Qt — databáze a vizualizace

SQL · QtSql · NoSQL · Qt Charts

Stanislav Vítek · Katedra radioelektroniky · ČVUT FEL

Přehled přednášky

I. Relační databáze a SQL

- Klíčové pojmy, datový model
- DDL — CREATE TABLE, typy dat
- DML — INSERT, UPDATE, DELETE
- DQL — SELECT, JOIN, GROUP BY

I½. MVC architektura

- Model-View-Controller — koncept
- Qt Model/View/Delegate
- `QAbstractItemModel` — vlastní model

II. Qt SQL modul

- `QSqlDatabase` — připojení k DB
- `QSqlQuery` — přímé dotazy
- Prepared statements, bezpečnost
- `QSqlTableModel` + `QTableView`
- Transakce, chyby

III. NoSQL a alternativní úložiště

- Typy NoSQL databází
- `QSettings` — klíč/hodnota
- JSON soubory jako úložiště
- SQLite JSON1, Redis, embedded DB

IV. Vizualizace dat — Qt Charts

- `QChart`, `QChartView`
- Typy grafů: linie, sloupce, koláč
- Real-time aktualizace
- Customizace, alternativy

I. Relační databáze a SQL

Datový model · DDL · DML · DQL

Co je relační databáze

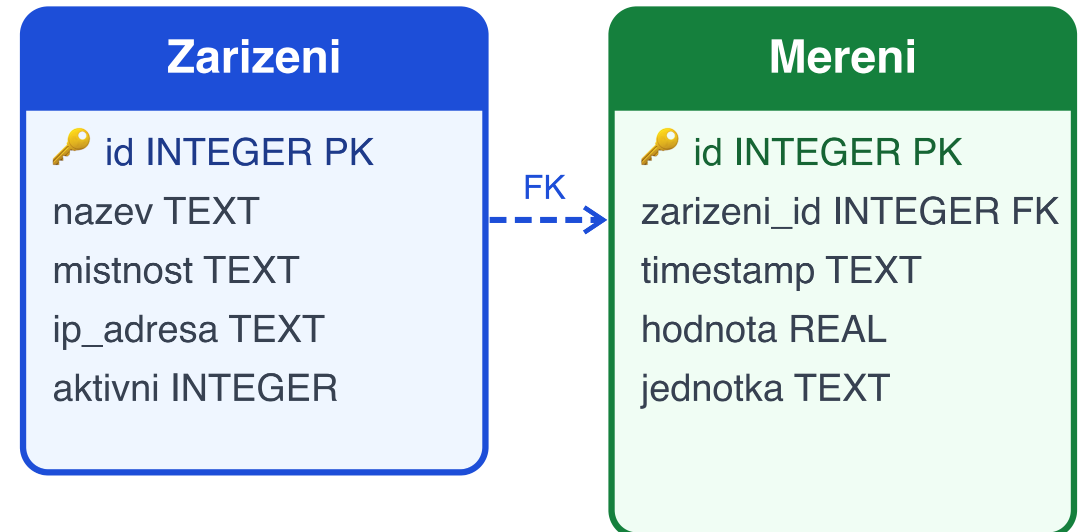
Relační databáze ukládá data ve formě **tabulek** (relací). Tabulky jsou propojeny **vztahy** přes klíče.

Klíčové pojmy

Pojem	Vysvětlení
Tabulka	sada řádků se stejnou strukturou
Řádek (záznam)	jedna entita (jeden senzor, jeden uživatel)
Sloupec (atribut)	vlastnost entity
Primární klíč	jednoznačný identifikátor řádku
Cizí klíč	odkaz na primární klíč jiné tabulky
Index	urychlení vyhledávání

Oblíbené RDBMS

- **SQLite** — embedded, bez serveru, soubor `.db`
- **PostgreSQL** — open-source, plně funkční
- **MySQL / MariaDB** — web a enterprise
- **Microsoft SQL Server** — enterprise Windows



Mereni — ukázka dat

id	zarizeni_id	timestamp	hodnota
1	1	2025-04-01 10:00:00	23.4
2	1	2025-04-01 10:01:00	23.6

SQL — kategorie příkazů

SQL se dělí do čtyř skupin příkazů dle jejich účelu.

DDL — Data Definition

Definice struktury

```
-- Vytvoření tabulky
CREATE TABLE zarizeni (
  id       INTEGER PRIMARY KEY,
  nazev    TEXT NOT NULL,
  ip       TEXT,
  aktivni  INTEGER DEFAULT 1
);

-- Úprava tabulky
ALTER TABLE zarizeni
  ADD COLUMN popis TEXT;

-- Smazání tabulky
DROP TABLE IF EXISTS zarizeni;
```

DML — Data Manipulation

Manipulace s daty

```
-- Vložení záznamu
INSERT INTO zarizeni
  (nazev, ip, aktivni)
VALUES
  ('Senzor A', '10.0.0.1', 1);

-- Úprava záznamu
UPDATE zarizeni
SET ip = '10.0.0.2'
WHERE id = 1;

-- Smazání záznamu
DELETE FROM zarizeni
WHERE aktivni = 0;
```

DQL — Data Query

Dotazování

```
-- Výběr dat
SELECT nazev, ip
FROM zarizeni
WHERE aktivni = 1
ORDER BY nazev;

-- Agregace
SELECT COUNT(*), AVG(hodnota)
FROM mereni
WHERE zarizeni_id = 1;
```

DCL — Data Control

Přístupová práva

```
GRANT SELECT ON mereni
  TO uzivatel;
REVOKE INSERT ON mereni
  FROM uzivatel;
```

SELECT — dotazování dat

`SELECT` je nejpoužívanější příkaz SQL — vybírá, filtruje, třídí a agreguje data.

Základní syntax a klauzule

```
SELECT sloupce
FROM   tabulka
WHERE  podminka
GROUP BY sloupce
HAVING   podminka_agregace
ORDER BY sloupce [ASC|DESC]
LIMIT  pocet;
```

Příklady

```
-- Posledních 10 měření senzoru 3
SELECT timestamp, hodnota, jednotka
FROM   mereni
WHERE  zarizeni_id = 3
ORDER BY timestamp DESC
LIMIT 10;

-- Průměr na zařízení za poslední den
SELECT zarizeni_id,
       AVG(hodnota) AS prumer,
       MAX(hodnota) AS maximum
FROM   mereni
WHERE  timestamp >= datetime('now','-1 day')
GROUP BY zarizeni_id;
```

JOIN — spojení tabulek

```
-- Vrátí jméno zařízení i naměřenou hodnotu
SELECT z.nazev,
       m.timestamp,
       m.hodnota,
       m.jednotka
FROM   mereni   AS m
JOIN   zarizeni AS z
  ON   m.zarizeni_id = z.id
WHERE  z.aktivni = 1
ORDER BY m.timestamp DESC;
```

Typy JOIN

Typ	Výsledek
`INNER JOIN`	jen řádky se shodou v obou
`LEFT JOIN`	všechny z levé + shody z pravé
`RIGHT JOIN`	všechny z pravé + shody z levé
`FULL JOIN`	všechny z obou

SQLite - datové typy a omezení

SQLite používá **dynamické typování** — hodnota nese typ, ne sloupec. Přesto existují afinní typy.

