sequality SOFTWARE ENGINEERING

Enterprise Meets Embedded

Object-Relational Mapping with Qt

Dmitriy Purgin, BSc

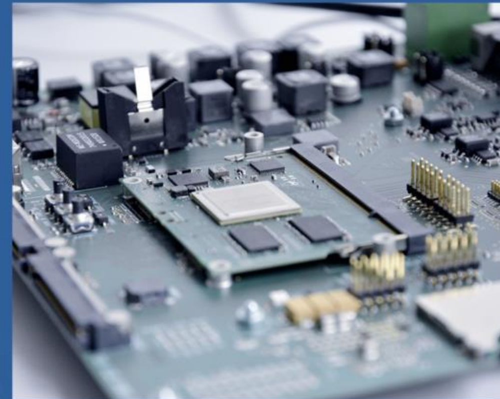
sequality software engineering

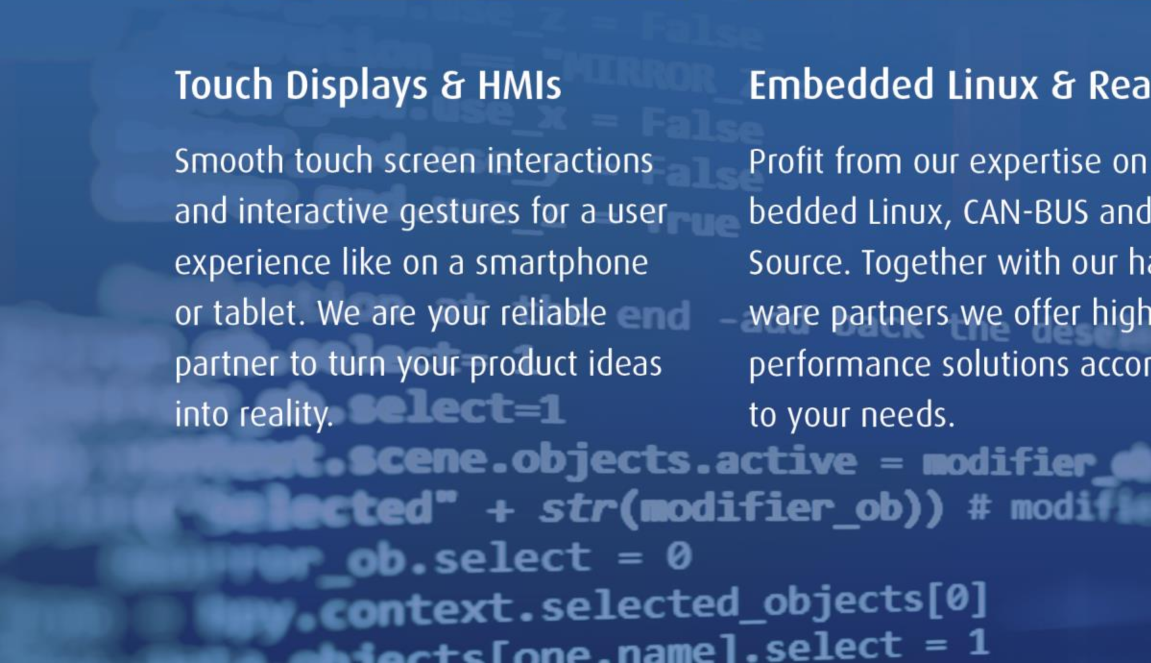
Softwarepark 26, A-4232 Hagenberg, Austria

dmitriy.purgin@sequality.at

sequality SOFTWARE ENGINEERING

Your Partner for Technical Software





Touch Displays & HMIs

Smooth touch screen interactions and interactive gestures for a user experience like on a smartphone or tablet. We are your reliable partner to turn your product ideas into reality.

Embedded Linux & Realtime

Profit from our expertise on embedded Linux, CAN-BUS and Open Source. Together with our hardware partners we offer high-performance solutions according to your needs.

