


An aerial photograph of a historic Italian city, likely Florence, showing a dense cluster of buildings with terracotta roofs and a river winding through the center. A large, prominent building with multiple arches is visible in the lower right. The image is overlaid with text and a logo.

# DELPHIDAY

  
italian conference

## TRYSIL

A lightweight, attribute-driven ORM for Delphi



**David Lastrucci**

**OSItalia S.r.l.** – Product Manager

*Autore di Trysil – Delphi ORM*



@ david.lastrucci@gmail.com

github.com/davidlastrucci

in linkedin.com/in/david-lastrucci

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza





OPEN-SOURCE PROJECTS

[github.com/davidlastrucci](https://github.com/davidlastrucci)

## Trysil – Delphi ORM

*A lightweight, attribute-driven ORM for Delphi*

Copyright © David Lastrucci

Licensed under BSD 3-Clause

[github.com/davidlastrucci/Trysil](https://github.com/davidlastrucci/Trysil)

[trysil.lastrucci.net](https://trysil.lastrucci.net)

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza



wintech  
italia



## OS1 Gravity

Il nuovo ERP di OSItalia che utilizza  
Trysil per accedere al database



Open Source Italia S.r.l.

Via del Molinuzzo, 93

59100 - Prato

[www.ositalia.com](http://www.ositalia.com)

[info@ositalia.com](mailto:info@ositalia.com)

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza





**Denny Sbanchi**

**OSItalia S.r.l. – Project Manager**



**@** denny\_sbanchi@live.it

**in** linkedin.com/in/denny-sbanchi

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza





# AGENDA

---

- Cos'è Trysil: vantaggi e svantaggi di un ORM
- Concetti fondamentali
- Feature avanzate
- Demo
- Il plugin di Trysil



# Perché Trysil?

# 1



# Cos'è Trysil?

---

*Trysil è un ORM Delphi moderno che ti permette di scrivere una classe Pascal, decorarla con attributi e dimenticarti dell'SQL*

**ORM:** Un **ORM (Object-Relational Mapper)** è una libreria che fa da ponte automatico tra il mondo a **oggetti** del tuo codice e il mondo a **tabelle** del database relazionale





# Quale problema risolve?

---

Le applicazioni Delphi lavorano con **classi**: *oggetti, ereditarietà, riferimenti tra istanze, collezioni*.

I database relazionali lavorano con **tabelle**: *righe, colonne, foreign key, set di risultati*.

Sono due paradigmi che descrivono gli stessi dati in modi diversi, e il "ponte" tra loro è il problema centrale.



# Vantaggi di adottare un ORM

---

- **Dichiari** la struttura delle entità e l'ORM **genera** l'SQL al posto tuo
- Le operazioni CRUD diventano semplici chiamate ad un metodo (niente righe e righe di SQL per ogni operazione)
- Transazioni, locking, soft delete... feature cablate nel framework, non da reinventare ogni volta
- Il **codice business** ritrova il suo linguaggio naturale, quello degli oggetti
- Cambiare RDBMS diventa una scelta di configurazione, non un progetto di migrazione



# Svantaggi di adottare un ORM

---

- Devi studiare come funziona (menomale che c'è l'AI)
- Meno trasparenza sull'interrogazione al DB
  - Possibile comunque loggare le query costruite dall'ORM prima che vengano eseguite
- Per query analitiche complesse rimane più semplice scrivere direttamente la sintassi SQL



# Chi usa Trysil?

---

- **OS1Gravity:** l'ERP targato OSItalia ha alla base Trysil come ORM per la gestione delle numerose entità che appartengono all'applicativo
- **Wippo:** Software di gestione cantieri sviluppato da Gioele Buonadonna
- **SmartLauncher:** Software gratuito per Windows che avvia un programma e, alla sua chiusura, esegue automaticamente un backup o una sincronizzazione dei dati. Sviluppato da Olray Dragon