Afinní typy SQLite

Afinita	Uložení	Příklady
`INTEGER`	1–8 bajtů	id, počty, booleany
`REAL`	8B float	teplota, napětí
`TEXT`	UTF-8/16	jména, timestamp
`BLOB`	raw bytes	obrázky, binární data
`NUMERIC`	INT nebo REAL	datum, částky

Omezení (constraints)

```sql
CREATE TABLE senzor (
  id       INTEGER PRIMARY KEY AUTOINCREMENT,
  nazev    TEXT    NOT NULL UNIQUE,
  minimum  REAL    DEFAULT 0.0,
  maximum  REAL    CHECK (maximum > minimum),
  typ_id   INTEGER REFERENCES typy(id)
              ON DELETE SET NULL
);
```

Datum a čas v SQLite

SQLite nemá dedikovaný typ `DATETIME` — ukládá se jako TEXT nebo REAL.

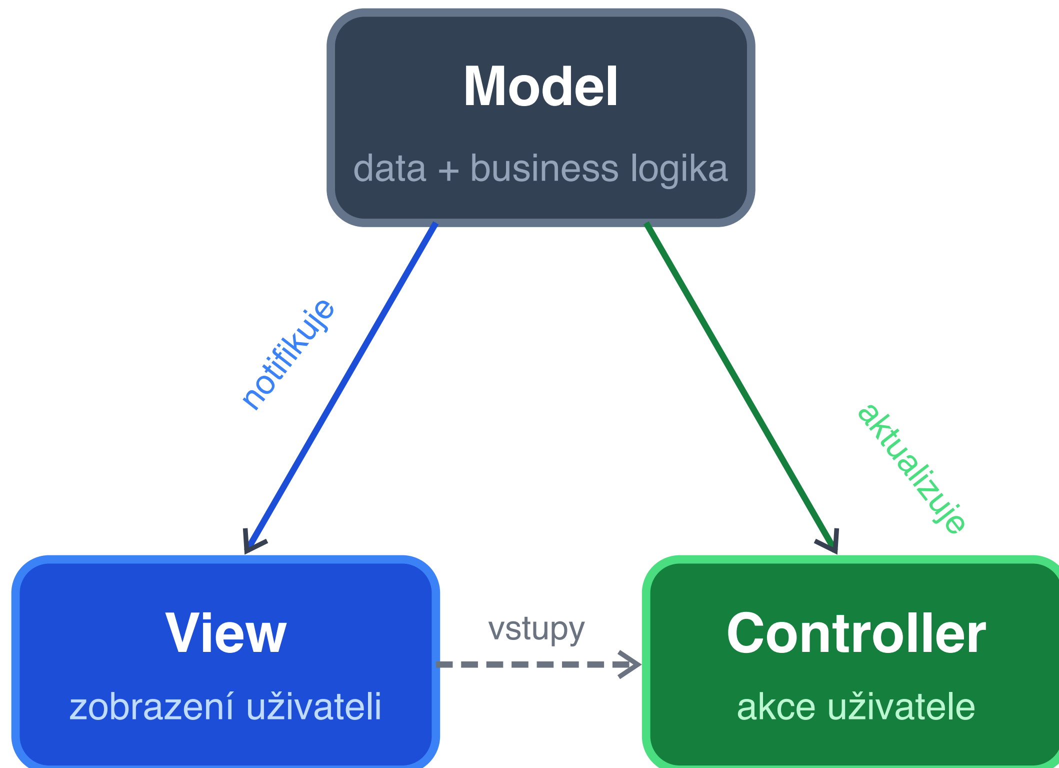
```sql
-- Uložení aktuálního času
INSERT INTO log (cas, zprava)
VALUES (datetime('now'), 'start');

-- Porovnání časů (ISO 8601 umožňuje
--   lexikografické řazení)
SELECT * FROM log
WHERE cas > '2025-01-01 00:00:00';

-- Časové výrazy
SELECT datetime('now', '-1 hour');
SELECT datetime('now', '+7 days');
SELECT strftime('%Y-%m', timestamp)
  AS mesic, COUNT(*)
FROM mereni
GROUP BY mesic;
```

> Pro jiné RDBMS (PostgreSQL, MySQL) existují dedikované typy `TIMESTAMP`, `DATE`, `INTERVAL` s bohatší podporou časových zón.

MVC — Model-View-Controller

MVC je návrhový vzor pro oddělení dat, jejich zobrazení a logiky ovládání.

Tři role



Model

- Drží data a business logiku
- Nezná View ani Controller
- Notifikuje pozorovatele o změnách
- *Příklady:* databáze, soubor, síťová data

