Build a database-independent cross-platform persistence layer with Qt

Johan Decorte – www.gorat.be/training

Johan.decorte@telenet.be

Who am I ?

ir. Johan Decorte (www.gorat.be)

burgerlijk ir. computerwetenschappen (M. Sc. in Eng., KUL, '86)

- Software Developer
- Technical architect
- Project Manager-Analyst
- IT (development) manager

Siemens
Barco
Attentia Sociaal Secretariaat
Agfa Healthcare

Since 2011:

- lecturer IT at University College Ghent
- freelance trainer C++, databases, business analysis, IT management
- juridical dispute resolution (*gerechtsdeskundige)* in ICT cases

Contact: johan.decorte@telenet.be

Contents of C++ training

- Basic C++ syntax
- Pointers & references
- dynamic memory allocation: new and delete
- Introduction tot **object oriented programming** and UML
- classes
- single and multiple inheritance
- **polymorphism**
- operator overloading
- templates
- **exceptions**
- streams
- file processing
- Standard Template Library (**STL**)
- Advanced sample code with **Design Patters**
- **C++11** and C++14: **smart pointers**, multi-threading, **lambda**-expressions and move semantics
- Best practices in C++ according to Deitel&Deitel and **Scott Meyers**
- Access to **SQL Server**, **mysql** and **Oracle**

> C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.
>
> -- *Bjarne Stroustrup (2007)–*

Contents

- What is Qt?
- The importance of database independence
- The Qt API for database access
  - Plain SQL
  - Generate SQL
  - Link table content to GUI objects
  - Stored Procedures
- Demo
- Drawbacks and remarks
- Alternatives
- Conclusions
- Q&A

Contents

- **What is Qt?**
- The importance of database independence
- The Qt API for database access
  - Plain SQL
  - Generate SQL
  - Link table content to GUI objects
  - Stored Procedures
- Demo
- Drawbacks and remarks
- Alternatives
- Conclusions
- Q&A

What is Qt?

- Multiplatform framework: networking, GUI, multimedia, data (SQL, Json, XML), …
- IDE: QtCreator
- Tools & Toolchains
- Licensing
  - Qt licensed under commercial licenses are appropriate for development of proprietary/commercial software
  - Qt licensed under the GNU Lesser General Public License (LGPL) version 3 is appropriate for the development of Qt applications provided you can comply with the terms and conditions of the GNU LGPL version 3 (or GNU GPL version 3)

Contents

The importance of database independence

- Customers might require their favorite database
- "Encapsulate what varies": put database specific code in separate classes or outside your C++ program
- Scalability: e.g. migration from MySQL to Oracle without coding
- Qt offers an OO wrapper around native C-APIs of database vendors

