11. Qt – databáze, MVC

B2B99PPC – Praktické programování v C/C++

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Relační databáze

- **Databáze** je místo, kam lze ukládat data definovat operace, které s nimi lze provádět.
- Základem relačních databází jsou databázové **tabulky**, které jsou na sebe určitým způsobem závislé – existuje mezi nimi jistá logická vazba (**relace**).
- Tabulky jsou též nazývány entitami – jsou chápány jako prvek reálného světa.
- Každá tabulka je tvořena **sloupci** a **řádky**, sloupce reprezentují vlastnosti této entity.
  - Každý sloupec musí mít jedinečný název a určený datový typ podle toho, jaká data jsou vněm uložena (číslo, text, logická hodnota, atd.).
  - Řádky tabulky reprezentují samotné záznamy v databázové tabulce
- Každý řádek by měl mít svůj jedinečný identifikátor, podle kterého bude možné určit příslušný záznam – **klíč**.
  - Primární klíče slouží jako jednoznačný (unikátní) identifikátor záznamu (řádku) tabulky.
  - Cizí klíče slouží k vyjádření vztahů (relací) mezi jednotlivými tabulkami
- Pro manipulaci s daty se nejčastěji využívá jazyk SQL (Structured Query Language)

SQL

- Standardizovaný jazyk relačních databází, vytvořený po představení relačního modelu
- Neprocedurální jazyk – je třeba zadat, jaké informace požadujeme, nikoli jak je získat
- Ve skutečnosti nejde o jeden jazyk, ale o různé dialekty různých databázových systémů (ORACLE, MS SQL, MySQL, ...)

SQL příkazy

- **DML** (Data Manipulation Language) – množina příkazů pro manipulaci s daty.
- **DDL** (Data Definition Language) – množina příkazů pro definici dat.
- **DCL** (Data Control Language) – množina příkazů pro definici řízení přístupových práv.
- **TCL** (Transaction Control) – množina příkazů pro řízení transakcí.

DML – příkazy pro manipulaci s daty

- SELECT – vybírá data z databáze.
- INSERT – vkládá data do databáze.
- UPDATE – edituje data v databázi.
- DELETE – odstraňuje data z celých tabulek nebo řádky tabulek odpovídající podmínce.
- CALL – volá uložené procedury.
- LOCK TABLE – zamyká celé tabulky:
  - zámky čtení (READ)
  - zámky zápisu (WRITE)

DDL – příkazy pro definici dabázových struktur

- CREATE – vytváří objekty databáze (i databázi samotnou).
- ALTER – mění strukturu databázových objektů.
- DROP – odstraňuje objekty databáze (tabulky, indexy, databáze).
- TRUNCATE – odstraní všechny záznamy z tabulky, přičemž zachová strukturu tabulky (objekt tabulky).
- COMMENT – umožní přidávat komentáře k objektům databáze.
- RENAME – změna názvu objektu databáze

DCL – příkazy pro řízení uživatelských práv

- GRANT – nastavuje uživatelská práva k tabulkám databáze.
- REVOKE – odebírá přístupová práva přidělená příkazem GRANT.

TCL – příkazy pro řízení transakcí

- COMMIT – potvrzení provedení transakce.
- SAVEPOINT – definuje záchytný bod transakce, ke kterému se lze během provádění transakce vrátit.
- ROLLBACK – ruší transakci a vrací se zpět ke stavu původnímu.
- START TRANSACTION – zahajuje blok transakce.
- SET TRANSACTION – umožňuje změnit level izolace transakce globálně nebo pro aktuální session.