View

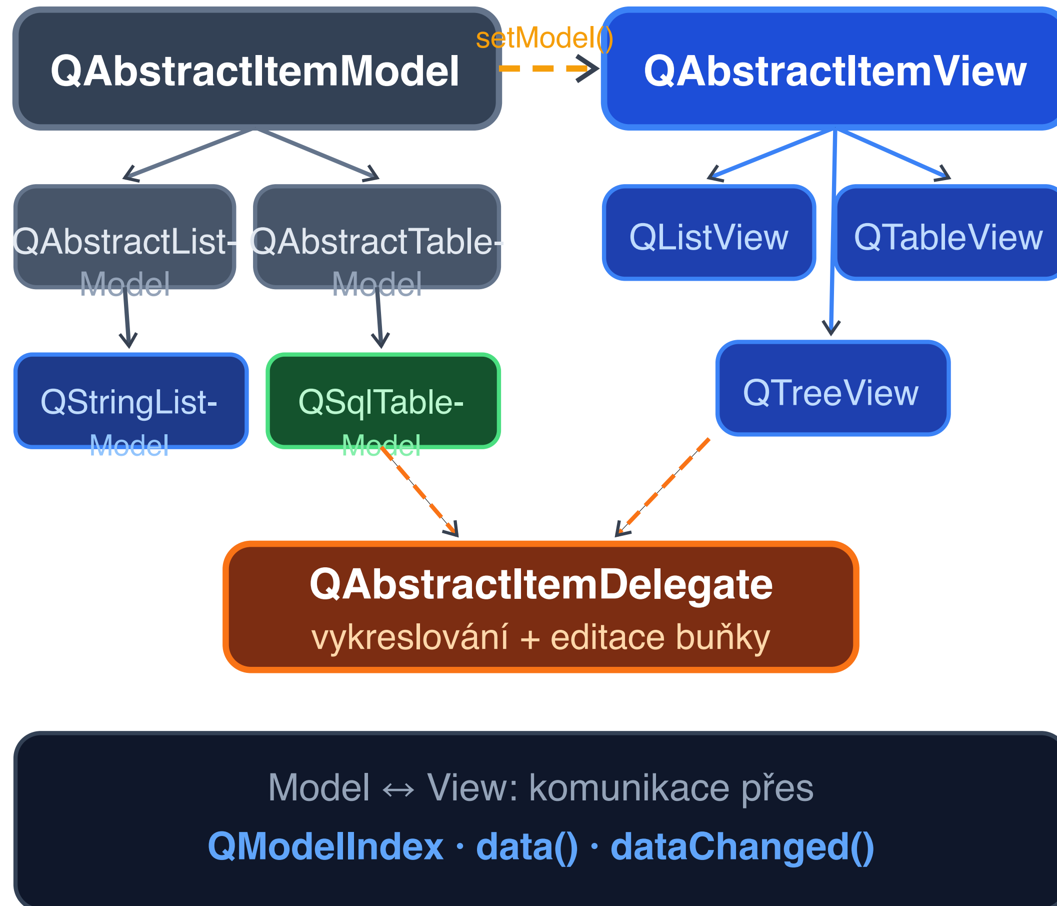
- Zobrazuje data z Modelu
- Nezná strukturu dat — jen volá Model
- Reaguje na notifikace Modelu
- *Příklady:* tabulka, graf, seznam

Controller

- Zpracovává vstupy uživatele
- Volá operace nad Modelem
- Vybírá správné View
- *Příklady:* handlery tlačítek, menu

Qt Model/View architektura

Qt neimplementuje čisté MVC — používá **Model/View/Delegate** (Controller je součástí View).



Model — co implementuje

- `data(index, role)` — vrátí hodnotu
- `rowCount()`, `columnCount()`
- `setData()` — zápis při editaci
- `headerData()` — popis sloupců/řádků
- signál `dataChanged` — notifikace View

View — co dělá

- Volá `model->data()` pro každou buňku
- Registruje se na `dataChanged`
- Při editaci volá `model->setData()`
- Nezávislý na zdroji dat

Delegate — co dělá

- `paint()` — vlastní vykreslení buňky
- `createEditor()` — widget pro editaci
- Umožňuje `ComboBox`, `SpinBox` v buňce

QAbstractItemModel — vlastní model

Implementace vlastního modelu: stačí přepsat 5 metod.

Minimální read-only model

```
class SensorModel
    : public QAbstractTableModel
{
    Q_OBJECT
    QList<SensorData> m_data;

public:
    int rowCount(
        const QModelIndex& = {}) const override
    { return m_data.size(); }

    int columnCount(
        const QModelIndex& = {}) const override
    { return 3; }  // čas, hodnota, jednotka

    QVariant data(
        const QModelIndex& idx,
        int role = Qt::DisplayRole)
            const override
    {
        if (!idx.isValid() ||
            role != Qt::DisplayRole)
            return {};
        const auto& r = m_data[idx.row()];
        switch (idx.column()) {
        case 0: return r.timestamp;
        case 1: return r.value;
        case 2: return r.unit;
        }
        return {};
```

Hlavičky a aktualizace dat

```
    QVariant headerData(
        int section,
        Qt::Orientation orient,
        int role = Qt::DisplayRole)
            const override
    {
        if (role != Qt::DisplayRole)
            return {};
        if (orient == Qt::Horizontal) {
            switch (section) {
            case 0: return "Čas";
            case 1: return "Hodnota";
            case 2: return "Jednotka";
            }
        }
        return section + 1; // čísla řádků
    }

    // Přidání nového řádku
    void addRow(const SensorData& d) {
        beginInsertRows(
            {}, m_data.size(),
            m_data.size());
        m_data.append(d);
        endInsertRows();
    }
};
```

II. Qt SQL modul

QSqlDatabase · QSqlQuery · QSqlTableModel

QSqlDatabase — připojení k databázi

`QSqlDatabase` reprezentuje jedno databázové připojení. Qt podporuje SQLite, PostgreSQL, MySQL a ODBC.

SQLite — embedded, bez serveru

```
// .pro soubor:  QT += sql
#include <QSqlDatabase>
#include <QSqlError>

// Otevření (nebo vytvoření) SQLite souboru
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName("data.db");
// Pro in-memory DB:
// db.setDatabaseName(":memory:");

if (!db.open()) {
    qWarning() << "DB chyba:" << db.lastError().text();
    return;
}
qDebug() << "DB připojena.";
```

Pojmenovaná připojení (více DB)

```
QSqlDatabase db2 =
    QSqlDatabase::addDatabase(
        "QSQLITE", "backup");
db2.setDatabaseName("backup.db");
db2.open();
// Přístup kdekoli v kódu:
QSqlDatabase::database("backup");
```

PostgreSQL a MySQL

```
// PostgreSQL
QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
db.setHostName("db.example.com");
db.setPort(5432);
db.setDatabaseName("iot_db");
db.setUserName("appuser");
db.setPassword("tajne_heslo");

// MySQL / MariaDB
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setDatabaseName("vyuka");
db.setUserName("root");
db.setPassword("password");
db.open();
```

Dostupné drivery

- `QSQLITE` SQLite 3
- `QPSQL` PostgreSQL
- `QMYSQL` MySQL / MariaDB
- `QODBC` ODBC (Windows)
- `QOCI` Oracle

QSqlQuery — přímé dotazy

`QSqlQuery` umožňuje spouštět libovolné SQL příkazy a iterovat přes výsledky.

CREATE TABLE a INSERT

```
#include <QSqlQuery>

QSqlQuery q;

// Vytvoření tabulky
q.exec(R"(
    CREATE TABLE IF NOT EXISTS zarizeni (
        id      INTEGER PRIMARY KEY,
        nazev   TEXT NOT NULL,
        ip      TEXT,
        aktivni INTEGER DEFAULT 1
    )
)");

// Vložení záznamu
q.exec("INSERT INTO zarizeni "
       "(nazev, ip) "
       "VALUES ('Senzor A', '10.0.0.1')");

qDebug() << "Vloženo, id:" << q.lastInsertId().toInt();
```

Kontrola chyb

```
if (!q.exec("SELECT ...")) {
    qWarning() << "SQL chyba:" << q.lastError().text();
}
```

SELECT a iterace výsledků

```
QSqlQuery q;
q.exec("SELECT id, nazev, ip "
       "FROM zarizeni "
       "WHERE aktivni = 1 "
       "ORDER BY nazev");

while (q.next()) {
    int     id    = q.value(0).toInt();
    QString nazev = q.value(1).toString();
    QString ip    = q.value(2).toString();

    qDebug() << id << nazev << ip;
}

// Přístup přes jméno sloupce:
q.value("nazev").toString();

// Počet řádků výsledku (nemusí být přesný)
q.size();
```

Navigace v sadě výsledků

```
q.first();    // skočí na první řádek
q.last();     // skočí na poslední
q.next();     // posune o jeden dopředu
q.previous(); // posune o jeden dozadu
q.seek(5);    // skočí na řádek 5
```

Prepared Statements — bezpečné dotazy

Parametrizované dotazy zabraňují **SQL injection** a jsou efektivnější při opakovaném volání.

