Ressource R2.02 – chapitre 12

Focus sur le développement d'applications avec IHM exploitant une base de données

T. Nodenot

Thierry.Nodenot@iutbayonne.univ-pau.fr

Plan

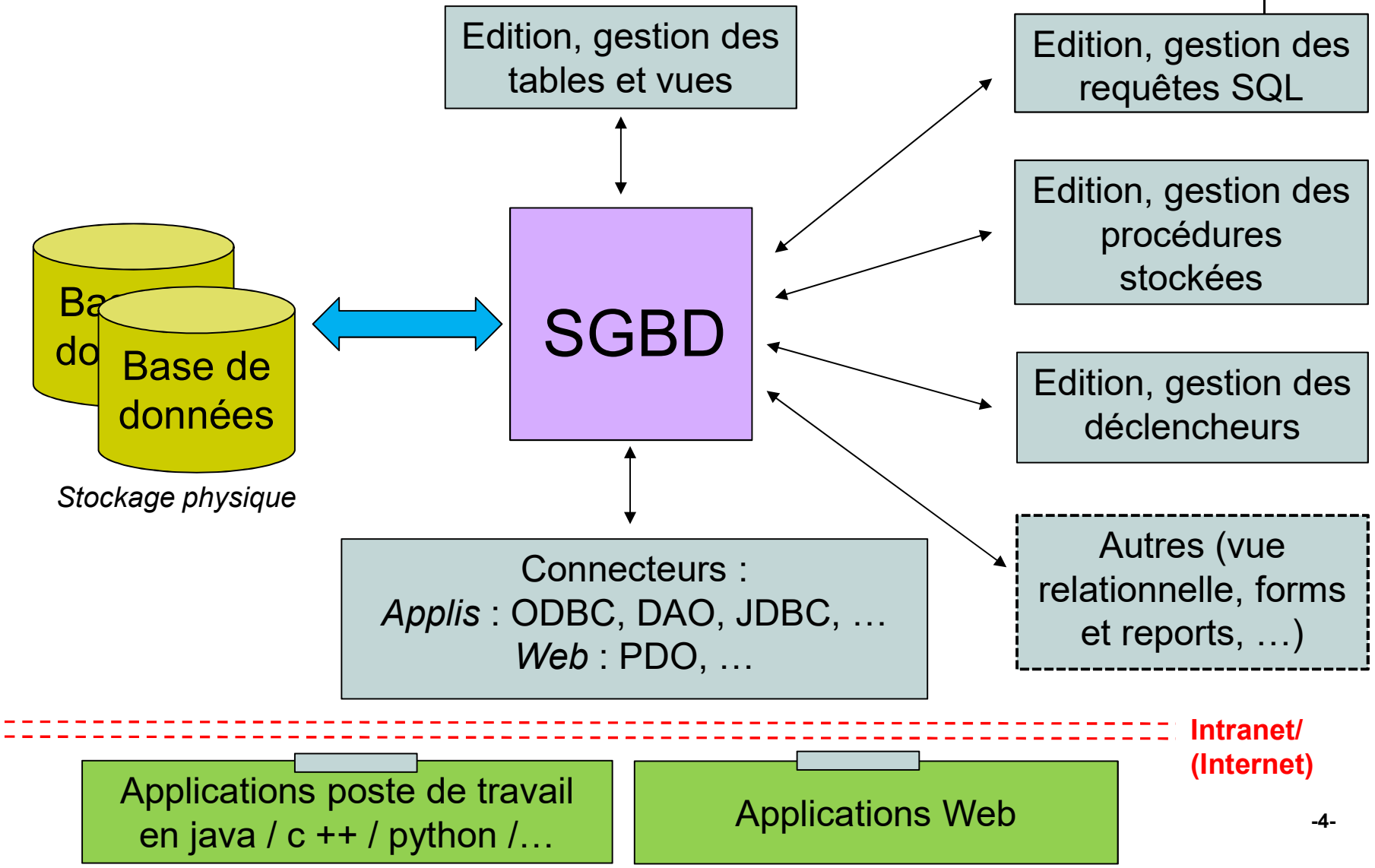
1. Rappels des principes d'exploitation d'une base de données via une application informatique