Relational databases supported by QtSQL

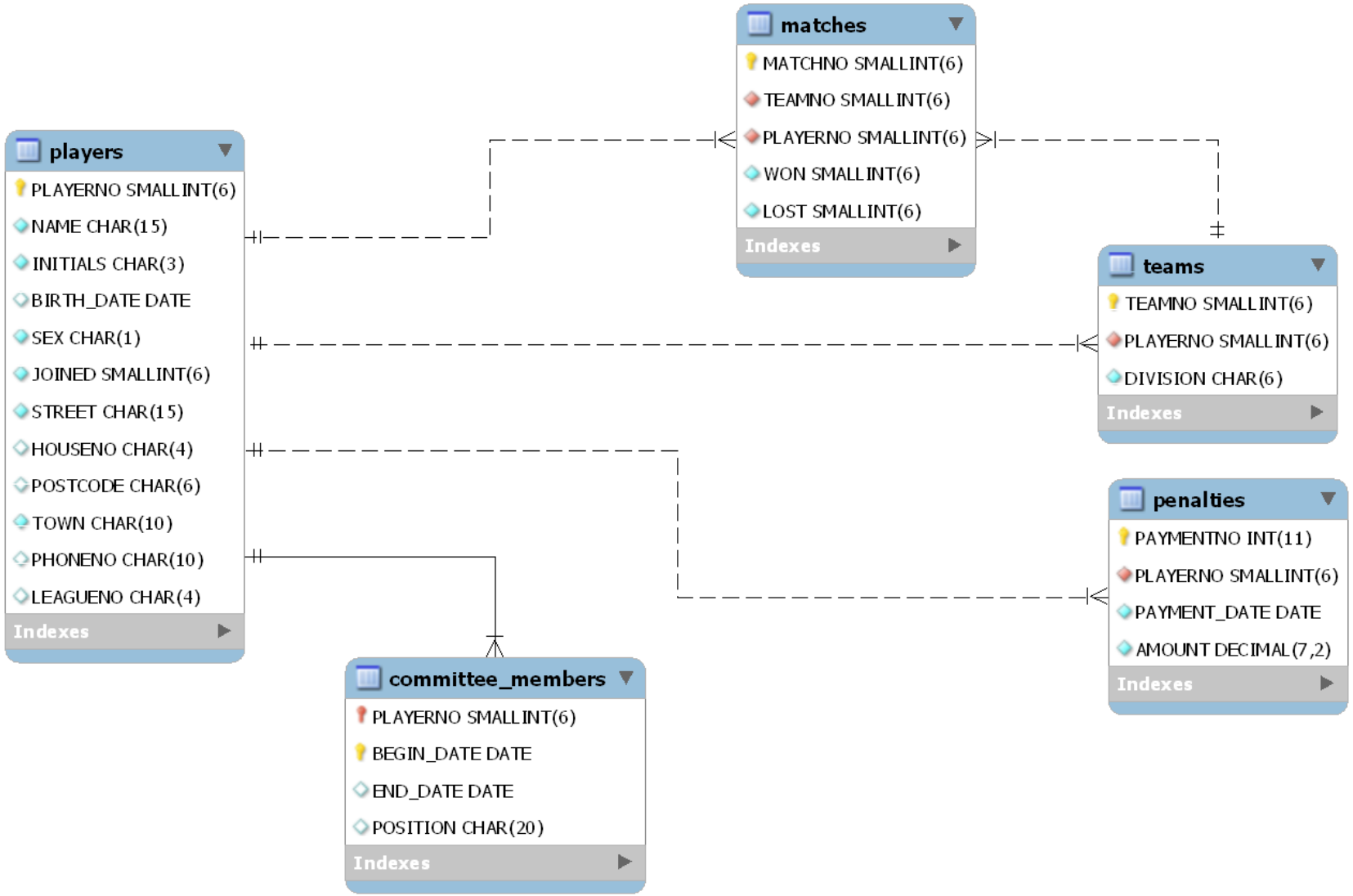
- IBM DB2
- MySQL
- Oracle
- ODBC (generic)
- Microsoft SQL Server: through ODBC
- PostgreSQL 7.3 and later
- SQLite
- Sybase

Link native library of database vendors you support with your Qt application

Contents

- What is Qt?
- The importance of database independence
- **The Qt API for database access**
  - Plain SQL
  - Generate SQL
  - Link table content to GUI objects
  - Stored Procedures
- Demo
- Drawbacks and remarks
- Alternatives
- Conclusions
- Q&A

Sample database: admin of tennis club



Plain SQL

Problem: list all players that live in either Stratford or Inglewood
and got a total penalty amount of at least £100. List number, name, birth date, sex and total amount. Order by name.