Automotive & Off-Highway

Digital instrument clusters, integrated solutions for navigation, as well as captivating graphics in 2D and 3D: In this field, we focus on open and platform-independent technologies based on embedded Linux, C++ and Qt.

Industrial Applications

High-performance HMI solutions for machine operations with an intuitive usability: We combine efficient technology and innovative design, when developing your industrial products.

Plan for Today

Data processing in embedded

What is object-relational mapping and why do we need it?

OR mapping using Qt and live demonstration

QtOrm: a new ORM library prototype

Dmitriy.Purgin@sequality.at
© sequality software engineering

Data Processing in Embedded

Dmitriy.Purgin@sequality.at
© sequality software engineering

State-of-the-Art in Embedded

Industry 4.0 is the heart of many enteprises nowadays [1]

- Large amounts of generated data can be used to improve productivity

Constantly increasing device performance and the complexity of business logic both enable edge or fog computing

- Data storage and processing moved in close proximity to where it is needed, as opposed to cloud computing [2, 7]
- Preventive maintenance as in [3]

Dmitriy.Purgin@sequality.at
© sequality software engineering

Data Representation

sequality SOFTWARE ENGINEERING

Data is structured

We cannot do anything with unstructured data

Data is relational

A car has an engine, the engine produces telemetry

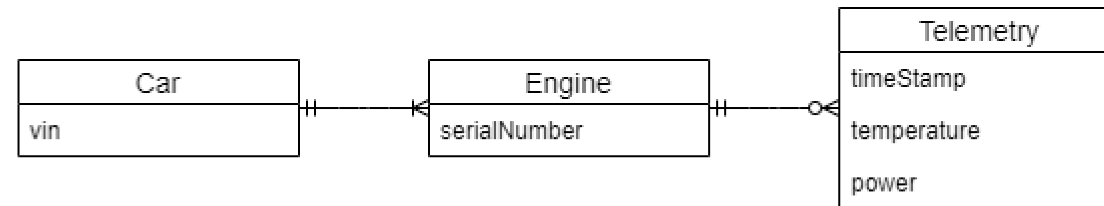
Data represents real-world entities

A car has an engine, the engine produces telemetry, telemetry messages represent the internal state of the engine

Relational databases were proposed in 1970 by Edgar Codd [4]

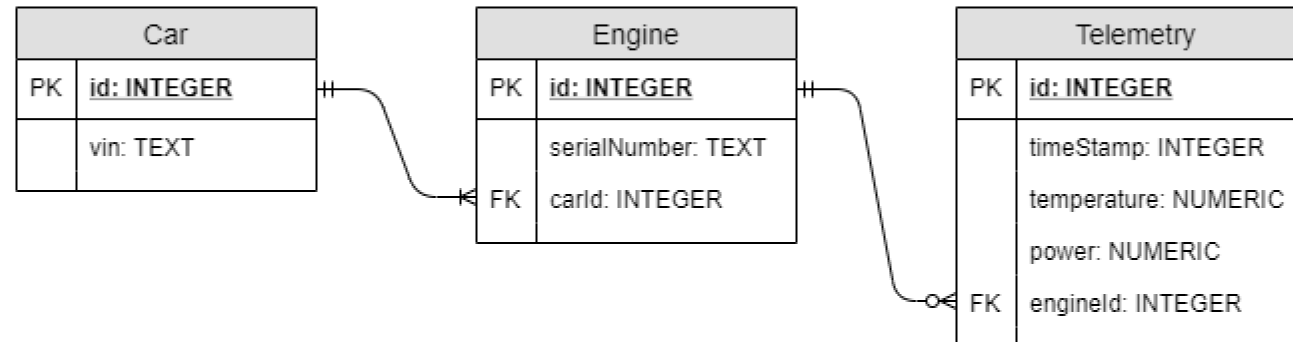
Dmitriy.Purgin@sequality.at
© sequality software engineering

