Marshmallow de uma forma diferente

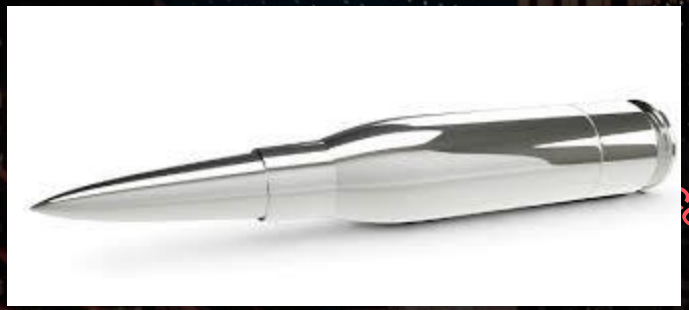
{ Palestrante

Juliomar Marchetti

Embarcadero Conference 2023

O que você achou da palestra?

Acesse o link do QR Code ao lado e responda a pesquisa.



Conference 2023

O que é ORM?

Object-Relational Mapping (ORM), em português, mapeamento objeto-relacional, é uma técnica para aproximar o paradigma de desenvolvimento de aplicações orientadas a objetos ao paradigma do banco de dados relacional. O uso da técnica de mapeamento objeto-relacional é realizado através de um mapeador objeto-relacional que geralmente é a biblioteca ou framework que ajuda no mapeamento e uso do banco de dados.

Onde acho ?

https://bitbucket.org/sglienke/spring4d/src/master/

https://bitbucket.org/soundvibe/marshmallow/wiki/Home

Onde acho ?

https://bitbucket.org/sglienke/spri

https://bitbucket.org/soundvibe/m

Wiki

Criar pág

Marshmallow / Home

Project "Marshmallow". Modern ORM/OPF framework for Delphi

SQL Server SQLite Firebird ORACLE MySQL MongoDB

Background

Project "Marshmallow" was inspired by .NET micro ORM's (mostly by PetaPoco) and Java Hibernate. The mair
language features, including generics, attributes, enhanced RTTI, records, operator overloading, etc. This allo

Features

- Works with attribute decorated PODO's (Plain Old Delphi Object)
- Can create and/or update database tables from PODOs
- Helper methods for Insert/Delete/Update/Save and IsNew

Onde acho ?

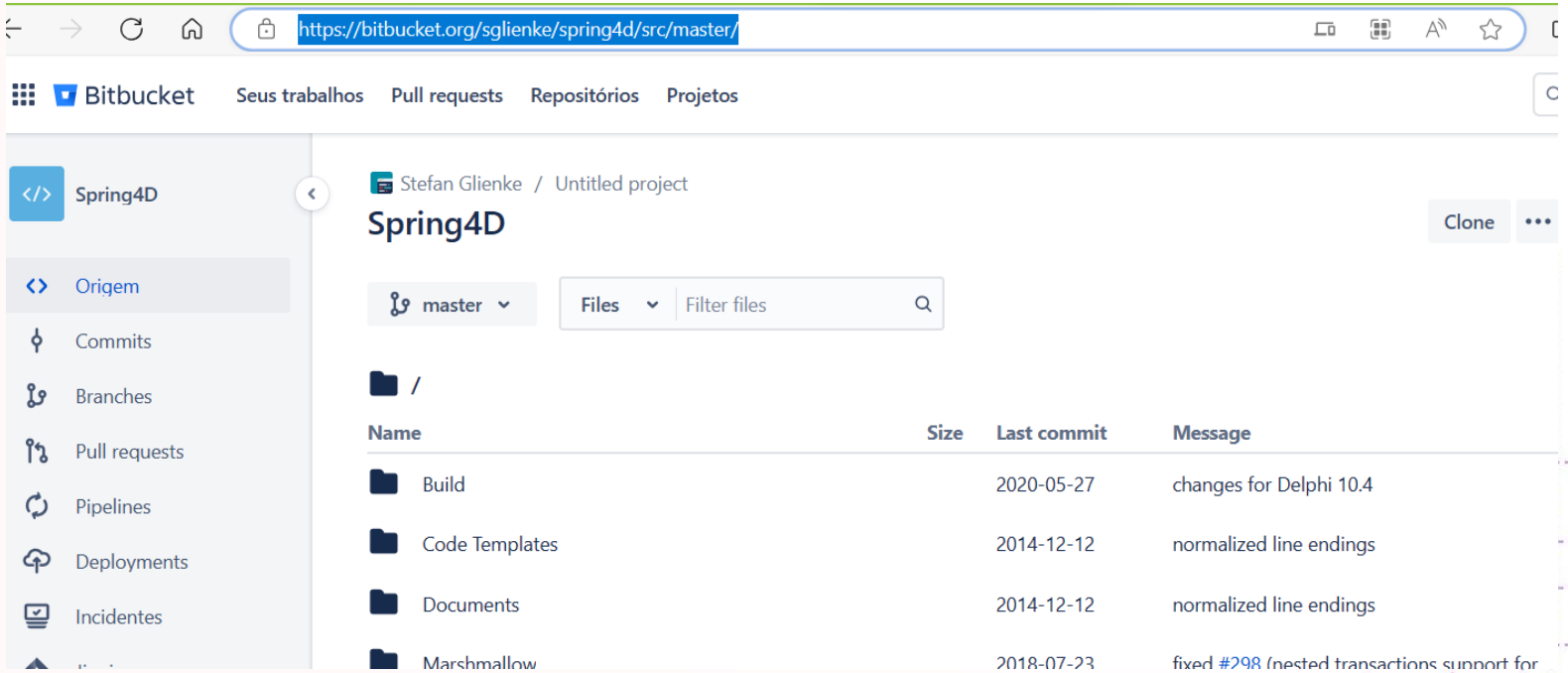
https://bitbucket.org/sglienke/spring4d/src/master/