2. Composants de base offerts par Qt pour exploiter une base de données relationnelle

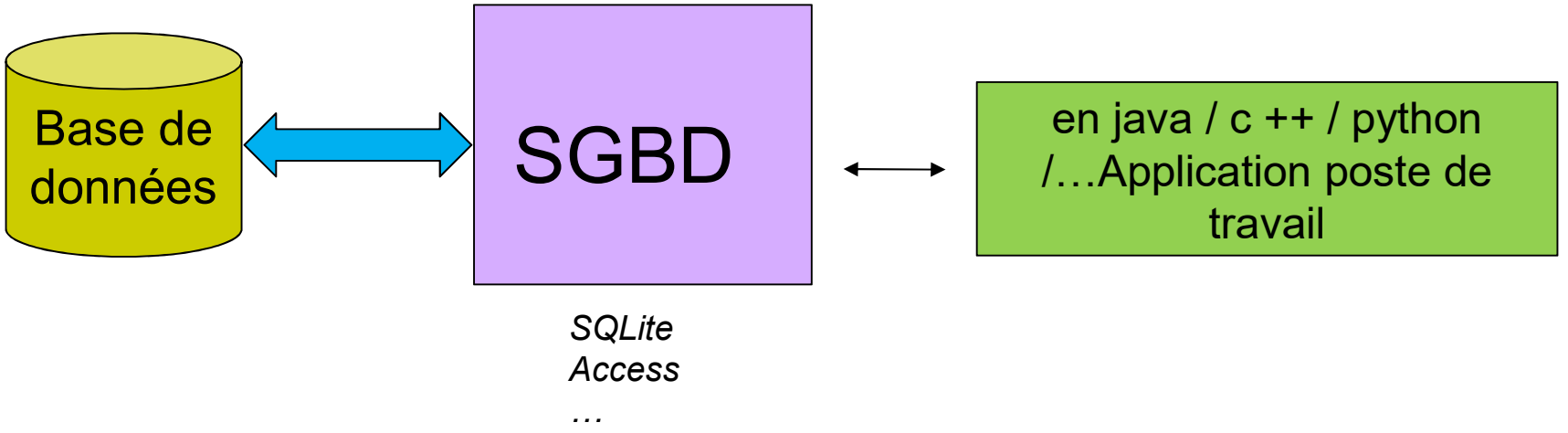
3. Composants spécifiques offerts par Qt pour exploiter un modèle de données relationnel à travers une interface graphique

Partie 1 : Rappels des principes d'exploitation d'une base de données via une application informatique