Nebezpečné — SQL injection

```
// NIKDY takto! Uživatelský vstup přímo
// v SQL umožňuje injection útok.
QString name = lineEdit->text();
q.exec("SELECT * FROM users "
      "WHERE name = '" + name + "'");

// Útočník zadá: ' OR '1'='1
// → vrátí VŠECHNY záznamy!
```

Hrozba SQL injection

Vstup `'; DROP TABLE users; --` může smazat celou tabulku. Nikdy nevkládejte uživatelský vstup přímo do SQL řetězce.

Bezpečné — prepared statement

```
QSqlQuery q;

// Příprava dotazu s parametry
q.prepare("INSERT INTO mereni "
          "(zarizeni_id, timestamp, hodnota) "
          "VALUES (:zid, :ts, :val)");

// Bindování hodnot
q.bindValue(":zid", deviceId);
q.bindValue(":ts", QDateTime::currentDateTime().toString(Qt::ISODate));
q.bindValue(":val", measurement);

if (!q.exec()) {
    qWarning() << q.lastError().text();
}

// Opakované vložení (jen nové bindValue)
q.bindValue(":val", 24.1);
q.exec();
```

Výhody prepared statements

- Imunní vůči SQL injection
- DB optimalizuje plán jen jednou, pak jen dosazuje hodnoty → rychlejší v cyklu

Transakce

Transakce zaručuje, že buď proběhnou **všechny** operace, nebo žádná (atomičnost — ACID).

Příklad — hromadný import

```
QSqlDatabase db = QSqlDatabase::database();

db.transaction();   // zahájení transakce

QSqlQuery q;
q.prepare("INSERT INTO mereni "
          "(zarizeni_id, timestamp, hodnota)"
          " VALUES (?, ?, ?)");

bool ok = true;
for (const auto& m : measurements) {
    q.addBindValue(m.deviceId);
    q.addBindValue(m.timestamp);
    q.addBindValue(m.value);
    if (!q.exec()) {
        qWarning() << q.lastError();
        ok = false;
        break;
    }
}

if (ok) {
    db.commit();   // potvrzení
    qDebug() << "Import hotov.";
} else {
    db.rollback(); // zrušení všeho
    qWarning() << "Import selhal, rollback.";
}
```

ACID vlastnosti

A — Atomicity

Vše nebo nic. Částečné změny nejsou možné.

C — Consistency

DB přechází mezi konzistentními stavy.

I — Isolation

Souběžné transakce se navzájem neovlivňují.

D — Durability

Po commitu jsou data trvale uložena.

QSqlTableModel — model pro tabulku

`QSqlTableModel` spojuje SQL tabulku přímo s Qt model/view architekturou — žádné ruční načítání.

Nastavení modelu

```
#include <QSqlTableModel>
#include <QTableView>

// Model napojený na tabulku "zarizeni"
QSqlTableModel* model =
    new QSqlTableModel(this);
model->setTable("zarizeni");
model->setFilter("aktivni = 1");
model->setSort(1, Qt::AscendingOrder);
model->select();  // načtení dat z DB

// Popisky sloupců
model->setHeaderData(0, Qt::Horizontal,
    "ID");
model->setHeaderData(1, Qt::Horizontal,
    "Název");
model->setHeaderData(2, Qt::Horizontal,
    "IP adresa");

// Editační strategie
model->setEditStrategy(
    QSqlTableModel::OnManualSubmit);
```

Propojení s QTableView

```
QTableView* view = new QTableView(this);
view->setModel(model);
view->hideColumn(0); // skrýt ID
view->resizeColumnsToContents();
view->setSelectionBehavior(QAbstractItemView::SelectRows);
```

Uložení / zrušení změn

```
// Uložit všechny nevyřízené změny do DB
if (!model->submitAll()) {
    qWarning() << model->lastError();
}
// Zahodit neuložené změny
model->revertAll();
```

Přidání a smazání řádku

```
// Nový prázdný řádek
model->insertRow(model->rowCount());

// Smazání vybraného řádku
int row = view->currentIndex().row();
model->removeRow(row);
model->submitAll();
```

QSqlRelationalTableModel — vztahy

`QSqlRelationalTableModel` automaticky nahradí cizí klíče čitelnou hodnotou z jiné tabulky.

Model s relací

```
#include <QSqlRelationalTableModel>
#include <QSqlRelationalDelegate>

QSqlRelationalTableModel* model =
    new QSqlRelationalTableModel(this);
model->setTable("mereni");
model->setSort(2, Qt::DescendingOrder);

// Sloupec 1 (zarizeni_id) →
//   zobrazit "nazev" z tabulky "zarizeni"
model->setRelation(1,
    QSqlRelation(
        "zarizeni",   // cizí tabulka
        "id",         // klíč
        "nazev"));    // zobrazený sloupec

model->setHeaderData(0, Qt::Horizontal, "ID");
model->setHeaderData(1, Qt::Horizontal, "Zařízení");
model->setHeaderData(2, Qt::Horizontal, "Čas");
model->setHeaderData(3, Qt::Horizontal, "Hodnota");
model->select();
```

View s delegátem

```
QTableView* view = new QTableView;
view->setModel(model);

// Delegát zobrazí ComboBox pro cizí klíče
view->setItemDelegate(
    new QSqlRelationalDelegate(view));

view->hideColumn(0);
view->resizeColumnsToContents();
```

QSqlQueryModel — read-only dotaz

```
// Pro komplexní SELECT (JOIN, agregace)
// kde QSqlTableModel nestačí
#include <QSqlQueryModel>

QSqlQueryModel* qm = new QSqlQueryModel;
qm->setQuery(
    "SELECT z.nazev, "
    "       AVG(m.hodnota) AS prumer "
    "FROM mereni m "
    "JOIN zarizeni z ON m.zarizeni_id=z.id "
    "GROUP BY z.id");

view->setModel(qm);
```