Hybrid Car Telemetry Data Model



Dmitriy.Purgin@sequality.at
© sequality software engineering

…an SQLite Instance



Dmitriy.Purgin@sequality.at
© sequality software engineering

Save Telemetry Using QtSql

```
void saveTelemetry(int engineId, qreal temperature, int power)
{
    QSqlQuery query;
    query.prepare("INSERT INTO Telemetry(timeStamp, temperature, power, engineId) "
                  "VALUES(:timeStamp, :temperature, :power, :engineId)");

    query.bindValue(":timeStamp", QDateTime::currentDateTime());
    query.bindValue(":temperature", temperature);
    query.bindValue(":power", power);
    query.bindValue(":engineId", engineId);

    query.exec();
}
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Problems?

Need to learn SQL and database engine specifics

- Easy for simple statements but requires more knowledge as the project grows

SQL statements are in text

- Easy to break everything by renaming a column or a table
- Any typos are detected only in runtime

Strong coupling to QtSql

- Additional dependency is a problem for unit testing

Dmitriy.Purgin@sequality.at

© sequality software engineering

Read Telemetry Using QtSql

```
void readTelemetry(int engineId)
{
    QSqlQuery query;
    query.prepare("SELECT * FROM Telemetry WHERE engineId = :engineId ORDER BY timeStamp");

    query.bindValue(":engineId", engineId);

    query.exec();
    
    while (query.next())
    {
        // what next?
    }
}
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Even More Problems?

sequality SOFTWARE ENGINEERING

Query results are returned as a container of QVariant

- Still coupled to QtSql

Do we need a QAbstractListModel?

- Need to extract column names, generate roles, copy the data

Do we use the data directly as a container of QSqlRecord?

- Encapsulation violation and even stronger coupling to QtSql

What do we do with related data?

- Engine references are integers at this point

Dmitriy.Purgin@sequality.at
© sequality software engineering

Possible Solution

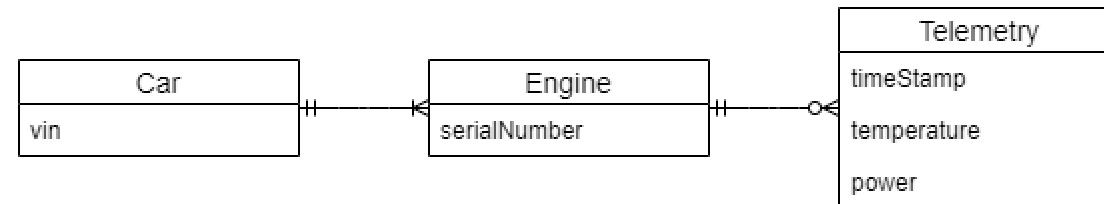
Abstract database records by domain-specific C++ objects with the same structure