https://bitbucket.org/soundvibe/marshmallow/wiki/Home

bind images to table blob fields. Engine will detect image type from stream and load it to
ith SQLite, Sybase ASA, SQL Server, Firebird, Oracle, MySQL, PostgreSQL, MongoDB but oth

Onde acho ?

https://bitbucket.org/sglienke/spring4d/src/master/



Onde acho ?

https://bitbucket.org/sglienke/spring4d/src/master/

spring4d / **Source**

Name	Size	Last commit	Message
..			
Base		2020-06-19	replaced workaround for RSP-20683 with a class var - avoi...
Core		2020-05-27	changes for Delphi 10.4
Data/ObjectDataSet		2021-09-09	Delphi 11 support
Extensions		2022-09-07	fixed defect with address calculation
Persistence		2018-07-23	fixed #298 (nested transactions support for FireDAC)
Spring.inc	5.26 KB	2021-05-28	fixed compile error on OSX64

Onde acho ?

https://bitbucket.org/sglienke/spring4d/src/master/

spring4d / Source / **Persistence**

Name	Size	Last commit	Message
..			
Adapters		2018-07-23	fixed #298 (nested transactions support for FireDAC)
Core		2018-02-01	fixed compile errors on XE2-XE4
Criteria		2018-01-29	updated copyright
Mapping		2018-01-29	updated copyright
SQL		2018-01-29	updated copyright

Onde a

https://bitbucket.c

spring4d / Source / Persistence / **Adapters**

Name	Size	Last commit	Message
..			
Spring.Persistence.Adapters.ADO.pas	8.98 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.ASA.pas	3.91 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.DBX.pas	8.27 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.FieldCache.pas	3.1 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.FireDAC.pas	9.21 KB	2018-07-23	fixed #298 (nested transactions support for...
Spring.Persistence.Adapters.MSSQL.pas	4.39 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.MongoDB.pas	17.81 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.Oracle.pas	4.79 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.SQLite.pas	9.77 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.UIB.pas	9.98 KB	2018-01-29	updated copyright
Spring.Persistence.Adapters.Zeos.pas	8.4 KB	2018-01-29	updated copyright

Onde acho ?