SQL statement:

```
select pl.playerno, name, birth_date, sex,sum(amount)
from players pl join penalties pe on pl.playerno=pe.playerno
where town in ('Stratford','Inglewood')
group by pl.playerno,name,birth_date,sex
having sum(amount) >= 100
order by name;
```

Connect to your database

```
QSqlDatabase db = QSqlDatabase::addDatabase("QODBC");  // specify database driver
if (db.lastError().isValid())
{
   throw(DBException(db.lastError().text().toStdString()));
}
db.setDatabaseName(tennis);
If (!db.open())
{
   throw(DBException(db.lastError().text().toStdString()));
}
```

Pass plain SQL to database: the primitive way

```
string qrystring = "select pl.playerno, name, birth_date, sex, sum(amount)from players pl ";
qrystring        += "join penalties pe on pl.playerno=pe.playerno ";
qrystring        += "where town in ('Stratford','Inglewood') ";
qrystring        += "group by pl.playerno,name,birth_date,sex ";
qrystring        += "having sum(amount) >= 100 ";
qrystring        += "order by name";

QSqlQuery query(qrystring.c_str());
if (query.lastError().isValid())
{
   ostringstream msg;
   msg << __FILE__ << ": " << __LINE__ << query.lastError().text().toStdString();
   throw(DBConnector::DBException(msg.str()));
}
```

Collect results in local variables

```
while (query.next())
{
  int    playerno   = query.value(0).toInt();
  string name       = query.value(1).toString().toStdString();
  QDate  birth_date = query.value(2).toDate();
  string sex        = query.value(3).toString().toStdString();
  float  penalties  = query.value(4).toFloat();
  // process variables
}
```

Transaction support

- Transaction = a **logical** unit of work, a sequence of statements that should be carried out together or not
- The RDBMS ensures transactions run **logically** isolated
- Example:
  - when inserting a new penalty record the paymentno to insert should be the highest + 1
  - two separate processes can simultaneously ask for the highest number and try to insert the same primary key

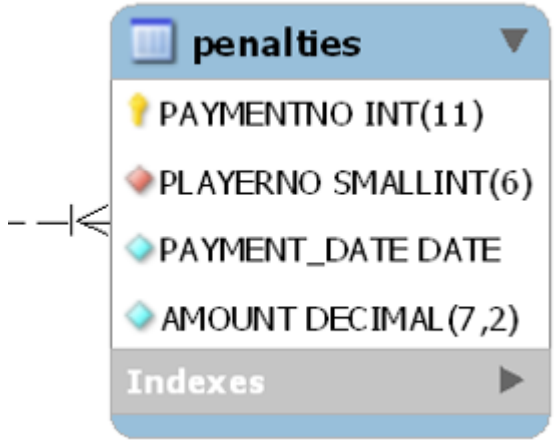
Transaction support

› QSqlQuery can also be used for database-specific commands:

```
QSqlQuery query;
query.exec("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE");
if (query.lastError().isValid())
{
    ostringstream msg;
    msg << __FILE__ << ": " << __LINE__ << query.lastError().text().toStdString();
    throw(DBException(msg.str()));
}
```

Insert a new record: the primitive way

```
// start transaction
QSqlDatabase::database().transaction();
// determine highest payment number
string maxquerystring = "select max(paymentno) from penalties";
QSqlQuery maxquery(maxquerystring.c_str());
maxquery.next();
int maxpaymentno = maxquery.value(0).toInt();
// insert new penalty
string insertString = "insert into penalties values (";
insertString += to_string(maxplayerno+1);
insertString += ",39,'2016-06-29',150)";
QSqlQuery insertStmt(insertString.c_str());
// commit transaction
QSqlDatabase::database().commit();
```



Why primitive ?

- SQL is just a string: no compile time syntax check
- No validation of table and field names
- Frequent typing errors: blanks, quotes, …
- Convert every input variable to string
- Change of SQL statement → recompile and build

Use bind variables: the better approach

Oracle style:

```
QSqlQuery insertStmt2;
insertStmt2.prepare("insert into penalties values (:paymentno,:playerno,:date,:amount)");
insertStmt2.bindValue(":paymentno",maxpaymentno+1);
insertStmt2.bindValue(":playerno",57);
insertStmt2.bindValue(":date",QDate(2016,6,28));
insertStmt2.bindValue(":amount",200);
insertStmt2.exec();
```

ODBC style:

```
QSqlQuery insertStmt2;
insertStmt2.prepare("insert into penalties values (?,?,?,?)");
insertStmt2.bindValue(maxpaymentno+1);
// …
```

Why better?