Use the domain classes in the business logic

Enclose the persistence logic into a separate subsystem in the application

Dmitriy.Purgin@sequality.at
© sequality software engineering

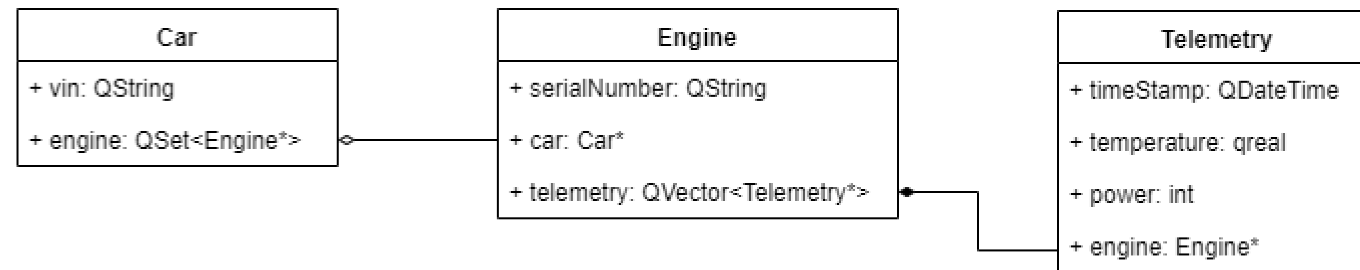
Hybrid Car Telemetry Data Model



Dmitriy.Purgin@sequality.at

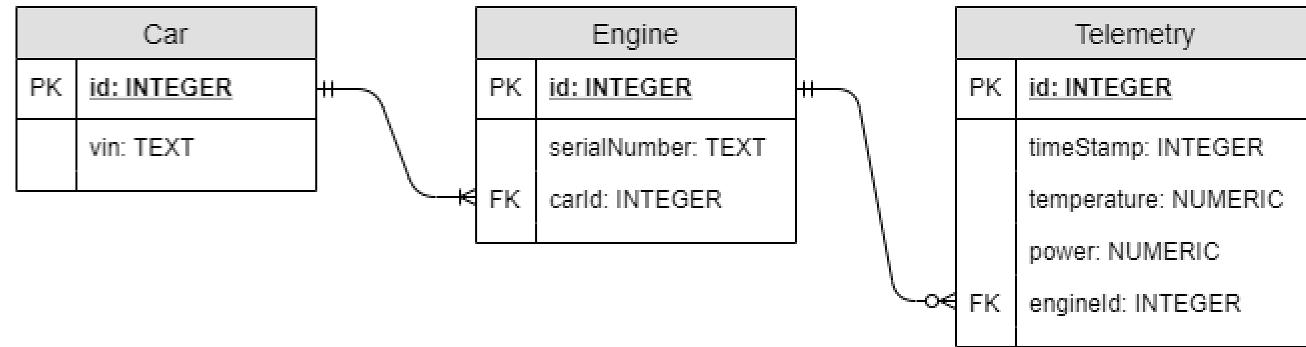
© sequality software engineering

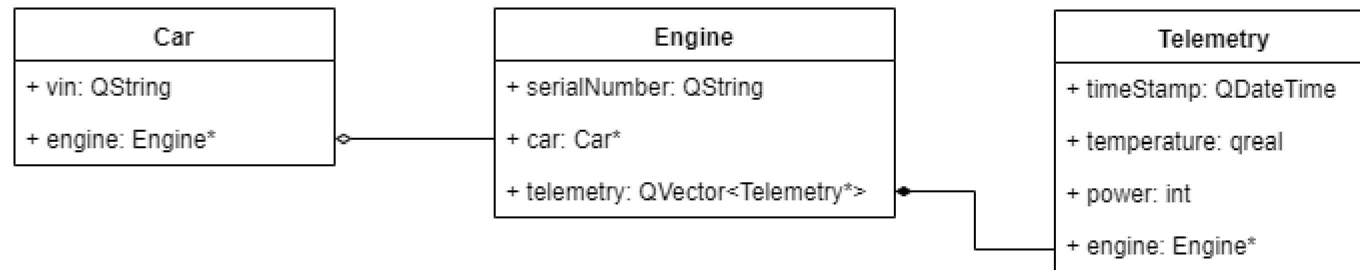
…and a C++ Instance



Dmitriy.Purgin@sequality.at
© sequality software engineering

Difference?





Dmitriy.Purgin@sequality.at
© sequality software engineering

Problems Again?

Both C++ and SQL data models must be in sync

Identity vs. Equality

- C++ objects are identified by their addresses and not by their data
- Tables usually have a designated primary key field, unique value for each row

SQL data model joins multiple tables, C++ data model navigates a graph

Dmitriy.Purgin@sequality.at
© sequality software engineering

Solution?

Object-relational mapping libraries

- Put simply: A technique to map database tables to object-oriented classes.

Most of the problems are solved by automation or by convention

- Automated schema synchronization
- Predefined fetching strategies for related entities
- Automatic SQL statement generation
- But: object-relational impedance mismatch [6]

Dmitriy.Purgin@sequality.at
© sequality software engineering

Advantages of ORM

Data model is described in one place

Schema is created from C++ classes

Quick introduction of persistence

No need to write database connectors, SQL queries, convert references etc.