Kompletní příklad — SQLite kontaktník

Inicializace DB a tabulky

```
class ContactDB : public QObject {
    Q_OBJECT
    QSqlDatabase m_db;
    QSqlTableModel* m_model = nullptr;

public:
    explicit ContactDB(QObject* p = nullptr)
        : QObject(p)
    {
        m_db = QSqlDatabase::addDatabase(
            "QSQLITE");
        m_db.setDatabaseName("contacts.db");
        if (!m_db.open()) {
            qFatal("DB: %s",
                qPrintable(
                    m_db.lastError().text()));
        }
        createSchema();
        m_model = new QSqlTableModel(
            this, m_db);
        m_model->setTable("contacts");
        m_model->setSort(1,
            Qt::AscendingOrder);
        m_model->select();
    }

    QSqlTableModel* model() {
        return m_model;
    }
```

Schema a operace

```
private:
    void createSchema() {
        QSqlQuery q(m_db);
        q.exec(R"(
            CREATE TABLE IF NOT EXISTS
            contacts (
                id    INTEGER PRIMARY KEY,
                name  TEXT NOT NULL,
                phone TEXT,
                email TEXT
            )
        )");
    }

public:
    void add(const QString& name,
             const QString& phone,
             const QString& email)
    {
        QSqlQuery q(m_db);
        q.prepare(
            "INSERT INTO contacts "
            "(name, phone, email) "
            "VALUES (?, ?, ?)");
        q.addBindValue(name);
        q.addBindValue(phone);
        q.addBindValue(email);
        q.exec();
        m_model->select(); // obnovit view
    }
};
```

III. NoSQL a alternativní úložiště

QSettings · JSON · Redis · Embedded DB

Typy databází — přehled

Různé datové modely jsou vhodné pro různé úlohy. Výběr závisí na struktuře dat a způsobu přístupu.

Relační (SQL)

- Pevné schéma
- Silné konzistence (ACID)
- JOIN, agregace
- SQLite, PostgreSQL

Klíč–hodnota

- Extrémně rychlé
- Jednoduché schéma
- Redis, LMDB

Dokumentové

- Flexibilní JSON/BSON
- Vnořené dokumenty
- MongoDB, CouchDB

Wide-column

- Velká data, analýzy
- Cassandra, HBase

Grafové

- Entity a vztahy
- Neo4j, ArangoDB

Kdy SQL?

- Strukturovaná data
- Vztahy mezi tabulkami
- Složité dotazy, JOIN

Kdy NoSQL?

- Schéma se mění
- Horizontální škálování
- Jednoduché dotazy na klíč
- Velký objem dat

> SQLite pokryje ~80 % embedded use-case. NoSQL zvažte při škálování nad jeden server.

QSettings — klíč/hodnota

`QSettings` je nativní Qt úložiště pro konfiguraci aplikace — bez externích závislostí.

Zápis a čtení hodnot

```
#include <QSettings>

// Výchozí umístění (OS-specific):
// Linux:   ~/.config/Firma/App.ini
// Windows: registry nebo .ini
// macOS:   ~/Library/Preferences/...
QSettings cfg("FEL", "SenzorApp");

// Zápis
cfg.setValue("server/host",    "10.0.0.1");
cfg.setValue("server/port",    1883);
cfg.setValue("ui/dark_mode",   true);
cfg.setValue("ui/font_size",   14);
cfg.setValue("last_device",    "COM3");

// Čtení (s výchozí hodnotou)
QString host = cfg.value(
    "server/host", "localhost").toString();
int port = cfg.value(
    "server/port", 1883).toInt();
bool dark = cfg.value(
    "ui/dark_mode", false).toBool();
```

Skupiny a výpis klíčů

```
// Skupiny zjednodušují práci s prefixem
cfg.beginGroup("server");
    cfg.setValue("host", "10.0.0.1");
    cfg.setValue("port", 1883);
    cfg.setValue("tls",  true);
cfg.endGroup();

// Přečíst celou skupinu
cfg.beginGroup("server");
    QStringList keys = cfg.childKeys();
    for (const QString& k : keys)
        qDebug() << k << cfg.value(k);
cfg.endGroup();

// Smazání klíče / skupiny
cfg.remove("last_device");
cfg.remove("server");  // celá skupina
```

Explicitní INI soubor

```
QSettings ini("config.ini",
    QSettings::IniFormat);
ini.setValue("mqtt/host", "broker.local");
```

JSON soubory jako úložiště

Pro strukturovaná data bez nutnosti SQL: JSON soubory jsou čitelné a přenositelné.

Zápis JSON dat

```
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QFile>

// Vytvoření datové struktury
QJsonArray devices;
for (const auto& d : deviceList) {
    QJsonObject obj;
    obj["id"]     = d.id;
    obj["name"]   = d.name;
    obj["ip"]     = d.ip;
    obj["active"] = d.active;
    devices.append(obj);
}

QJsonObject root;
root["version"] = 1;
root["devices"] = devices;

// Zápis do souboru
QFile file("devices.json");
file.open(QIODevice::WriteOnly);
file.write(QJsonDocument(root)
    .toJson(QJsonDocument::Indented));
```

Čtení JSON dat

```
QFile file("devices.json");
if (!file.open(QIODevice::ReadOnly)) {
    qWarning() << "Nelze otevřít soubor";
    return;
}

QJsonDocument doc =
    QJsonDocument::fromJson(
        file.readAll());

if (doc.isNull() || !doc.isObject()) {
    qWarning() << "Neplatný JSON";
    return;
}

QJsonObject root = doc.object();
int version = root["version"].toInt();

QJsonArray arr = root["devices"].toArray();
for (const QJsonValue& v : arr) {
    QJsonObject d = v.toObject();
    qDebug() << d["name"].toString()
             << d["ip"].toString();
}
```

SQLite s JSON1 — dokumentová DB v SQL

SQLite obsahuje vestavěné JSON funkce — kombinace relačního a dokumentového přístupu.

Ukládání JSON do SQLite sloupce

```
CREATE TABLE events (
    id        INTEGER PRIMARY KEY,
    timestamp TEXT DEFAULT
                 (datetime('now')),
    source    TEXT,
    payload   TEXT  -- JSON dokument
);

-- Vložení události s JSON payloadem
INSERT INTO events (source, payload)
VALUES ('sensor_1',
    '{"temp":23.4,"hum":61,"unit":"C"}');

INSERT INTO events (source, payload)
VALUES ('gateway',
    '{"status":"ok","uptime":3600,
     "devices":["s1","s2"]}');
```

Dotazování JSON polí

```
-- Extrakce hodnoty z JSON
SELECT source,
       json_extract(payload, '$.temp') AS t,
       json_extract(payload, '$.hum')  AS h
FROM   events
WHERE  source = 'sensor_1'
  AND  json_extract(payload, '$.temp') > 22;

-- Filtr přes JSON pole
SELECT * FROM events
WHERE json_extract(payload,'$.status')
      = 'ok';

-- Sestavení JSON z relačních dat
SELECT json_object(
    'name', nazev,
    'ip',    ip_adresa
) FROM zarizeni WHERE aktivni = 1;
```

> Funkce `json_extract`, `json_object`, `json_array` jsou dostupné v SQLite ≥ 3.9 (2015) — obvykle bez potřeby dalšího nastavení.