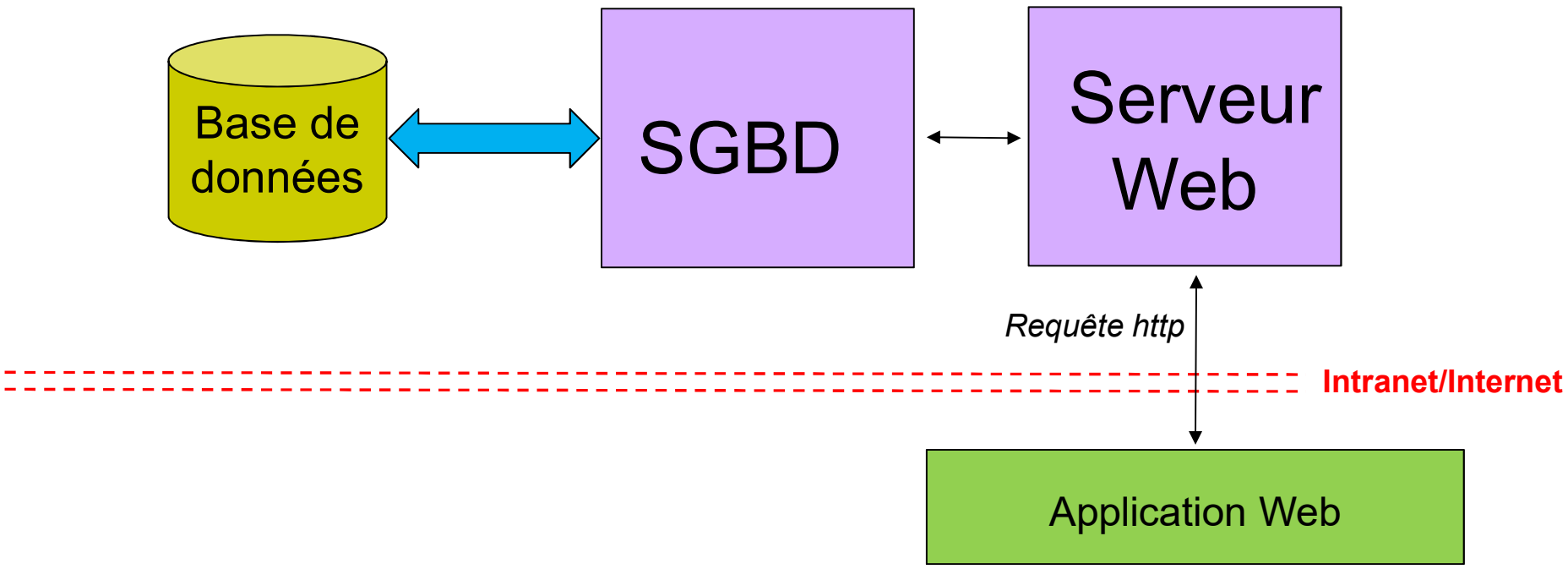
Les SGBD et leur environnement



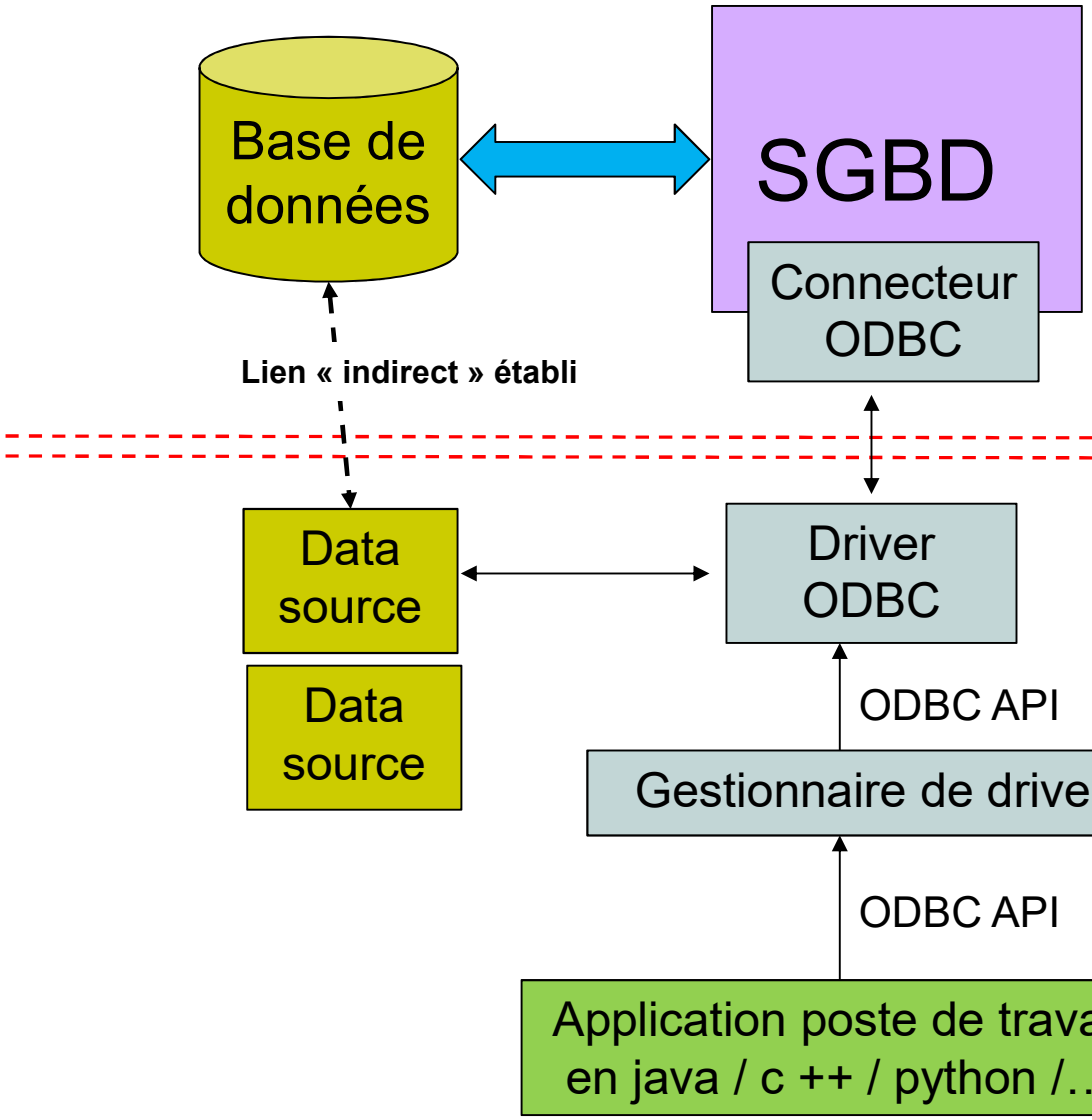
Exploitation d'un SGBD exploité localement