Better abstractions lead to better separation of concerns

Less dependencies, less complexity in single units

Established approach in enterprise

67.5% of Java developers used an ORM framework in 2014 [5]

Dmitriy.Purgin@sequality.at
© sequality software engineering

Disadvantages of ORM

Hand-tailored SQL is usually better in terms performance and memory

Hard to impossible to use the underlying database features

- Stored procedures, functions, views, aggregates, indices

Complexity is shifted to the ORM code from the SQL code

Requires type introspection to implement the mapping

- Still impossible for “vanilla” C++

Dmitriy.Purgin@sequality.at
© sequality software engineering

Why ORM in Qt?

sequality SOFTWARE ENGINEERING

C++/Qt professionals are generally far from databases

Other major technologies have established ORM approaches

- Java: Hibernate, Spring Data
- C#: NHibernate, LINQ-to-SQL

Qt provides type introspection (meta-object system)

Java and C# developers switching to C++ are looking for OR mappers

- Because that is how they usually do it in Java and C#

Dmitriy.Purgin@sequality.at
© sequality software engineering

Notable Existing C++ OR Mappers

sequality SOFTWARE ENGINEERING

QxORM [8]

- GPL or Commercial
- A comprehensive library, feature-rich even outside the scope of OR mapping

ODB [9]

- GPL, Commercial, or Free Proprietary depending on database backend
- Not Qt-specific
- Requires an additional code generator in the toolchain

Dmitriy.Purgin@sequality.at
© sequality software engineering

Why a New One?

sequality SOFTWARE ENGINEERING

License considerations

GPL or Commercial is often a no-go for smaller companies

Base on modern C++ and Qt

Existing implementations are burdened by legacy code

Integrate into Qt as seamlessly as possible

There should be no additional code generators, dependencies or duplication of the existing Qt funtionality

Dmitriy.Purgin@sequality.at
© sequality software engineering

Object-Relational Mapping Using Qt

QtOrm: a new ORM library prototype for Qt

Dmitriy.Purgin@sequality.at
© sequality software engineering

QtOrm

Open source under LGPLv3

- Can be used in closed-source projects under specific conditions (see the license)

Implemented as a Qt module

- Just add QT += orm and CONFIG += c++17

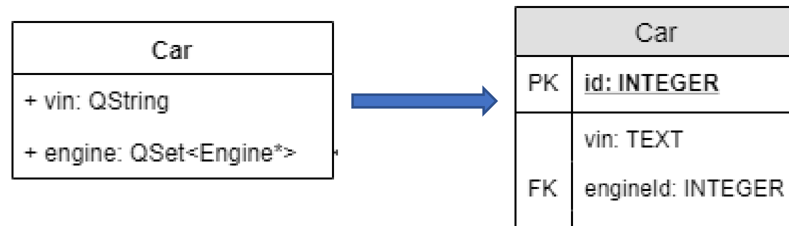
Still in prototype phase, not an official Qt module

Dmitriy.Purgin@sequality.at
© sequality software engineering

sequality SOFTWARE ENGINEERING

Domain Classes to Tables

What we want



What we can

```
class Car : public QObject
{
    Q_OBJECT

    Q_PROPERTY(int id READ id WRITE setId NOTIFY idChanged)
    Q_PROPERTY(QString vin READ vin WRITE setVin NOTIFY vinChanged)
    Q_PROPERTY(QVector<Engine*> engines READ engines WRITE setEngines NOTIFY enginesChanged)

public:
    Q_INVOKABLE explicit Car(QObject* parent = nullptr);
```

Domain Classes to Tables

Q_PROPERTY defines a mappable class property

With Q_INVOKABLE constructor the class can be instantiated via QMetaObject::newInstance()