Redis a embedded databáze

Pro specializované úlohy (cache, rychlý přístup) existují alternativy k SQLite.

Redis přes QTcpSocket

Redis komunikuje jednoduchým textovým protokolem (RESP) — lze ho ovládat přímo přes TCP.

```
// Připojení k Redis serveru
QTcpSocket* redis = new QTcpSocket(this);
redis->connectToHost("localhost", 6379);

// SET klíč hodnota
redis->write("SET sensor:1:temp 23.4\r\n");

// GET klíč
redis->write("GET sensor:1:temp\r\n");

// EXPIRE — platnost 60 sekund
redis->write(
    "EXPIRE sensor:1:temp 60\r\n");

// LPUSH — přidat do listu
redis->write(
    "LPUSH history:1 23.4\r\n");

// Pro produkci: knihovna hiredis
// nebo Qt Redis klienti (redisclient)
```

Embedded key-value databáze

Fungují jako knihovna (bez serveru) — podobně jako SQLite.

DB	Vlastnosti
LMDB	Memory-mapped, velmi rychlé čtení
RocksDB	Facebook, optimalizovaný zápis
LevelDB	Google, jednoduchý
SQLite WAL	SQLite s Write-Ahead Logging

LMDB v Qt projektu

```
// Integrace přes CMake/qmake
// lmdb je C knihovna — volání přes C API
MDB_env* env;
mdb_env_create(&env);
mdb_env_open(env, "./db", 0, 0664);

// Qt wrapper: lmdbxx (C++17)
// nebo vlastní QObject obal
```

> Pro embedded IoT zařízení (Raspberry Pi, iMX6): LMDB je extrémně rychlé a odolné vůči výpadku napájení díky MVCC.

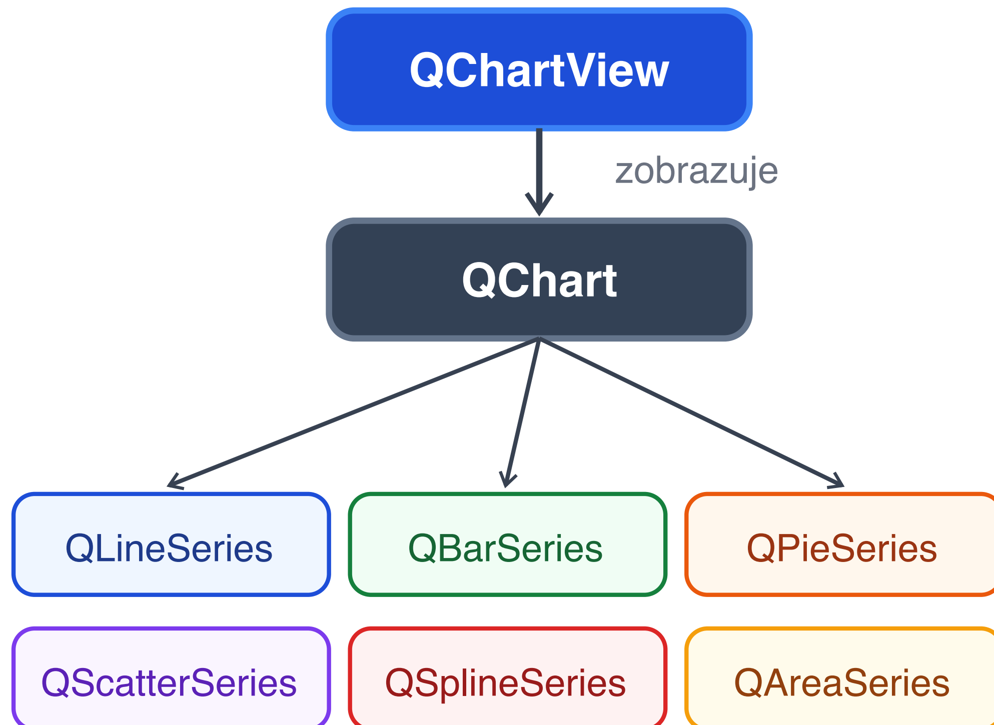
IV. Vizualizace dat — Qt Charts

QChart · QChartView · Real-time grafy

Qt Charts — přehled

Modul `Qt Charts` (dříve komerční, od Qt 5.7 LGPL) poskytuje interaktivní grafy přímo v Qt aplikaci.

Klíčové třídy



Minimální příklad

```
// .pro soubor:  QT += charts
#include <QtCharts>

// Série dat
QLineSeries* series = new QLineSeries();
series->setName("Teplota");
series->append(0,  22.1);
series->append(1,  22.8);
series->append(2,  23.4);
series->append(3,  23.1);

// Graf
QChart* chart = new QChart();
chart->addSeries(series);
chart->setTitle("Měření teploty");
chart->createDefaultAxes();
chart->legend()->setVisible(true);

// Widget pro zobrazení
QChartView* view = new QChartView(chart);
view->setRenderHint(
    QPainter::Antialiasing);
view->resize(600, 400);
view->show();
```

QLineSeries — spojnicový graf

Ideální pro časové řady: teplota, napětí, průtok — libovolný průběh veličiny v čase.

Nastavení os a vzhledu

```
QLineSeries* series = new QLineSeries();
series->setName("Teplota [°C]");

// Přidání bodů
for (int i = 0; i < data.size(); ++i)
    series->append(i, data[i]);

QChart* chart = new QChart();
chart->addSeries(series);
chart->setTitle("Průběh teploty");

// Vlastní osy
QValueAxis* axisX = new QValueAxis();
axisX->setTitleText("čas [s]");
axisX->setRange(0, data.size());
axisX->setTickCount(6);

QValueAxis* axisY = new QValueAxis();
axisY->setTitleText("Teplota [°C]");
axisY->setRange(0, 50);

chart->addAxis(axisX, Qt::AlignBottom);
chart->addAxis(axisY, Qt::AlignLeft);
series->attachAxis(axisX);
series->attachAxis(axisY);
```

Časová osa (QDateTimeAxis)

```
// Osa s reálnými časovými razítky
QLineSeries* series = new QLineSeries();

for (const auto& m : measurements) {
    qint64 ms = m.timestamp.toMSecsSinceEpoch();
    series->append(ms, m.value);
}

QDateTimeAxis* axisX = new QDateTimeAxis();
axisX->setFormat("hh:mm");
axisX->setTitleText("Čas");

QChart* chart = new QChart();
chart->addSeries(series);
chart->addAxis(axisX, Qt::AlignBottom);
series->attachAxis(axisX);
```

Styl série

```
// Barva, šířka čáry, tečky
QPen pen(Qt::red);
pen.setWidth(2);
series->setPen(pen);
series->setPointsVisible(true);
series->setPointLabelsVisible(true);
series->setPointLabelsFormat("@yPoint");
```

QBarSeries — sloupcový graf

Sloupcové grafy srovnávají hodnoty mezi kategoriemi nebo v čase.