Exploitation d'un SGBD à travers une connexion Web



Exploitation d'un SGBD à travers une connexion ODBC (1)



Traite les appels ODBC et soumet chaque requête SQL à la source de données destinataire via le SGBD

Charge le driver requis pour chaque application demandeuse

Traite *et fait appel aux fonctions ODBC, soumet les requêtes SQL*

Exploitation d'un SGBD à travers une connexion ODBC (2)

- Côté serveur :
  - Le serveur de Base de Données est opérationnel et écoute les éventuelles requêtes (port 3306)
  - Le serveur gère les authentifications des utilisateurs et les droits/privilèges de chacun sur chaque base de données et chaque table

- Côté Client (PC / Bureau virtuel)
  - Le développeur définit le DSN par son nom (cf. sources ODBC64) et ses autres paramètres : adresse du serveur mySQL et port (port 3306), login et mot de passe sur le SGBD, base de données à utiliser via le DSN
  - Le développeur exploite le DSN dans les applications qu'il code

Partie 2 : Composants de base offerts par Qt pour exploiter une base de données relationnelle

Les classes offertes par Qt et leur niveau de spécificité

Services	Classe QT	Classe spécifique à Qt ?
Database driver layer	QSqlDriver	standard
Database connection layer	QSqlDatabase	standard
Database execution layer	QSqlQuery	standard
Result set	QSqlRecord QSqlField QSqlIndex QSqlResult	standard standard standard standard
Result set et presentation layer	QSqlQueryModel QSqlTableModel QSqlRelationalTableModel	spécifique spécifique spécifique
Database transaction layer	QSqlDatabase.commit(); QSqlDatabase.rollback();	standard standard
Database exception	QSqlError	standard

Les classes QSqlDriver et QSqlDatabase

classe	Explications
QSqlDriver	C'est une classe abstraite qui va permettre d'établir un lien ou un accès vers une base de données.

https://doc.qt.io/qt-5/qsqldriver.html

classe	Explications
QSqlDatabase	Cette classe fournit des méthodes pour se connecter à une base de données, gérer les accès et contrôler les transactions sur la base. Les méthodes permettent aussi de déterminer le driver à utiliser et d'obtenir des informations sur ce driver, d'exécuter en direct des requêtes, d'obtenir la clé primaire d'une table ou des informations sur un champ d'une table.