Getters and setters can be generated using Qt Creator’s code refactoring

qRegisterOrmEntity<>() is required for all domain classes

Dmitriy.Purgin@sequality.at
© sequality software engineering

QOrmSession

Entry point to the OR mapper

Can be configured manually with QOrmSessionConfiguration or with a JSON file

Takes the ownership of all entity instances

Dmitriy.Purgin@sequality.at
© sequality software engineering

QOrmSession File Configuration

sequality SOFTWARE ENGINEERING

Embed into the resource or put near the application executable

qtorm.json

```json
{
    "provider": "sqlite",
    "sqlite": {
        "verbose": true,
        "schemaMode": "recreate",
        "databaseName": "qws.db"
    }
}
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

sequality SOFTWARE ENGINEERING

Writing Data

```cpp
#include <QCoreApplication>

#include "domain/car.h"
#include "domain/engine.h"
#include "domain/telemetry.h"

#include <QOrmSession>

int main(int argc, char *argv[])
{
    QCoreApplication app{argc, argv};

    qRegisterOrmEntity<Car, Engine, Telemetry>();

    Car* car1 = new Car();
    car1->setVin("ABC001");

    Car* car2 = new Car();
    car2->setVin("ABC002");

    QOrmSession session;
    session.merge(car1, car2);

    return app.exec();
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
Executing: "DROP TABLE Car"
Executing: "CREATE TABLE Car(id INTEGER PRIMARY KEY AUTOINCREMENT,vin TEXT)"
Executing: "INSERT INTO Car(vin) VALUES(:vin)"
Bound parameters: QMap((":vin", QVariant(QString, "ABC001")))
Executing: "INSERT INTO Car(vin) VALUES(:vin)"
Bound parameters: QMap((":vin", QVariant(QString, "ABC002")))
```


Dmitriy.Purgin@sequality.at

© sequality software engineering

Schema Synchronization

sequality SOFTWARE ENGINEERING

Synchronization modes: Recreate, Update, Validate, Bypass

- Currently only Recreate and Bypass are implemented

Table schema is synchronized once the entity is used in an ORM session

Dmitriy.Purgin@sequality.at
© sequality software engineering

Writing Referenced Data

```
#include <QCoreApplication>

#include "domain/car.h"
#include "domain/engine.h"
#include "domain/telemetry.h"

#include <QOrmSession>

int main(int argc, char *argv[])
{
    QCoreApplication app{argc, argv};

    qRegisterOrmEntity<Car, Engine, Telemetry>();

    Car* car1 = new Car();
    car1->setVin("ABC001");

    Engine* car1engine1 = new Engine();
    car1engine1->setSerialNummber("0001");
    car1engine1->setCar(car1);

    Engine* car1engine2 = new Engine();
    car1engine2->setSerialNummber("0002");
    car1engine2->setCar(car1);

    car1->setEngines({car1engine1, car1engine2});

    Car* car2 = new Car();
    car2->setVin("ABC002");

    QOrmSession session;
    session.merge(car1engine1, car1engine2, car1, car2);

    return app.exec();
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
Executing:  "DROP TABLE Car"
Executing:  "CREATE TABLE Car(id INTEGER PRIMARY KEY AUTOINCREMENT,vin TEXT)"
Executing:  "INSERT INTO Car(vin) VALUES(:vin)"
Bound parameters: QMap((":vin", QVariant(QString, "ABC001")))
Executing:  "DROP TABLE Engine"
Executing:  "CREATE TABLE Engine(id INTEGER PRIMARY KEY AUTOINCREMENT,serialnummber TEXT,car_id INTEGER)"
Executing:  "INSERT INTO Engine(serialnummber,car_id) VALUES(:serialnummber,:car_id)"
Bound parameters: QMap((":car_id", QVariant(int, 1))(":serialnummber", QVariant(QString, "0001")))
Executing:  "INSERT INTO Engine(serialnummber,car_id) VALUES(:serialnummber,:car_id)"
Bound parameters: QMap((":car_id", QVariant(int, 1))(":serialnummber", QVariant(QString, "0002")))
Executing:  "INSERT INTO Car(vin) VALUES(:vin)"
Bound parameters: QMap((":vin", QVariant(QString, "ABC002")))
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Transactions