Tabulky

Table: characters

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock
3	Daenerys	Dragon Stone

Table: appearance

episode	character_id	appeared
1	1	10
1	1	5
1	1	7
2	1	5
2	2	5
2	3	9
3	1	10
3	2	2

Základní datové typy

- Integer (INT) – celé číslo
- Float (REAL) – raálná čísla
- Varchar (VARCHAR) – krátký řetězec do 255 znaků
- Text (TEXT) – delší text
- Binární data (BLOB) – obecná data, např. obrázky
- Boolean (BOOLEAN) – logická hodnota
- Date (DATE) – datum ve formátu YYYY-MM-DD
- Time (TIME) – čas ve formátu HH:mm:ss
- Timestamp (TIMESTAMP) – časová značka ve formátu YYYY-MM-DD HH:mm:ss

SQL – vytváření, změna a rušení tabulek

```
CREATE TABLE characters
(
  id INT PRIMARY KEY,
  name VARCHAR (255),
  home VARCHAR (255)
);
```

```
ALTER TABLE characters MODIFY name VARCHAR (100) NOT NULL;
```

```
DROP TABLE IF EXISTS characters;
```

SQL – vkládání, mazání a změna řádků

```
INSERT INTO characters
VALUES (1, 'John', 'Winterfall');
INSERT INTO characters
VALUES (2, 'Tyrion', 'Casterly Rock');
INSERT INTO characters
VALUES (3, 'Daenerys', 'Dragon Stone');
```

id	name	home
1	John	Winterfall
2	Tyrion	Casterly Rock
3	Daenerys	Dragon Stone

```
DELETE FROM characters
WHERE name = 'John' OR id = 3;
```

id	name	home
2	Tyrion	Casterly Rock

```
UPDATE characters SET
home = 'Winterfell' WHERE id = 1
```

id	name	home
1	John	Winterfell

SQL – dotaz v jedné tabulce

```
SELECT id, name FROM characters
WHERE id < 3;
```

id	name
1	John
2	Tyrion

```
SELECT * FROM characters
WHERE id < 3;
```

id	name	home
1	John	Winterfell
2	Tyrion	Casterly Rock

SQL – dotaz ve více tabulkách, agregace

```
SELECT name, episode, appeared
FROM characters, appearance
WHERE id = character_id
AND episode = 1
AND appeared < 10
```

name	episode	appearance
john	1	5
daenerys	1	7

```
SELECT name, sum(appeared) as sum
FROM characters, appearance
WHERE id = character_id
GROUP BY id
```

name	sum
John	20
Tyrion	5
Daenerys	16

SQLite

- Jednoduchá databáze , která ukládá data v binární formě na lokální paměťové médium
- Podmnožina SQL jazyka
- Řádkový klient, podpora v řadě progamovacích jazyků včetně C, C++, Python, ...
- Uživatelský profil v prohlížeči Mozilla, lokální storage v Androidu, ...
- Celá řada GUI nástrojů: SQLiteStudio SQLite Database Browser

Databáze v Qt

- Qt poskytuje unifikované připojení k databázím
  - QtSql – přímé SQL dotazy
  - model/view architektura (MVC)
- Dostupné ovladače (open-source edice)
  - MySQL, PostgreSQL, MSSQL, ODBC, DB2 (IBM), Oracle, Sybase
  - SQLite
- Nastavení projektu: `QT += sql`

Připojení k SQLite

```
1  QSqlDatabase db;
2  QString filename ("got.db");
4  db.addDatabase("QSQLITE");
5  db.setDatabaseName(filename);
7  if (!db.open())
8  {
9    qDebug() << db.lastError().text();
10 }
```

lec13/01-sql

SQL dotazy – QSqlQuery

- Výsledkem dotazu je záznam (recordset), ve kterém je možné se pohybovat kurzorem
  - QSqlQuery::next() , QSqlQuery::previous() , QSqlQuery::first() , QSqlQuery::last()
- SQL dotaz může být argumentem konstruktoru nebo funkce QSqlQuery::exec()
- Položky aktuálního řádku jsou dostupné pomocí funkce QSqlQuery::value()
  - argumentem funkce je index sloupce (číslováno o 0)
  - datový typ návratové hodnoty je QVariant
  - pro cílový datový typ existují konverzní metody

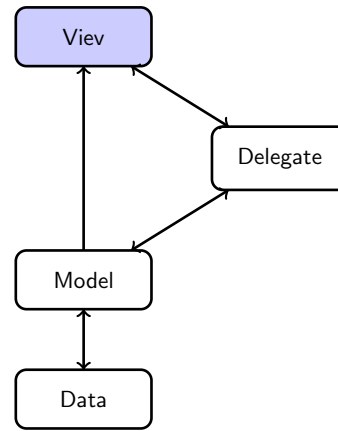
```
1  QSqlQuery query ("SELECT name FROM characters");
3  while (query.next())
4  {
5    QString name = query.value(0).toString();
6    qDebug() << name;
7  }
```