https://doc.qt.io/qt-5/qsqldatabase.html

```
QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
 db.setHostName("postgresServeur");
 db.setDatabaseName("madb");
 db.setUserName("mojito");
 db.setPassword("J0a1m8");
 bool ok = db.open();
```

La classe QSqlQuery (1)

classe	Explications
QSqlQuery	Cette classe est utilisé pour définir et exécuter des requêtes SQL qu'elles soient ou non paramétrées. Dans ce dernier cas, la méthode boundValue() contient les valeurs des paramètres passés à la requête. L'exécution renvoie un resultSet constitué d'un ensemble d'enregistrements. QSqlQuery permet d'accéder au jeu de résultats un enregistrement à la fois. Après l'appel à exec (), le pointeur interne de QSqlQuery est situé à une position "before" le premier enregistrement. Nous devons appeler QSqlQuery::next () une fois pour passer au premier enregistrement, puis next () à plusieurs reprises pour accéder aux autres enregistrements, jusqu'à ce qu'il renvoie false .

https://doc.qt.io/qt-5/qsqlquery.html

```
QSqlQuery query;
 query.exec("SELECT name, salary FROM employee WHERE salary > 3000");

while (query.next()) {
        QString name = query.value(0).toString();
        int salary = query.value(1).toInt();
        qDebug() << name << salary;
    }
```

La classe QSqlQuery (2)

Exécution de requêtes et exécution de requêtes paramétrées

```
QSqlQuery query;
query.exec("INSERT INTO employee (id, name, salary) "
           "VALUES (1001, 'Jean Marcelin', 35000)");
```

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
              "VALUES (:id, :name, :salary)");
query.bindValue(":id", 1001);
query.bindValue(":name", "Jean Marcelin");
query.bindValue(":salary", 65000);
query.exec();
```

ou

```
QSqlQuery query;
query.prepare("INSERT INTO employee (id, name, salary) "
              "VALUES (?, ?, ?)");
query.addBindValue(1001);
query.addBindValue("Jean Marcelin");
query.addBindValue(65000);
query.exec();
```

La classe QSqlQuery (3)

Utiliser la méthode execbatch() au lieu de la méthode exec()

```
QSqlQuery q;
q.prepare("insert into maTable values (?, ?)");

QVariantList ints;
ints << 1 << 2 << 3 << 4;
q.addBindValue(ints);

QVariantList names;
names << "Louis" << "Michel" << "Alain" << QVariant(QVariant::String);
q.addBindValue(names);

if (!q.execBatch())
    qDebug() << q.lastError();
```

Les classes QSqlRecord, QSqlField et QSqlIndex (1)

classe	Explications
QSqlRecord	Cette classe est utliisée pour exploiter un enregistrement issu d'une table de la base ou d'un requête. un QSqlRecord peut contenir 0 ou plusieurs qsqlfields. Vous pouvez utiliser la méthode field() les qsqlfields contenus dans un QSqlRecord, ou utiliser la méthode value() pour récupérer les valeurs de chaque field (la velur retournée est QVariant qu'il faut convertir en accord avec le type réel de ce champ).
QSqlField	Cette classe est utilisée pour traiter les champs d'une table ou d'une vue / requête. Grâceà la classe QSqlField vous obtenez des informaitons sur le nom du champ, e nom de la table, vous pouvez savoir si ce champ peut être à Null, sa valeur par défaut, sa taille, son type, ...
QSqlIndex	Cette classe permet d'exploiter les index contenus dans une base de données. Avec cette class, vous pouvez obtenir le nom des index, savoir si l'index organise les enregistrements de manière ascendante ou descendante.

https://doc.qt.io/qt-5/qsqlrecord.html
https://doc.qt.io/qt-5/qsqlfield.html
https://doc.qt.io/qt-5/qsqlindex.html

Les classes QSqlRecord, QSqlField et QSqlIndex (2)