Obtain a transaction token from the QOrmSession

Transaction commits or rolls back automatically at the end of the block

```
auto token = session.declareTransaction(QOrm::TransactionPropagation::Require,
                                        QOrm::TransactionAction::Commit);
if (!session.merge(addTelemetry(car1engine1, 300, 1200),
                   addTelemetry(car1engine1, 320, 1300),
                   addTelemetry(car1engine1, 200, 500)))
{
    token.rollback();
}
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Fetching Data

```
auto cars = session.from<Car>().select().toVector();

for (const Car* car : cars)
{
    qDebug() << "Car: ID" << car->id() << ", VIN:" << car->vin() << ", engines:";

    for (const Engine* engine : car->engines())
    {
        qDebug() << "\tSerial number:" << engine->serialNumber()
                 << ", messages:" << engine->messages().size();
    }
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process...

```
Executing: "SELECT * FROM Car ORDER BY vin DESC"
Car: ID 2 , VIN: "ABC002" , engines:
Car: ID 1 , VIN: "ABC001" , engines:
        Serial number: "0001" , messages: 0
        Serial number: "0002" , messages: 0
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Filtering and Ordering Data

```
auto result = session.from<Telemetry>()
                  .filter(Q_ORM_CLASS_PROPERTY(temperature) >= 300 &&
                          Q_ORM_CLASS_PROPERTY(engine) == car1engine1)
                  .order(Q_ORM_CLASS_PROPERTY(power), Qt::DescendingOrder)
                  .select()
                  .toVector();

qDebug() << "Fetched: " << result.size();
for (const Telemetry* msg : result)
{
    qDebug() << "\t" << msg->timeStamp() << ", temperature:" << msg->temperature() << ","
             << "power:" << msg->power() << ", VIN:" << msg->engine()->car()->vin();
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
Executing: "SELECT * FROM Telemetry WHERE (temperature >= :temperature) AND (engine_id = :engine_id) ORDER BY power DESC"
Bound parameters: QMap((":engine_id", QVariant(int, 1))(":temperature", QVariant(int, 300)))
Fetched:  2
         QDateTime(2019-11-05 06:50:55.804 W. Europe Standard Time Qt::LocalTime) , temperature: 320 , power: 1300 , VIN: "ABC001"
         QDateTime(2019-11-05 06:50:55.804 W. Europe Standard Time Qt::LocalTime) , temperature: 300 , power: 1200 , VIN: "ABC001"
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Data Removal

```
session.remove(car2);

for (auto car : session.from<Car>().select().toVector())
{
    qDebug() << "Car:" << car->vin();
}
```