https://bitbucket.org/sglienke/spring

Spring.Persistence.Core.AbstractSession.pas	23.41 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Base.pas	15.99 KB	2018-01-29	updated copyright
Spring.Persistence.Core.ConnectionFactory.pas	7.06 KB	2018-01-29	updated copyright
Spring.Persistence.Core.DatabaseManager.pas	5.77 KB	2018-01-29	updated copyright
Spring.Persistence.Core.DetachedSession.pas	2.8 KB	2018-01-29	updated copyright
Spring.Persistence.Core.EmbeddedEntity.pas	7.52 KB	2018-01-29	updated copyright
Spring.Persistence.Core.EntityCache.pas	14.71 KB	2018-01-29	updated copyright
Spring.Persistence.Core.EntityMap.pas	8.73 KB	2018-01-29	updated copyright
Spring.Persistence.Core.EntityWrapper.pas	6.48 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Exceptions.pas	10.81 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Graphics.pas	7.53 KB	2018-02-01	fixed compile errors on XE2-XE4
Spring.Persistence.Core.Interfaces.pas	13.8 KB	2018-01-29	updated copyright
Spring.Persistence.Core.ListSession.pas	4.04 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Repository.MongoDB.pas	2.97 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Repository.Proxy.pas	10.51 KB	2018-01-29	updated copyright
Spring.Persistence.Core.Repository.Simple.pas	6.46 KB	2018-01-29	updated copyright

Onde a

https://bitbucket.

Spring.Persistence.Criteria.Criterion.Abstract.pas	3.53 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.BetweenExpressio...	4.14 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.Conjunction.pas	2.18 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.Disjunction.pas	2.18 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.InExpression.pas	4.57 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.Junction.pas	3.54 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.LikeExpression.pas	3.46 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.LogicalExpression....	3.77 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.NullExpression.pas	3.32 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.PropertyExpressio...	4.12 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Criterion.SimpleExpression....	3.74 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Interfaces.pas	9.23 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.OrderBy.pas	3.46 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Properties.pas	19.62 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.Restrictions.pas	13.99 KB	2018-01-29	updated copyright
Spring.Persistence.Criteria.pas	5.16 KB	2018-01-29	updated copyright

Onde acho ?

https://bitbucket.org/sglienke/spring4d/src/master/

spring4d / Source / Persistence / **Mapping**

Name	Size	Last commit	Message
..			
Spring.Persistence.Mapping.Attributes.pas	19.88 KB	2018-01-29	updated copyright
Spring.Persistence.Mapping.CodeGenerator.Abstract.pas	4.92 KB	2018-01-29	updated copyright
Spring.Persistence.Mapping.CodeGenerator.DB.pas	12.94 KB	2018-01-29	updated copyright
Spring.Persistence.Mapping.CodeGenerator.pas	9.24 KB	2018-01-29	updated copyright

Onde ac

https://bitbucket.org

spring4d / Source / Persistence / **SQL**

Name	Size	Last commit	Message
..			
Spring.Persistence.SQL.Commands.Abstract.pas	5.2 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.BulkInsert.MongoD...	3.61 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.CreateForeignKey.p...	3.74 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.CreateSequence.pas	4.01 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.CreateTable.pas	3.75 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.Delete.pas	4.47 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.Insert.pas	8.57 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.Page.pas	3.09 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.Select.pas	5.5 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.Update.pas	5.36 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Commands.pas	18.45 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Generators.ASA.pas	3.88 KB	2018-01-29	updated copyright
Spring.Persistence.SQL.Generators.Abstract.pas	5.65 KB	2018-01-29	updated copyright

Marshmallow

Code examples

Very simple PODO example:

```
[Entity]
[Table('Customers')]
TCustomer = class
private
  [Column('CUSTID', [cpRequired, cpPrimaryKey])][AutoGenerated]
  FId: Integer;
  FName: string;
  FAge: Integer;
  FLastEdited: TDateTime;
  FEmail: string;
  FMiddleName: Nullable<string>;
public
  property ID: Integer read FId;
  [Column] property Name: string read FName write FName;
  [Column] property Age: Integer read FAge write FAge;
  [Column] property LastEdited: TDateTime read FLastEdited write FLastEdited;
  [Column] property EMail: string read FEmail write FEmail;
  [Column] property MiddleName: Nullable<string> read FMiddleName write FMiddleName;
end;
```

Marshmallow

Start working with "Marshmallow"

Suppose we want to use SQLite3 database engine. At first we must register SQLite3 adapter by using it's unit somewhere in our project, e.g.:

```
uses
  Adapters.SQLite;
```

List of all available adapters:

- Adapters.ADO - use with any ADO supported database
- Adapters.ASA - use with Sybase ASA
- Adapters.DBX - use with any DBX supported database
- Adapters.MSSQL - use with Microsoft SQL Server
- Adapters.SQLite - use with SQLite3 embedded database
- Adapters.UIB - use with Firebird, Interbase
- Adapters.MongoDB - use with MongoDB
- Adapters.Zeos - use with any Zeos supported database
- Adapters.Oracle - use with Oracle
- Adapters.FireDAC - use with any FireDAC supported database

Marshmallow

Session

Then we need to create our *IDBConnection* and *TSession* instances:

```
var
  TestDB: TSQLiteDatabase;
  Connection: IDBConnection;
  Session: TSession;
begin
  TestDB := TSQLiteDatabase.Create(':memory:');
  Connection := TConnectionFactory.GetInstance(dtSQLite, TestDB);
  Session:= TSession.Create(Connection);
  //TSession is our main work class.
  //Do something with the session...
```

Marshmallow

Connection

Or we can simply get new connection instance from the json configured file or string.

```
var
  Connection: IDBConnection;
  Session: TSession;
begin
  Connection := TConnectionFactory.GetInstanceFromFilename(dtUIB, 'conn_Firebird.json');
  Session:= TSession.Create(Connection);
  //Do something with the session...
```

Filename should be valid json file which specifies connection's qualified class name and it's properties. Our *conn_Firebird.json* can look like this:

```
{
    "uib.TUIBDataBase": {
        "UserName": "SYSDBA",
        "PassWord": "masterkey",
        "DatabaseName": "localhost:D:\\DB\\GDB\\TEST.GDB"
    }
}
```

Marshmallow

Repositories

Repositories are a simple way to work with a single type of entities.

```
FCustomerRepository := TMongoDBRepository<TCustomer, Integer>.Create(FSession);
LCustomers := FCustomerRepository.FindAll;
```

Marshmallow can even implement your repository methods automatically! All you need is to extend your interface from IPagedRepository<T, TID> and create it with TProxyRepository, e.g.:

```
type
  ICustomerRepository = interface(IPagedRepository<TCustomer, Integer>)
    ['{955BF130-3E2F-45E2-A9E9-79647CA3F33B}']

    [Query('SELECT * FROM CUSTOMERS WHERE CUSTNAME = :0')] //Query for the SQL database
    [Query('{"CustName": ?0}')] //Query for the MongoDB database
    function FindByName(const AName: string): TCustomer;

    [Query('SELECT * FROM CUSTOMERS WHERE CUSTNAME = :0')]
    function FindByNamePaged(const AName: string; APage: Integer; APageSize: Integer): IDBPage<TCustomer>;//paged methods
  end;
//---
var
  FCustomerRepository: ICustomerRepository;
//----
FCustomerRepository := TProxyRepository<TCustomer, Integer>.Create(FSession, TypeInfo(ICustomerRepository)) as ICustomerRepository;
//---
LCustomer := FCustomerRepository.FindByName('Foo');
CheckEquals('Foo', LCustomer.Name);
```

Note that Marshmallow automatically checks your method's return type and returns one entity, list of entities or IDBPage<T> interface.

Marshmallow

Session Methods:

Get a single record:

```
var
  LCustomer: TCustomer;
begin
  LCustomer := Session.SingleOrDefault<TCustomer>('SELECT * FROM CUSTOMERS WHERE CUSTID=:0', [1]);
```

Or without writing any SQL:

```
var
  LCustomer: TCustomer;
begin
  LCustomer := Session.FindOne<TCustomer>(1);
```

Marshmallow

Get list of customers:

```
var
  LCustomers: IList<TCustomer>;
begin
  LCustomers := Session.GetList<TCustomer>('SELECT * FROM CUSTOMERS;', []);
```

Or without writing any SQL:

```
var
  LCustomers: IList<TCustomer>;
begin
  LCustomers:= Session.FindAll<TCustomer>();
```

Marshmallow

Paged fetches

```
var
  LPage: IDBPage<TCustomer>;
  LFirstCustomer: TCustomer;
begin
  LPage := Session.Page<TCustomer>(1, 10, 'SELECT * FROM CUSTOMERS;', []);
  LFirstCustomer := LPage.Items.First;
```

Or without writing any SQL:

```
var
  LPage: IDBPage<TCustomer>;
  LFirstCustomer: TCustomer;
begin
  LPage := Session.Page<TCustomer>(1,10);
  LFirstCustomer := LPage.Items.First;
```