# Concetti fondamentali

# 2



# Modello dell'entità

Il mapping tra campi della classe Pascal e colonne della tabella a cui fa riferimento avviene tramite attributi Rtti specifici, così come la validazione dei valori che quel campo può assumere

```
[TTable('Brands')]
[TSequence('BrandsID')]
[TRelation('Products', 'BrandID', False)]
TBrand = class
strict private
  [TColumn('ID')]
  [TPrimaryKey]
  FID: TTPrimaryKey;

  [TColumn('Description')]
  [TRequired]
  [TMaxLength(100)]
  FDescription: String;

  [TColumn('VersionID')]
  [TVersionColumn]
  FVersionID: TTVersion;
public
  property ID: TTPrimaryKey read FID;
  property Description: String read FDescription write FDescription;
  property VersionID: TTVersion read FVersionID;
end;
```



# CRUD type-safe via TTContext

---

TTContext è l'**unico entry point** dell'API. Tutte le operazioni sono metodi generici `<T: class>`

- **Lettura:** `Get<T>`, `Select<T>`, `SelectAll<T>`, `Refresh<T>`
- **Scrittura:** `Insert<T>`, `Update<T>`, `Delete<T>`, `Save<T>`
- **Lifecycle:** `CreateEntity<T>`, `FreeEntity<T>`



# Query builder fluente: TFilterBuilder<T>

Un oggetto che costruisce una query passo dopo passo, concatenando chiamate a metodi che si leggono come una frase.

```
Builder.Where('City').Like('Pra%').AndWhere('Age').GreaterOrEqual(18)
```

- **Sicurezza:** i valori diventano parametri (:p0, :p1...), niente SQL injection
- **Leggibilità:** diventa immediato capire quali sono le condizioni di filtro che si andrà ad applicare
- **Portabilità:** il builder produce un filtro astratto, la traduzione SQL la fa il dialetto del DB
- **Consistenza:** effettua controlli riguardo ai valori per cui effettuo le operazioni di filtro sulla base dell'entità che deve gestire





# RawSelect

---

- Soluzione (parziale) allo svantaggio di non poter scrivere query eccessivamente complesse
- Dati un modello ed una query SQL esegue la query e mappa i risultati in una lista di oggetti della tipologia `<T>` mappata dal modello
  - `procedure RawSelect<T>(const ASQL: String; const AResult: TList<T>);`



# Relazioni e lazy loading

---

- Corrispettivo delle relazioni che intercorrono tra le varie tabelle del DB che si sta mappando implementate tramite due oggetti specifici:
  - `TTLazy<T>`: relazione 1 a 1
  - `TTLazyList<T>`: relazione 1 a N
- Ottimizzazione delle letture tramite **Lazy Loading**
- Gestione delle relazioni in cancellazione di un'entità tramite **[TRelationAttribute]**



# Supporto multi-database

- **Supporto multi-database** trasparente
  - Trysil supporta **4 RDBMS** con lo stesso codice (Firebird, PostgreSQL, SQLite, SQL Server)
- La selezione del DB è una **scelta di configurazione**, non di codice

Operazione	Firebird	PostgreSQL	SQLite	SQL Server
Versione DB	<code>rdb\$get_context(...)</code>	<code>VERSION()</code>	<code>sqlite_version()</code>	<code>@@VERSION</code>
Paginazione	<code>FIRST n SKIP m</code>	<code>LIMIT n OFFSET m</code>	<code>LIMIT n OFFSET m</code>	<code>OFFSET m ROWS FETCH NEXT n ROWS ONLY</code>



# Feature avanzate

# 3





# Audit automatico e soft delete

---

*Chi ha modificato quel record e quando?*

- **[TCreatedAt] / [TCreatedBy] → data e utente di creazione**
- **[TUpdatedAt] / [TUpdatedBy] → data e utente dell'ultima modifica**

*Siamo sicuri che non mi serva in futuro?*

- **[TDeletedAt] / [TDeletedBy] → data e utente della cancellazione**
  - **Attivazione automatica del meccanismo di soft delete se presenti su una classe**
- **Utile per poter revertare una cancellazione o per**



# Optimistic Locking

---

- Nasce dalla scommessa «ottimistica» che il conflitto in modifica di un record sia un evento piuttosto raro
- Gestione del conflitto invece che prevenzione
  - Tramite una colonna di versionamento [TVersionColumn] vado a specificare su quale campo del modello applicare il meccanismo di lock