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Executing: "DELETE FROM Car WHERE id = :id"
Bound parameters: QMap((":id", QVariant(int, 2)))
Executing: "SELECT * FROM Car "
Car: "ABC001"
```


Dmitriy.Purgin@sequality.at
© sequality software engineering

Library Design Principles

sequality SOFTWARE ENGINEERING

As little boilerplate as possible

Try to use the Qt metasystem as much as possible

Reasonable ORM defaults for smaller databases

Less flexibility but easier to use

Fail fast

Crash on any mapper inconsistency with reasonable error messages

Based on Qt 5.12 LTS with Qt 6 kept in mind

C++ standard library and C++17 required

Dmitriy.Purgin@sequality.at
© sequality software engineering

Limitations

Early prototype phase

SQLite backend is being developed

- Maybe: CSV, XML

Currently single-threaded only

Source code must be UTF-8 encoded

Dmitriy.Purgin@sequality.at
© sequality software engineering

Further Plans until June 2020

Integration with QML

Extending ORM capabilities

- Custom table and column names, nullability attributes, index hints, relaxed ID property, cascade deletes

(Academic) research on tools extension

- Qt Creator: code completion, refactoring
- moc: extend Q_PROPERTY, add ORM attributes

Dmitriy.Purgin@sequality.at
© sequality software engineering

sequality SOFTWARE ENGINEERING

References

1. Li Da Xu, Eric L. Xu & Ling Li (2018) Industry 4.0: state of the art and future trends, International Journal of Production Research, 56:8, 2941-2962, DOI: 10.1080/00207543.2018.1444806
2. M. Satyanarayanan, "The Emergence of Edge Computing," in *Computer*, vol. 50, no. 1, pp. 30-39, Jan. 2017. doi: 10.1109/MC.2017.9
3. Mishra, Krishna Mohan, and Kalevi Huhtala. "Elevator fault detection using profile extraction and deep autoencoder feature extraction for acceleration and magnetic signals." *Applied Sciences* 9.15 (2019): 2990.
4. Codd, Edgar F. "A relational model of data for large shared data banks." *Communications of the ACM* 13.6 (1970): 377-387.
5. Chen, Tse-Hsun, et al. "An empirical study on the practice of maintaining object-relational mapping code in Java systems." *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016.
6. Bauer, Christian, and Gavin King. *Java Persistance with Hibernate*. Dreamtech Press, 2006.
7. Hamilton, Eric. *What is Edge Computing: The Network Edge Explained*. https://www.cloudwards.net/what-is-edge-computing/, 2018
8. QxORM Library https://www.qxorm.com/
9. C++ ODB (https://www.codesynthesis.com/products/odb/)

Dmitriy.Purgin@sequality.at

© sequality software engineering

sequality
SOFTWARE ENGINEERING

Thank you for your attention

BSc

Dmitriy Purgin

dmitriy.purgin@sequality.at

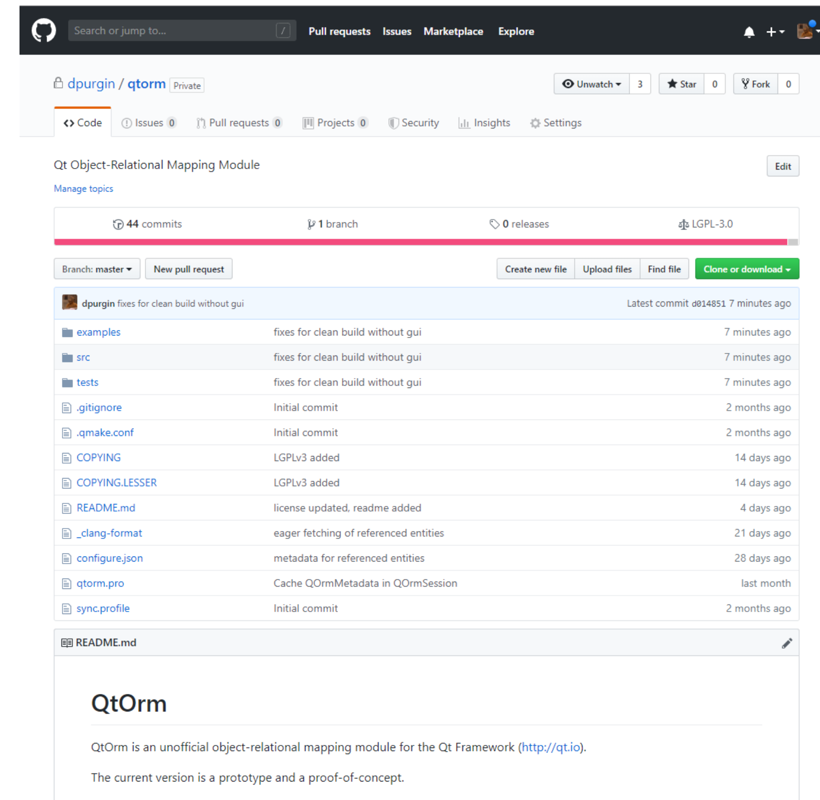
sequality software engineering

Softwarepark 26

A-4232 Hagenberg, Austria

T: +43 7236 / 26 101

M: +43 676 / 97 72 681



https://sequality.at

https://github.com/dpurgin/qtorm

© sequality software engineering