Marshmallow

Non-query Commands

To execute non-query commands, use the Execute method

```
Session.Execute('INSERT INTO CUSTOMERS SELECT * FROM CUSTOMERS;', []);
```

To get a single value, use ExecuteScalar method:

```
var
  LResult: Integer;
begin
  LResult := Session.ExecuteScalar<Integer>('SELECT COUNT(*) FROM CUSTOMERS', []);
```

Marshmallow

Inserts, Updates and Deletes

```
var
  LCustomer: TCustomer;
begin
  LCustomer := TCustomer.Create;
  try
    LCustomer.Name := 'Insert test';
    //most of the time you should use save, which will automatically insert or update your PODO based on it's state
    Session.Save(LCustomer);

    //explicitly inserts customer into the database
    Session.Insert(LCustomer);

    LCustomer.Name := 'Update customer name';
    //explicitly updates customer's name in the database
    Session.Update(LCustomer);

    //deletes customer from the database
    Session.Delete(LCustomer);
  finally
    LCustomer.Free;
  end;
```

Marshmallow

Transactions

Transactions are very simple to use. Remember that if you didn't commit your started transaction, rollback will be executed when interface goes out of scope and is freed.

```
var
  LTran: IDBTransaction;
begin
  LTran := Session.BeginTransaction;
  //commit
  LTran.Commit;
  //explicit rollback
  LTran.Rollback;
```

Marshmallow

Nullable types

"Marshmallow" supports declaring any type as Nullable<T> type. *Core.Types* unit contains N
has property declared as:

```
[Column]
property MiddleName: Nullable<string> read FMiddleName write FMiddleName;
```

Then we can check if it's value is null:

```
var
  LCustomer: TCustomer;
begin
  LCustomer := Session.FindOne<TCustomer>(1);
  try
    if not LCustomer.MiddleName.IsNull then
    begin
      //do something with the value
      WriteLn(LCustomer.MiddleName);
    end;
  finally
    LCustomer.Free;
  end;
end;
```

Marshmallow

Lazy loading

```
var
  LCustomer: TCustomer;
begin
  //add 2 customers orders to the table
  InsertCustomerOrder(LCustomer.ID, 10, 5, 100.59);
  InsertCustomerOrder(LCustomer.ID, 20, 15, 150.59);
  //get customer
  LCustomer := Session.SingleOrDefault<TCustomer>('SELECT * FROM ' + TBL_PEOPLE, []);
  try
    CheckEquals(2, LCustomer.Orders.Count); //LCustomer.Orders fetches database data only now
    CheckEquals(LCustomer.ID, LCustomer.Orders.First.Customer_ID);
    CheckEquals(10, LCustomer.Orders.First.Customer_Payment_Method_Id);
    CheckEquals(5, LCustomer.Orders.First.Order_Status_Code);
    CheckEquals(LCustomer.ID, Orders.OrdersIntf.Last.Customer_ID);
    CheckEquals(20, LCustomer.Orders.Last.Customer_Payment_Method_Id);
    CheckEquals(15, LCustomer.Orders.Last.Order_Status_Code);
  finally
    LCustomer.Free;
  end;
```

Marshmallow

Criteria API

"Marshmallow" exposes it's own Criteria API which helps to build queries in more
age, you can write your criteria like this:

```
var
  LCustomers: IList<TCustomer>;
begin
  FCriteria := Session.CreateCriteria<TCustomer>;
  LCustomers := FCriteria
    .Add(TRestrictions.Eq('Name', 'Foo'))
    .Add(TRestrictions.Eq('Age', 30))
    .AddOrder(TOrder.Desc('Age')).List;
...
```

Or use *IProperties* interface which makes it even more readable:

```
var
  LCustomers: IList<TCustomer>;
  Age, Name: IProperty;
begin
  Age := TProperty.ForName('Age');
  Name := TProperty.ForName('Name');
  FCriteria := Session.CreateCriteria<TCustomer>;
  LCustomers := FCriteria
    .Add(Name.Eq('Foo'))
    .Add(Age.Eq(30))
    .AddOrder(Age.Desc).List;
```