SQL dotazy – binding

```
1  QSqlQuery query;
3  query.prepare("INSERT INTO characters (id, name, home) "
4                "VALUES (:id, :name, :home)");
6  query.bindValue(":id", 4);
7  query.bindValue(":name", "Eddard");
8  query.bindValue(":home", "Winterfell");
9  query.exec();
11 // alternativne
12 query.prepare("INSERT INTO Characters (id, name, home) "
13               "VALUES (?, ?, ?)");
15 query.bindValue(0, 5);
16 query.bindValue(1, "Joffrey");
17 query.bindValue(2, "Red Keep");
18 query.exec();
```

Model-View-Controller (MVC)

- Architektura pro návrh aplikace s UI
- Rozděluje aplikaci na tři nezávislé komponenty
  - **Model** – datový model, se kterým aplikace pracuje
  - **View** – uživatelské rozhraní, zajišťuje vizualizaci dat aplikace do podoby vhodné k prezentaci
  - **Controller** – řídicí logika, zajišťuje změny dat nebo vizualizace nazákladě událostí (typicky od uživatele)
- Komponenty jsou na sobě nezávislé – změna jedné komponenty neovliňuje ostatní
  - Důležitost vhodně definovaného rozhraní



Model/View programming tutorial (Qt)

MVC – příklad

```
1  QApplication app{argc, argv};
3  QSplitter splitter{Qt::Vertical};
5  auto model = new QStringListModel{&app};
7  model->setStringList(
8      QStringList{"Gandalf", "Aragorn", "Legolas", "Samwise Gamgee" ,"
       Gimli", "Bilbo Baggins", "Peregrin Took", "Boromir"}
9    );
11 auto combo_box_view = new QComboBox{&splitter};
13 combo_box_view->setModel(model);
15 auto list_view = new QListView{&splitter};
17 list_view->setModel(model);
19 splitter.show();
```

lec13/02-mvc

Modely pro práci s SQL

- Kromě velmi přímočarého přístupu k SQL datům pomocí QSqlQuery nabízí Qt ještě třídy s vyšší úrovní abstrakce:
  - QSqlQueryModel – read-only model založený na SQL dotazech
  - QSqlTableModel – read-write model pro práci s jednou tabulkou
  - QSqlRelationalTableModel – podtřída `QSqlTableModel` s podporou cizích klíčů
- Třídy jsou navrženy tak, aby umožňovali snadnou prezentaci v komponentách jako jsou QListView a QTableView .
- Výhodou je snadná změna datového zdroje
- Podpůrné třídy:
  - QSqlRecord
  - QSqlField

QSqlQueryModel

```
1  QSqlQueryModel model;
2  model.setQuery("SELECT * FROM characters");
4  for (int i = 0; i < model.rowCount(); ++i) {
5    int id = model.record(i).value("id").toInt();
6    QString name = model.record(i).value("name").toString();
7    qDebug() << id << name;
8  }
10 model.setHeaderData(0, Qt::Horizontal, "id");
11 model.setHeaderData(1, Qt::Horizontal, "name"));
12 model.setHeaderData(1, Qt::Horizontal, "name"));
14 QTableView view;
15 view.setModel(model);
16 view.show();
```

lec13/03-sql-model

QSqlTableModel

```
1  QSqlTableModel model;
2  model.setTable("characters");
3  model.setFilter("id > 1");
4  model.setSort(2, Qt::DescendingOrder);
5  model.select();
6  //
7  model.insertRows(row, 1);
8  model.setData(model.index(row, 0), 4);
9  model.setData(model.index(row, 1), "Eddard");
10 model.setData(model.index(row, 2), "Winterfell");
11 model.submitAll();
12 //
13 model.removeRows(row, 5);
14 model.submitAll();
```

lec13/04-sql-table-model