Jednoduché sloupce (QBarSeries)

```
// Skupiny na ose X
QBarSet* set0 = new QBarSet("Leden");
QBarSet* set1 = new QBarSet("Únor");
QBarSet* set2 = new QBarSet("Březen");

*set0 << 22.1 << 21.8 << 22.5 << 23.0;
*set1 << 20.1 << 19.8 << 21.0 << 22.4;
*set2 << 24.3 << 25.1 << 24.8 << 23.9;

QBarSeries* series = new QBarSeries();
series->append(set0);
series->append(set1);
series->append(set2);

QChart* chart = new QChart();
chart->addSeries(series);
chart->setTitle("Průměrné teploty");

// Kategorická osa X
QBarCategoryAxis* axisX =
    new QBarCategoryAxis();
axisX->append({"S1","S2","S3","S4"});
chart->addAxis(axisX, Qt::AlignBottom);
series->attachAxis(axisX);
```

Varianty sloupcových grafů

```
// Skládaný (stacked) — součet sérií
QStackedBarSeries* stacked = new QStackedBarSeries();

// Procentuální → vždy 100 %
QPercentBarSeries* percent = new QPercentBarSeries();

// Horizontální
QHorizontalBarSeries* hbar = new QHorizontalBarSeries();
```

Popisky a tooltip

```
// Zobrazit hodnoty nad sloupci
series->setLabelsVisible(true);
series->setLabelsPosition(QAbstractBarSeries::LabelsOutsideEnd);
series->setLabelsFormat("@value °C");

// Interaktivní tooltip
connect(series,
    &QBarSeries::hovered,
    [](bool status,
       int idx, QBarSet* barSet) {
        if (status)
            qDebug() << barSet->label()
                     << barSet->at(idx);
    });
```

QPieSeries — koláčový graf

Koláčové grafy zobrazují podíly celku — distribuce chyb, zastoupení kategorií.

Základní koláčový graf

```
QPieSeries* series = new QPieSeries();

series->append("WiFi",    35.2);
series->append("Ethernet",28.1);
series->append("RS-485",  21.7);
series->append("CAN",     15.0);

// Výřez prvního dílu
QPieSlice* slice = series->slices().at(0);
slice->setExploded(true);   // vysunutý
slice->setLabelVisible(true);
slice->setPen(QPen(Qt::white, 2));

// Popisky všech dílů
for (auto* s : series->slices()) {
    s->setLabel(
        QString("%1: %2 %")
        .arg(s->label())
        .arg(s->percentage()*100, 0,'f',1));
    s->setLabelVisible(true);
}

QChart* chart = new QChart();
chart->addSeries(series);
chart->setTitle("Typy připojení");
chart->legend()->setAlignment(Qt::AlignRight);
```

QScatterSeries — bodový graf

```
// Bodový graf — distribuce nebo korelace
QScatterSeries* series = new QScatterSeries();
series->setName("Měření");
series->setMarkerShape(QScatterSeries::MarkerShapeCircle);
series->setMarkerSize(8.0);
series->setColor(QColor("#1d4ed8"));
series->setBorderColor(Qt::white);

for (const auto& pt : points)
    series->append(pt.x, pt.y);
```

QSplineSeries — vyhlazená křivka

```
// Jako QLineSeries, ale interpoluje
// kubickými spliny
QSplineSeries* spline = new QSplineSeries();
spline->setName("Vyhlazeno");
for (int i = 0; i < raw.size(); ++i)
    spline->append(i, raw[i]);
```

> Lze kombinovat více sérií různého typu v jednom `QChart` — např. `QLineSeries` + `QScatterSeries` pro naměřené hodnoty s vyznačenými outliers.

Real-time aktualizace grafu

Grafy lze aktualizovat živě — příchozí data ze sériového portu, MQTT nebo senzorů.

Posuvné okno (rolling buffer)

```
class RealtimeChart : public QWidget {
    Q_OBJECT
    QLineSeries* m_series;
    QChart*      m_chart;
    QValueAxis*  m_axisX;
    QValueAxis*  m_axisY;
    int          m_maxPoints = 60;
    int          m_tick      = 0;

public:
    explicit RealtimeChart(QWidget* p=nullptr) : QWidget(p)
    {
        m_series = new QLineSeries();
        m_chart  = new QChart();
        m_chart->addSeries(m_series);
        m_chart->setTitle("Live data");
        m_chart->legend()->hide();
        m_chart->setAnimationOptions(QChart::NoAnimation);

        m_axisX = new QValueAxis();
        m_axisX->setRange(0, m_maxPoints);
        m_axisY = new QValueAxis();
        m_axisY->setRange(0, 50);

        m_chart->addAxis(m_axisX, Qt::AlignBottom);
        m_chart->addAxis(m_axisY, Qt::AlignLeft);
        m_series->attachAxis(m_axisX);
        m_series->attachAxis(m_axisY);
    }
```

Přidání nového vzorku

```
public slots:
    void addSample(double value) {
        m_series->append(m_tick, value);
        m_tick++;

        // Posuvné okno — mazání starých bodů
        if (m_series->count() > m_maxPoints) {
            m_series->remove(0);
            m_axisX->setRange(m_tick - m_maxPoints, m_tick);
        }
    }
};

// Připojení na časovač nebo signál
QTimer* timer = new QTimer(this);
connect(timer, &QTimer::timeout, [this]() {
    double v = readSensor();
    chart->addSample(v);
});
timer->start(1000); // každou sekundu

// Nebo přímo na QSerialPort / MQTT
connect(serial, &QSerialPort::readyRead,
    [this, serial]() {
        double v = parseValue(
            serial->readLine());
        m_chart->addSample(v);
    });
```

Customizace grafů

Qt Charts umožňuje podrobnou úpravu vzhledu — barvy, legendy, osy, pozadí.