- No conversions to string necessary
- Ideal for inserting records in a loop: prepare once, bind and exec many times
- Less error prone

But:

- Still no compile time check
- Still rebuild if SQL statement changes

Let Qt generate SQL

```
QSqlTableModel model;
model.setTable("players");
model.setFilter("town in ('Stratford','Inglewood')");
model.setSort(1,Qt::AscendingOrder);
model.select();
```

Let Qt generate SQL: very limited use

- Only support for simple INSERT, DELETE, UPDATE statements
- Very limited support for SQL: no JOIN, GROUP BY / HAVING, subqueries, …
- Result will probably be that too much data goes "over the wire" and is filtered in the application
- Software engineering tip:
  - Use the database for data handling
  - Use the application for business logic

Link table contents to GUI objects

› Suppose we need an editable form for the PLAYERS table

In class Player:

class Player { public:

enum index

{PLAYERNO=0,NAME,INITALS,BIRTH_DATE,SEX,JOINED,STREET,HOUSENO,POSTCODE,TOWN,PHONENO,LEAGUENO}

…

};

In client:

```
QSqlTableModel model;
model.settable ("players");
model.setHeaderData (Player::NAME,Qt::Horizontal,tr("Name"));
model.setHeaderData (Player:: INITIALS,Qt::Horizontal,tr("Initials"));
// …
model.select();
```

Link table contents to GUI objects

In client (continued)

```
QTableView view;
view.setModel(model);
view.setSelectionMode(QAbstractItemView::SingleSelection);  // single field selection
view.setSelectionBehavior(QAbstractItemView::SelectRows);   // highlighted by row
view.setColumnHidden(Player::PLAYERNO, true);               // PLAYERNO invisible
view.resizeColumnsToContents();
view.setEditTriggers(QAbstractItemView::NoEditTriggers);    // table view is read-only
QHeaderView header = view.horizontalHeader();               // provides header row
Header.setStretchLastSection(true);   //  last visible section in the header takes up all the
                                      //available space
```

Call stored procedures

› Which players, from a parameterizable town, got more penalties than they played matches?

```
-- correlated subquery
select name,playerno,
(select count(*) from penalties where playerno=p.playerno) pen,
(select count(*) from matches where playerno=p.playerno) mats
from players p
where
town = 'Stratford' and
(select count(*) from penalties where playerno=p.playerno) >
(select count(*) from matches where playerno=p.playerno);
```

Call stored procedure

› In MS SQL Server:

```
create procedure MorePenaltiesThanMatches @town char(10)
as
select name,playerno,
(select count(*) from penalties where playerno=p.playerno) pen,
(select count(*) from matches where playerno=p.playerno) mats
from players p
where
town = @town and
(select count(*) from penalties where playerno=p.playerno) >
(select count(*) from matches where playerno=p.playerno);
```

Call stored procedure

› In MySQL:

```
create procedure  MorePenaltiesThanMatches (in p_town char(10))
begin
select name,playerno,(select count(*) from penalties where playerno=p.playerno) pen,
(select count(*) from matches where playerno=p.playerno) mats
from players p
where town = p_town and
(select count(*) from penalties where playerno=p.playerno) >
(select count(*) from matches where playerno=p.playerno);
end
```

Call stored procedure

› In Oracle:

```
CREATE OR REPLACE PROCEDURE MorePenaltiesThanMatches(p_town in players.town%type,
                                                     p_recordset OUT SYS_REFCURSOR)
as
BEGIN
OPEN p_recordset FOR
select name,playerno,(select count(*) from penalties where playerno=p.playerno) pen,
(select count(*) from matches where playerno=p.playerno) mats
from players p
where town = p_town and
(select count(*) from penalties where playerno=p.playerno) >
(select count(*) from matches where playerno=p.playerno);
END;
```