Marshmallow

More complex example:

```
Age := TProperty.ForName('Age');
  //WHERE ((A.AGE =:AGE1 OR A.AGE = :AGE2) OR A.AGE >=:AGE3)
  //ORDER BY A.AGE DESC
  LCustomers := FCriteria.Add(TRestrictions.Or(TRestrictions.Or(Age.Eq(42), Age.Eq(50)), Age.GEq(10)))
    .AddOrder(Age.Desc)
    .List;
```

Conjunctions and disjunctions are also supported:

```
  Age := TProperty.ForName('Age');
  Name := TProperty.ForName('Name');
  InsertCustomer(42, 'Foo');
  InsertCustomer(50, 'Bar');

  LCustomers := FCriteria.Add(
    TRestrictions
      .Disjunction()
        .Add(Age.Eq(42))
        .Add(Name.Eq('Foo'))
        .Add(Age.Eq(50)))
    .AddOrder(Age.Desc)
    .List;
```

Marshmallow

If your main entity contains other sub-entities and they are connected with ManyToOne relation, then it is even possible to add a criterion from the properties of these subentities. E.g.:

```
procedure TestTCriteria.Add_SubEntity_Criterion;
var
  LOrders: IList<TCustomer_Orders>;
  Age: IProperty;
  LCriteria: ICriteria<TCustomer_Orders>;
begin
  LCriteria := FSession.CreateCriteria<TCustomer_Orders>; //we want to fetch TCustomer_Orders
  //TCustomer_Orders has property Customer: TCustomer
  Age := TProperty<TCustomer>.ForName('Age'); //property from sub-entity

  LOrders := LCriteria.Add( Age.Eq(1) ) //but we want to filter by a Customer's age, because order doesn't have such property
    .List();
end;
```

Marshmallow

Optimistic locking

Optimistic locking can be easily implemented using [Version] attribute.

```
property [Version] Version: Integer
```

If someone tries to update entity which has already changed, *EORMOptimisticLockException* will be raised.

SQL Command Tracking

To log executed statement you'll need to add execution listeners:

```
FConnection.AddExecutionListener(
    procedure(const ACommand: string; const AParams: IList<TDBParam>)
    begin
      //write statement
    end);
```

Marshmallow

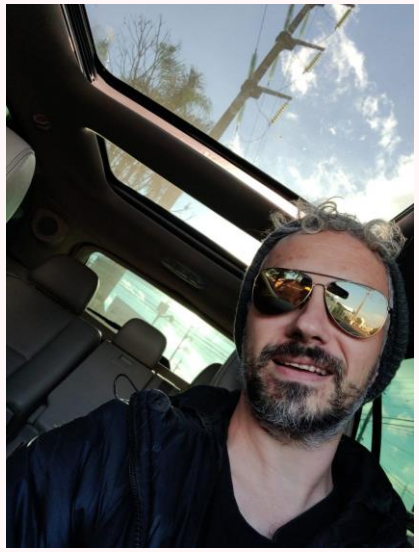
List Sessions

While working with ORM framework You would mostly deal with lists of your entities. "Marshmallow" *TSession* allows You to insert or update your list models based on their state using *SaveList* method. But what about the items which were fetched from the database and later were removed from the list? One option is to attach change listener to your list and delete your entity models from the database when the change occurs. This method is valid but it is not very convenient. To overcome this issue List Sessions were introduced to "Marshmallow". Some basic example how to work with List Sessions:

```
var
  LCustomers: IList<TCustomer>;
  LListSession: IListSession<TCustomer>;
begin
  LCustomers := Session.FindAll<TCustomer>; //fetch customers from the database
  LListSession := Session.BeginListSession<TCustomer>(LCustomers); //start list session for LCustomers
  //do something with LCustomers
  LCustomers.Delete(0); //delete
  LCustomers.Add(TCustomer.Create); //add new customer
  LCustomers[4].Name := 'Edited Name'; //edit customer

  LListSession.CommitListSession; //sends all changes made in LCustomers to the database table
  //List Session will add one new customer, update customers name and delete first customer from the database table
end;
```

Só falar não adianta, bora ver



Juliomar Marchetti

{ Bio Palestrante aqui

juliomarmarchetti@gmail.com

(49) 98426-8589

Embarcadero Conference 2023


OBRIGADO!

O que você achou da palestra?

Acesse o link do QR Code ao lado e responda a pesquisa.