Téma a pozadí

```
// Předdefinovaná témata
chart->setTheme(QChart::ChartThemeDark);
chart->setTheme(QChart::ChartThemeLight);
chart->setTheme(
    QChart::ChartThemeBlueCerulean);

// Vlastní pozadí grafu
chart->setBackgroundBrush(
    QBrush(QColor("#1e293b")));
chart->setBackgroundPen(Qt::NoPen);
chart->setPlotAreaBackgroundBrush(
    QBrush(QColor("#0f172a")));
chart->setPlotAreaBackgroundVisible(true);

// Titulek
chart->setTitle("Teplotní log");
chart->setTitleFont(
    QFont("Fira Sans", 14, QFont::Bold));
chart->setTitleBrush(
    QBrush(Qt::white));
```

Legenda a osy

```
// Legenda
QLegend* legend = chart->legend();
legend->setVisible(true);
legend->setAlignment(Qt::AlignBottom);
legend->setFont(QFont("Fira Sans", 9));

// Mřížka osy
QValueAxis* axisY = new QValueAxis();
axisY->setGridLineVisible(true);
axisY->setGridLinePen(
    QPen(QColor("#334155"), 1, Qt::DotLine));
axisY->setMinorGridLineVisible(true);
axisY->setLabelFormat("%.1f °C");
axisY->setTickCount(6);

// Interaktivita
view->setRubberBand(
    QChartView::RectangleRubberBand);
// Zoom myší — přetažení výběru
// Vrácení na výchozí pohled:
chart->zoomReset();
```

Graf z databáze — kompletní příklad

Načtení dat z SQLite a jejich zobrazení v Qt Charts grafu.

Načtení dat z DB

```
void loadChartFromDB(QChart* chart, int deviceId)
{
    // Smazání starých sérií
    chart->removeAllSeries();

    QLineSeries* series = new QLineSeries();
    series->setName(QString("Zařízení %1").arg(deviceId));

    QSqlQuery q;
    q.prepare(
        "SELECT timestamp, hodnota "
        "FROM mereni "
        "WHERE zarizeni_id = ? "
        "  AND timestamp >= "
        "      datetime('now','-1 hour') "
        "ORDER BY timestamp");
    q.addBindValue(deviceId);
    q.exec();

    while (q.next()) {
        QDateTime ts = QDateTime::fromString(
            q.value(0).toString(),
            Qt::ISODate);
        double val = q.value(1).toDouble();
        series->append(
            ts.toMSecsSinceEpoch(), val);
    }
    chart->addSeries(series);
```

Nastavení os a zobrazení

```
    // Časová osa X
    QDateTimeAxis* axisX =
        new QDateTimeAxis();
    axisX->setFormat("HH:mm");
    axisX->setTitleText("Čas");
    axisX->setTickCount(7);

    // Hodnotová osa Y
    QValueAxis* axisY = new QValueAxis();
    axisY->setTitleText("Hodnota");
    axisY->setLabelFormat("%.1f");

    chart->addAxis(axisX, Qt::AlignBottom);
    chart->addAxis(axisY, Qt::AlignLeft);
    series->attachAxis(axisX);
    series->attachAxis(axisY);
    chart->createDefaultAxes();
}

// Automatická obnova každou minutu
QTimer* refreshTimer = new QTimer(this);
connect(refreshTimer, &QTimer::timeout,
    [this]() {
        loadChartFromDB(m_chart,
            m_selectedDevice);
    });
refreshTimer->start(60'000);
```

Alternativy Qt Charts

QCustomPlot

- Opensource, MIT licence
- Jeden `.cpp` + `.h` soubor
- Velmi rychlé vykreslování
- Vhodné pro velká data

```
// Přidání do .pro:
// SOURCES += qcustomplot.cpp
// HEADERS += qcustomplot.h
#include "qcustomplot.h"

QCustomPlot* plot =
    new QCustomPlot(this);

plot->addGraph();
plot->graph(0)->setData(
    xData, yData);
plot->xAxis->setLabel("čas");
plot->yAxis->setLabel("°C");
plot->replot();
```

Matplotlib (Python)

Qt aplikace může spouštět Python skripty nebo volat matplotlib přes `QProcess`.

```
// Vygenerování grafu přes Python
QProcess proc;
proc.start("python3",
    {"plot_temp.py",
     "--device", "1",
     "--output", "chart.png"});
proc.waitForFinished(5000);

// Zobrazení PNG v Qt
QLabel* img = new QLabel;
img->setPixmap(
    QPixmap("chart.png"));
```

Výhody: bohatá knihovna, snadná publikace

Grafana (webový dashboard)

Grafana čte z InfluxDB / PostgreSQL a zobrazuje interaktivní dashboardy v prohlížeči.

Qt aplikace zapisuje data, Grafana vizualizuje — bez nutnosti řešit grafy v Qt.

```
# docker-compose.yml
services:
  grafana:
    image: grafana/grafana
    ports: ["3000:3000"]
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: iot
```

Kdy použít: sdílené dashboardy, mobilní přístup, alerty

Shrnutí — kdy použít co

Databáze a úložiště

Potřeba	Řešení
Nastavení aplikace	`QSettings`
Konfigurace v souboru	JSON soubor
Embedded relační data	SQLite + `QSqlite`
Real-time cache, session	Redis
Velká data, více serverů	PostgreSQL
Flexibilní dokumenty	MongoDB

Qt SQL — rychlý přehled

- `QSqlDatabase` — připojení
- `QSqlQuery` — přímé SQL, prepared stmt
- `QSqlTableModel` — model/view + edit
- `QSqlRelationalTableModel` — FK → combobox

Vizualizace

Typ grafu	Třída
Časová řada, průběh	`QLineSeries`
Vyhlazený průběh	`QSplineSeries`
Porovnání kategorií	`QBarSeries`
Podíly celku	`QPieSeries`
Distribuce, korelace	`QScatterSeries`
Velká data, rychlost	QCustomPlot
Webový dashboard	Grafana

Doporučení

- SQLite pro embedded a desktop aplikace
- PostgreSQL pro multi-user a produkci
- Qt Charts pro jednoduché grafy v GUI
- Grafana pro sdílené dashboardy a alerty

Zdroje a dokumentace

Qt dokumentace

- `QSqlDatabase` — `doc.qt.io/qt-6/qsqldatabase.html`
- `QSqlQuery` — `doc.qt.io/qt-6/qsqlquery.html`
- `QSqlTableModel` — `doc.qt.io/qt-6/qsqltablemodel.html`
- `Qt Charts` — `doc.qt.io/qt-6/qtcharts-index.html`
- `QSettings` — `doc.qt.io/qt-6/qsettings.html`

Databáze

- SQLite — `sqlite.org` · SQLite JSON1 — `sqlite.org/json1.html`
- PostgreSQL — `postgresql.org`
- Redis — `redis.io`

Vizualizace

- QCustomPlot — `qcustomplot.com`
- Grafana — `grafana.com`
- Matplotlib — `matplotlib.org`