Call stored procedure

```
QString querystring = "{ CALL MorePenaltiesThanMatches (?) }";
QSqlQuery query4;
bool prepOk = query4.prepare( querystring );
if (!prepOk)
{
  ostringstream msg;
  msg << __FILE__ << ": " << __LINE__ << query4.lastError().text().toStdString();
  throw(DBConnector::DBException(msg.str()));
}

query4.bindValue(0,"Stratford", QSql::In );
bool queryOk = query4.exec(); //Returns true
if (!queryOk) { /*error processing*/ }

while (query4.next())
{
  cout << query4.value(0).toString().toStdString() << " ";
  cout << query4.value(1).toInt() << " ";
  cout << query4.value(2).toInt() << " ";
  cout << query4.value(3).toInt() << endl;
}
```

Call stored procedure: + & -

+

- SQL code outside C++
- SQL code validated at dev time
- SQL code tested outside C++ program
- SQL code can be changed without rebuild of C++
- Ideal for single SQL statement reports or CRUD operations
- More secure, guard against SQL injection attacks
- Build an API to your database:

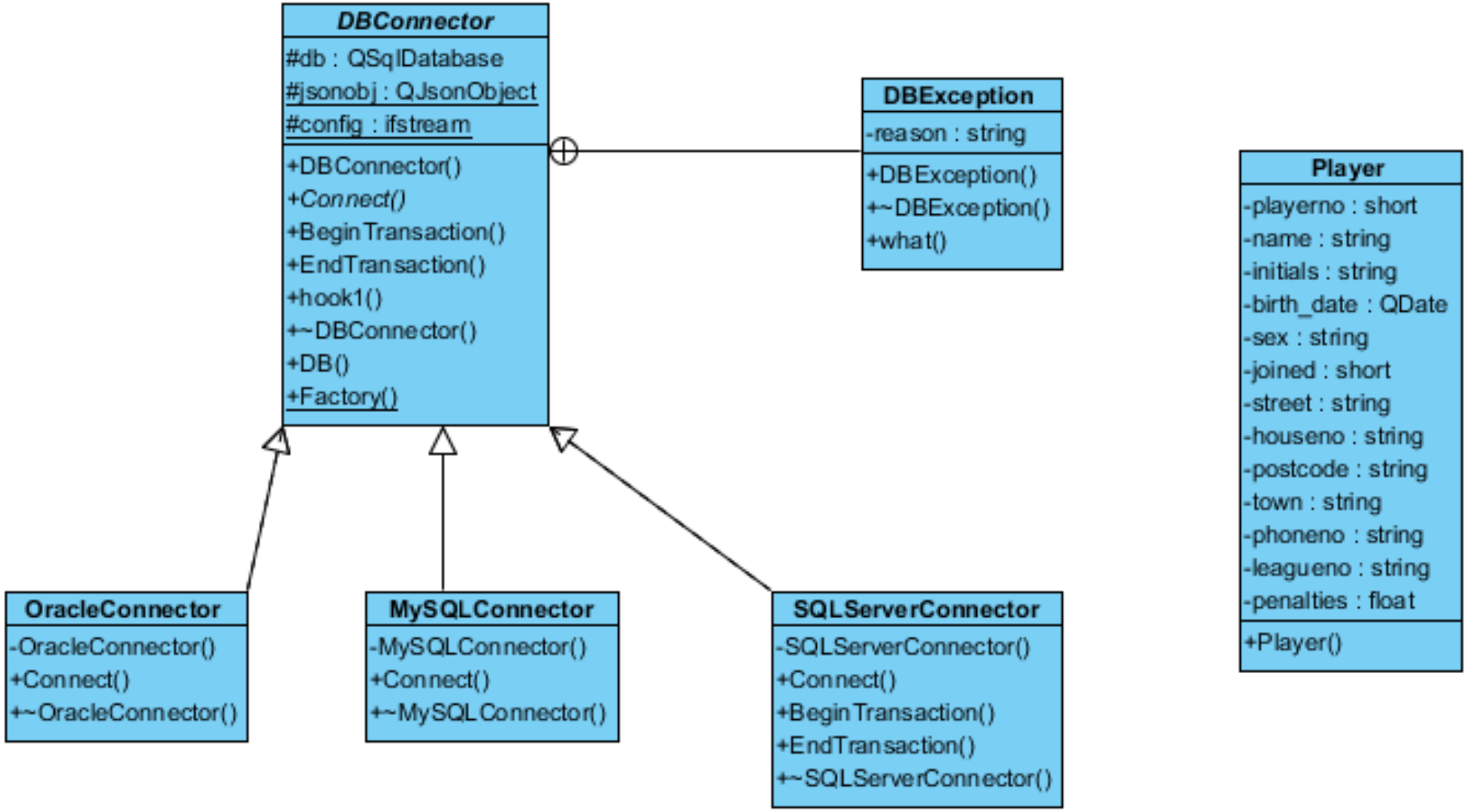
-

- Syntax is database dependent
- Don't use for business logic

Contents

- What is Qt?
- The importance of database independence
- The Qt API for database access
  - Plain SQL
  - Generate SQL
  - Link table content to GUI objects
  - Stored Procedures
- **Demo**
- Drawbacks and remarks
- Alternatives
- Conclusions
- Q&A

Demo


DBConnector
#db : QSqlDatabase
#jsonobj : QJsonObject
#config : ifstream
+DBConnector()
+Connect()
+BeginTransaction()
+EndTransaction()
+hook1()
+~DBConnector()
+DB()
+Factory()
DBException
-reason : string
+DBException()
+~DBException()
+what()
Player
-playerno : short
-name : string
-initials : string
-birth_date : QDate
-sex : string
-joined : short