# Sessioni e Transazioni

---

- Ogni sessione è un implementazione del pattern «**Unit of Work**». Accumulo modifiche su una lista di dettagli (le righe di un ordine) e ne gestisco il salvataggio su db al salvataggio del documento
- Le transazioni sono invece utilizzate per garantire **l'atomicità delle operazioni** (o tutto o niente)
  - In caso di eccezione durante il salvataggio di un master-detail viene revertato tutto ciò che è stato scritto in quella transazione evitando scenari inconsistenti sul DB



# Identity Map

---

- Meccanismo per cui **ogni record sul db corrisponde ad un unico oggetto** nell'applicazione
- **Attiva di default** in Trysil, ma può essere disabilitata dal Context (flag **UseIdentityMap**)
- Se attiva si occupa lei della liberazione delle entità create dal Context
- Dà il meglio su sessioni longeve e stateful come un'app desktop, mentre in REST stateless, il suo valore si riduce alla singola richiesta
  - E' legata alla Connection, nelle API REST creo Connection e Context ad ogni richiesta



# Demo

# 4



# DEMO — Setup dell'applicazione

---

- Pool FireDAC off; TConfig legge Orders.Config.json (profilo 'active')
- **Connection:** TTFireDACConnectionFactory.RegisterConnection + CreateConnection
- **Context:** TTContext.Create(FConnection) — unico entry point per tutta l'app
- **Audit:** OnGetCurrentUser popola i campi CreatedBy / UpdatedBy / DeletedBy
- Status bar: driver e versione del DB connesso



# DEMO — Brands: modello e CRUD

---

- **Modello:** [TTable] + [TSequence] (generator della PK) + [TRelation] su Products
- **[TRelation]:** False = no cascade → alla delete scatta il check "relazione in uso"
- **CreateEntity<TBrand>:** l'entità nasce sempre dal Context, mai Create manuale
- **Save<TBrand>:** upsert — Insert se nuova, Update se già tracciata dal Context
- **FreeEntity<TBrand>:** su annulla/errore rilascia l'istanza all'Identity Map



# DEMO — Products: lazy loading

---

- **Modello:** la FK verso Brand è un TTLazy<TBrand>
- **[TColumn('BrandID')]:** la colonna di Products si collega da sola alla PK di Brands
- **Lazy:** TTLazy e TTLazyList nascono col modello ma NON sono popolati
- **Fetch:** scatta solo al primo accesso a .Entity / .List





# DEMO — Customers: soft delete

---

- **Soft delete:** [TDeletedAt] / [TDeletedBy] → la Delete marca, non cancella
- **Query builder:** CreateFilterBuilder<TCustomer> — query costruita a passi



# DEMO — Orders: master/detail

---

- **Cascade:** [TRelation('Orders','OrderID', True)] su OrderDetail → cascade sulle righe
- **Session:** TTSession<TOrderDetail> sul dettaglio: modifiche solo in memoria
- **OrderDetailDialog:** ApplyEntity → FSession.Save (staging, nessun accesso al DB)
- **OrderDialog:** ApplyEntity → Save<TOrder> + FSession.ApplyChanges in TTTransaction



# DEMO — Cambio DB da config

---

- **Config:** Orders.Config.json — campo 'active' da 'SQLite' a 'MSSQL'
- **Restart:** la status bar mostra il nuovo driver e versione
- Stesso EXE, stesso Model/View/Dialog: zero ricompilazione
- **Dietro le quinte:** TTSyntaxClasses inietta il dialetto SQL corretto



# Il plugin di Trysil

# 5

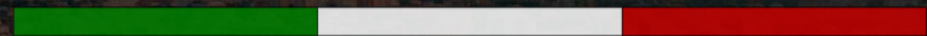


# Plugin: dal designer al progetto

---

- Plugin Delphi con designer visuale di entità
- Disegna entità, colonne, FK, master/detail
- Genera: codice del modello Trysil (le stesse classi del demo)
- Genera: DDL SQL dialect-specific (Firebird/PG/SQLite/MSSQL)

# DELPHIDAY



italian conference

## THANK YOU

[github.com/davidlastrucci/Trysil](https://github.com/davidlastrucci/Trysil) · [trysil.lastrucci.net](https://trysil.lastrucci.net)

BSD 3-Clause · [david.lastrucci@gmail.com](mailto:david.lastrucci@gmail.com)

*Domande?*