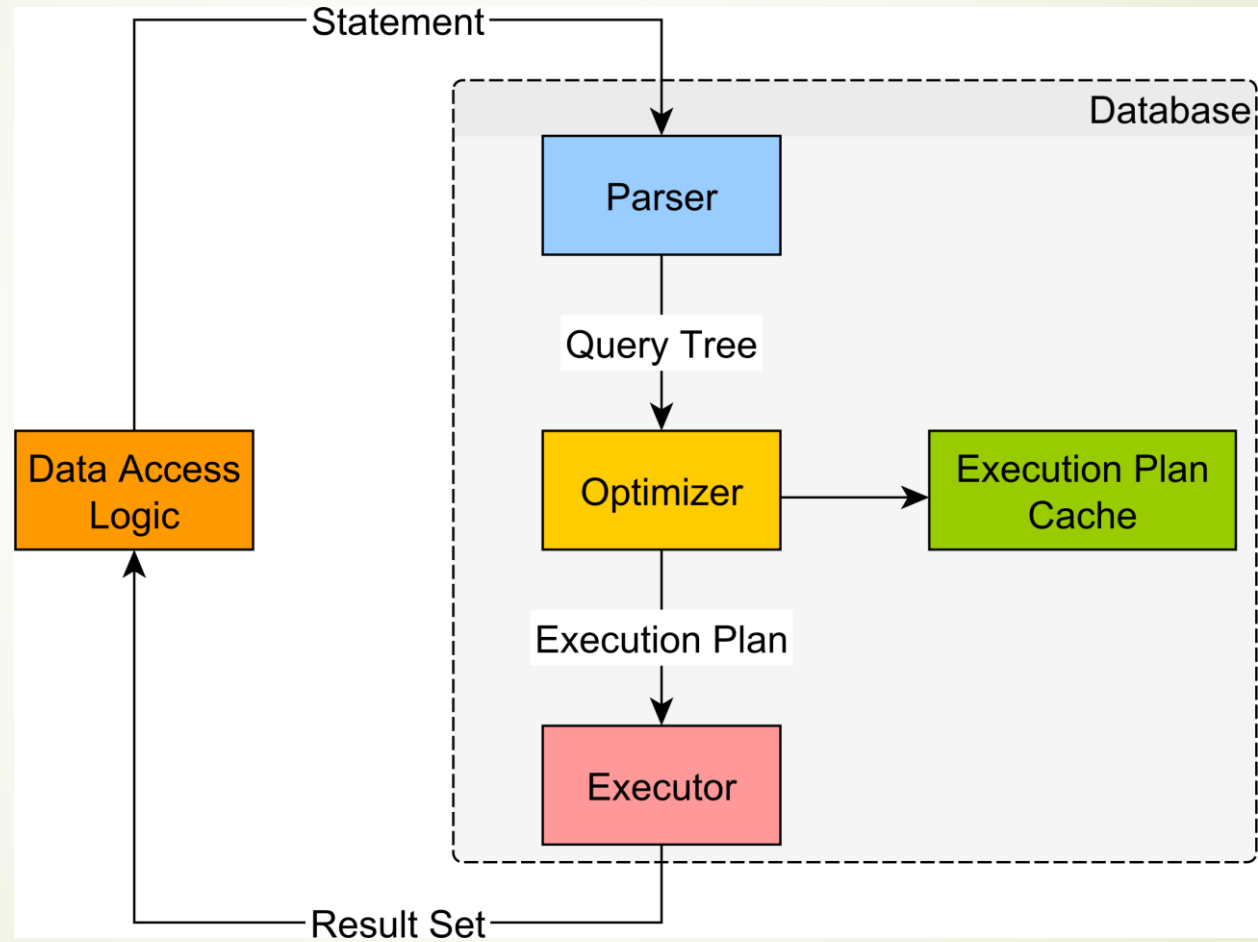




# SQL



- Structured Query Language (язык структурированных запросов) — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных:
- Data Definition Language, DDL - операторы определения данных
- Data Manipulation Language, DML - операторы манипуляции данными
- Data Control Language, DCL - операторы определения доступа к данным
- Transaction Control Language, TCL - операторы управления транзакциями