-street : string
-houseno : string
-postcode : string
-town : string
-phoneno : string
-leagueno : string
-penalties : float
+Player()
OracleConnector
-OracleConnector()
+Connect()
+~OracleConnector()
MySQLConnector
-MySQLConnector()
+Connect()
+~MySQLConnector()
SQLServerConnector
-SQLServerConnector()
+Connect()
+BeginTransaction()
+EndTransaction()
+~SQLServerConnector()

Contents

Drawbacks and remarks

- No native support available for MS SQL Server, only ODBC
- Community support for native libraries from Oracle and MySQL is limited.
- It's not an ORM tool
  - still make your own SQL statements
  - which might also be an advantage:
    - enhanced control
    - better performance (avoid SELECT *)
- QtSQL (and Qt in general) comes with its own string and collection classes, no support for std::string, std::vector, …

Alternatives

- Poco library offers similar functionality
  - SQLite, MySQL, ODBC
  - no native Oracle support
  - Supports std::vector, std::set, std::map
- ORM tools
  - http://www.codesynthesis.com/products/odb/

Conclusions

- Use Qt framework as a one-stop shopping
- It's not an ORM tool
- Use design patterns based on polymorphism to handle database specifics
- Stored procedure languages are proprietary but allow modifying SQL without recompiling and offer syntax check at development time
- "Encapsulate what varies": store your SQL code in the database
- Good GUI support
- If one-stop shopping is not an issue consider POCO library for its better support for modern C++

References and Literature

- http://doc.qt.io
- https://wiki.qt.io/QtWhitepaper
- http://pocoproject.org
- http://www-cs.ccny.cuny.edu/~wolberg/cs221/qt/books/C++-GUI-Programming-with-Qt-4-1st-ed.pdf

Q & A