```
QSqlQuery query;
QSqlQuery q("select * from Clients");
…
QSqlRecord record = q.record(); // Acces à l'enregistrement courant
for(int i=0;i<record.count();i++){
    QSqlField field=record.field(i);
    qDebug()<<field.name()<<field.value();
}
```

La classe QSqlError

classe	Explications
QSqlError	Cette classe pemet de traiter les erreurs constatées lors de accès à la base de données. La méthode QSqlError::isValid() renvoie vrai s'il n'y a pas d'erreur. Pour en savoir plus sur l'erreur les autres méthodes de la classe permettent de connaitre le texte d'erreur et d'autres informations générées par le SGBD, ...

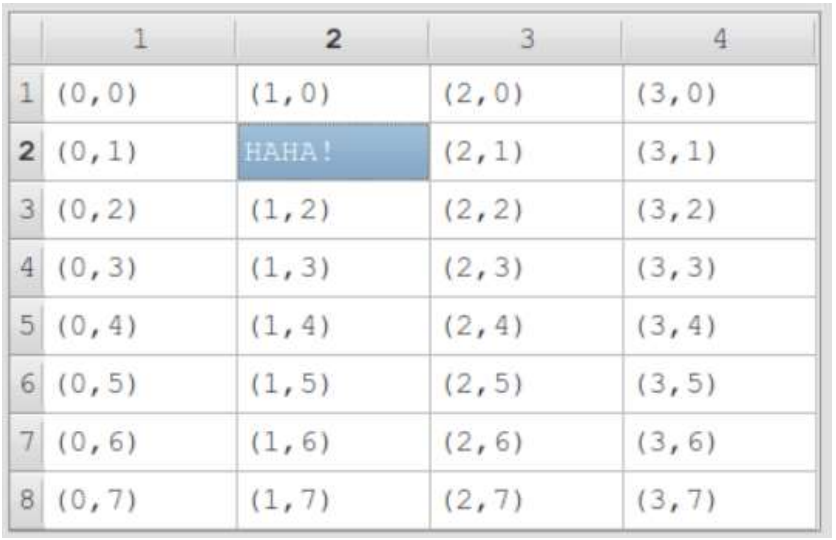
https://doc.qt.io/qt-5/qsqlerror.html

```
QSqlQuery query;
bool bSuccess=query.exec("delete from Articles WHERE PrixUHT > 1.5");
if (!bSuccess)
   {
       QSqlError lastError = query.lastError();
       qDebug() << lastError.driverText()
                <<QString(QObject::tr("select error"));
   }
```

Complément afficher des données de manière tabulaire avec Qt ?

QTableWidget

QTableView



classe	Explications
QTableWidget	Cette classe fournit une implémentation d'une vue sous la forme d'une table.
QTableView	Cette classe fournit une implémentation par défaut d'un modèle/vue sous la forme d'une vue en table. Une QTableView implémente une vue en table qui affiche les éléments contenus dans un modèle référencé par ce composant

QTableView est un composant graphique spécifique qui sera vu lors du prochain cours …

Le composant QTableWidget (1)

classe	Explications
QTableWidget	Cette classe fournit une implémentation d'une vue sous la forme d'une table.

https://doc.qt.io/qt-5/qtablewidget.html

```
QTableWidget *tableWidget = new QTableWidget(this);

QTableWidgetItem *element = new QTableWidgetItem(QString("HAHA!"));

tableWidget->insertRow(0);
tableWidget->insertRow(1);
…

tableWidget->setItem(1, 1, element);
…
```

Le composant QTableWidget (2)

```
QTableWidget *tableWidget = new QTableWidget(this);

QStringList header; // nom des colonnes
header << "Nom" << "Prénom"; // ...

// On fixe le nombre de colonnes
tableWidget->setColumnCount(header.size());
// On applique les noms des colonnes
tableWidget->setHorizontalHeaderLabels(header);

// on cache les numéros de ligne
tableWidget->verticalHeader()->setHidden(true);

QHeaderView * headerView = tableWidget->horizontalHeader();
// on redimensionne automatiquement la colonne pour occuper l'espace
disponible
headerView->setSectionResizeMode(QHeaderView::Stretch);
```

Partie 3 : Composants spécifiques offerts par Qt pour exploiter un modèle de données relationnel à travers une interface graphique