# DDL



```
create table client (  
    client_id INT(11) NOT NULL,  
    client_name VARCHAR(40),  
    PRIMARY KEY(client_id)  
);
```

```
create table service (  
    service_id INT(11) NOT NULL,  
    service_name VARCHAR(40),  
    service_client_id INT(11),  
    PRIMARY KEY(service_id),  
    CONSTRAINT service_fk1 FOREIGN KEY(service_client_id) REFERENCES client(client_id)  
);
```

```
DROP TABLE user;
```





# DML

➤ DML:

- INSERT

```
INSERT INTO <table_name> (<col_name1>, <col_name2>, ...)
VALUES (<value1>, <value2>, ...);
```

- SELECT

- SELECT <col\_name1>, <col\_name2>, ... FROM <table\_name>
- WHERE <condition> ORDER BY <col\_name>, ...;

- UPDATE

```
UPDATE <table_name> SET <col_name1> = <value1>, ...
WHERE <condition>;
```

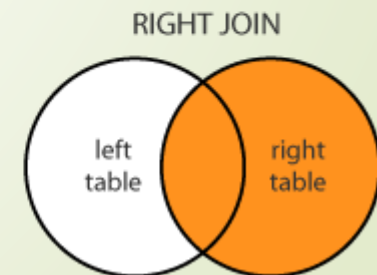
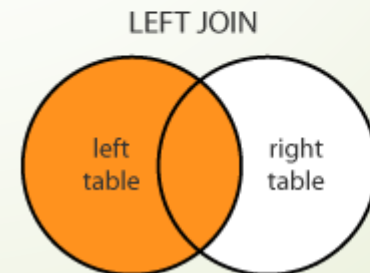
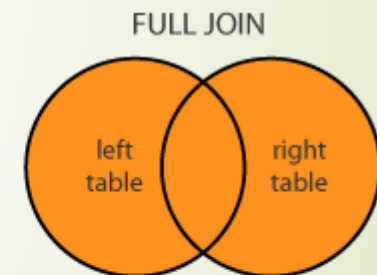
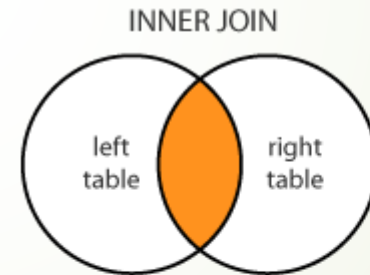
- DELETE

```
DELETE FROM <table_name> WHERE <condition>;
```



# Соединения таблиц

```
SELECT client_id, service_name  
FROM client  
INNER JOIN service ON client_id = service_client_id  
WHERE client_id IN (3, 5, 7)  
ORDER BY service_name
```





# Агрегатные функции

➤ MIN(), MAX(), COUNT(), AVG(), SUM()

```
SELECT COUNT(service_name) AS cnt, client_id  
FROM client  
INNER JOIN service ON client_id = service_client_id  
GROUP BY client_id  
HAVING cnt > 2
```

# CRUD-системы

➤ CRUD: Create, Read, Update, Delete

## CRUD Database

ID	First	Last	Email	Phone	Location	Hobby	Actions
35	Jane	Smith	js@gmail.com	503-555-5555	Portland, Oregon	Photography	<a href="#">Edit</a> <a href="#">Del</a>
36	Tristan	Rodriguez	tr@gmail.com	201-555-5555	New York, New York		<a href="#">Edit</a> <a href="#">Del</a>
37	Andrea	Jones	aj@gmail.com	503-555-5555	Portland, Oregon	Programming	<a href="#">Edit</a> <a href="#">Del</a>
38	Elliott	Richardson	er@gmail.com		Boise, Idaho		<a href="#">Edit</a> <a href="#">Del</a>
39	June	Bug	jb@gmail.com			Cooking	<a href="#">Edit</a> <a href="#">Del</a>

[Download CSV](#)[Add Item](#)