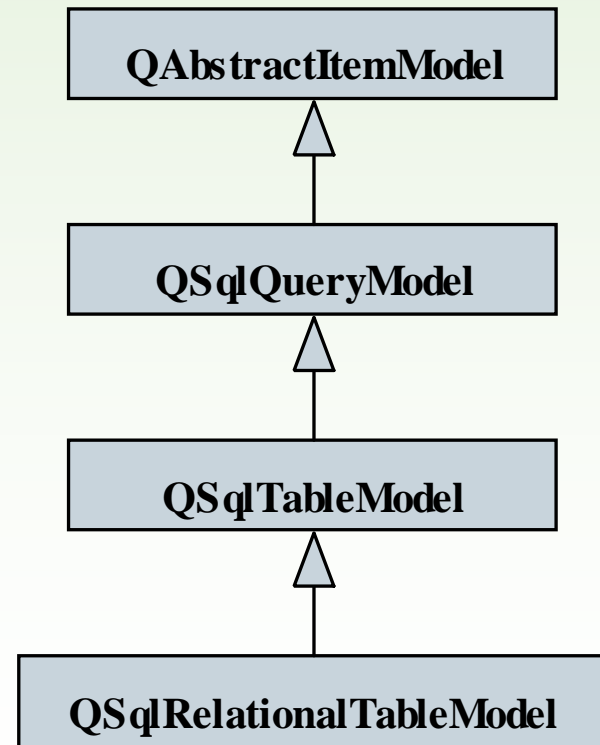


Adatbáziskezelés M/V architektúrában

Adatbázis-kezelő modellek

□ A modellek a **QAbstractItemModel** leszármazottai, ezek közül adatbázis-kezelésre 3 alkalmazható:

- **QSqlQueryModel**: egy lekérdezés eredményének kezelésére (csak olvasható)
- **QSqlTableModel**: egy tábla tartalmának kezelésére (írható és olvasható)
- **QSqlRelationalTableModel**: idegen kulcsokat tartalmazó tábla kezelésére (további táblákból begyűjtött adatokkal)



Lekérdezés modell

- ❑ A **QSqlQueryModel** típust lekérdezések megjelenítésére, olvasásra használhatjuk.
 - **setQuery(<lekérdezés>)** metódussal beállíthatunk tetszőleges lekérdezést (akár több táblát is felhasználva).
 - **setHeaderData(<oszlop>, <megjelenés>, <elnevezés>)** művelettel szabályozhatjuk az oszlopok fejlécének tulajdonságait.
 - A sorok számát a **rowCount()**, az oszlopok számát a **columnCount()** metódussal kérdezhetjük le.
 - Az adatokat soronként (**record(<sor>)**), vagy indexek segítségével (**index(<sor>, <oszlop>)**) érhetjük el.

Lekérdezés modell nézete

- ❑ Modellek megjelenítéséhez a **QAbstractItemView** leszármazottait kell használnunk, ezek közül a táblázatos megjelenítéshez a **QTableView** típust
 - a **setModel(<modell>)** művelettel állítjuk be a modellt

```
QSqlQueryModel model; // modell
model.setQuery("select ..."); // lekérdezés
model.setHeaderData(0, Qt::Horizontal, "Id");
    // oszlop fejlécének beállítása
...
QTableView view; // nézet
view.setModel(model); // modell beállítása
view.show(); // megjelenítés
```

Tábla modell

- ❑ Az adatbázisok kezelésénél a leggyakrabban használt modell a tábla.
- ❑ Egy tábla lekérdezését és szerkesztését a **QSqlTableModel** osztály biztosítja.
 - A **setTable(<táblanév>)** művelettel állíthatunk be egy táblát adatforrásnak, a **select()** művelet szolgál az adatok lekérdezésére.
 - Adatot lekérdezni a **data(<index>)**, beállítani a **setData(<index>, <adat>)** metódussal tudunk.
 - Lehetőségünk van tetszőlegesen rendezni az adatokat a **setSort(<oszlop>, <rendezési mód>)** művelettel.
 - Az **insertRow(<sor>)** beszúr egy üres sort a megadott helyre, a **removeRow(<sor>)** töröl egy sort.

Példa

```
QSqlTableModel *model = new QSqlTableModel(this);  
model->setTable("myTable");  
model->select();  
...  
model->insertRow(row);  
...  
QModelIndex index = model->index(row, 0);  
model->setData(index, 100);
```

modell létrehozása és
adatok begyűjtése

új sor beszúrása a modell
adott sorszámú soraként

a beszúrt sor elsőoszlopa
adatelemének indexe

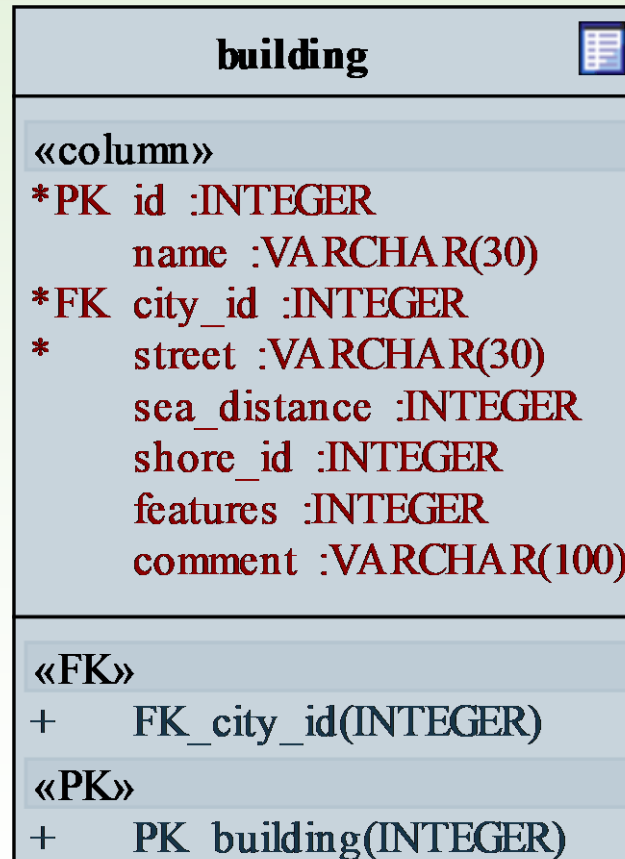
kiválasztott adatelem
értékének felülírása

1.Feladat

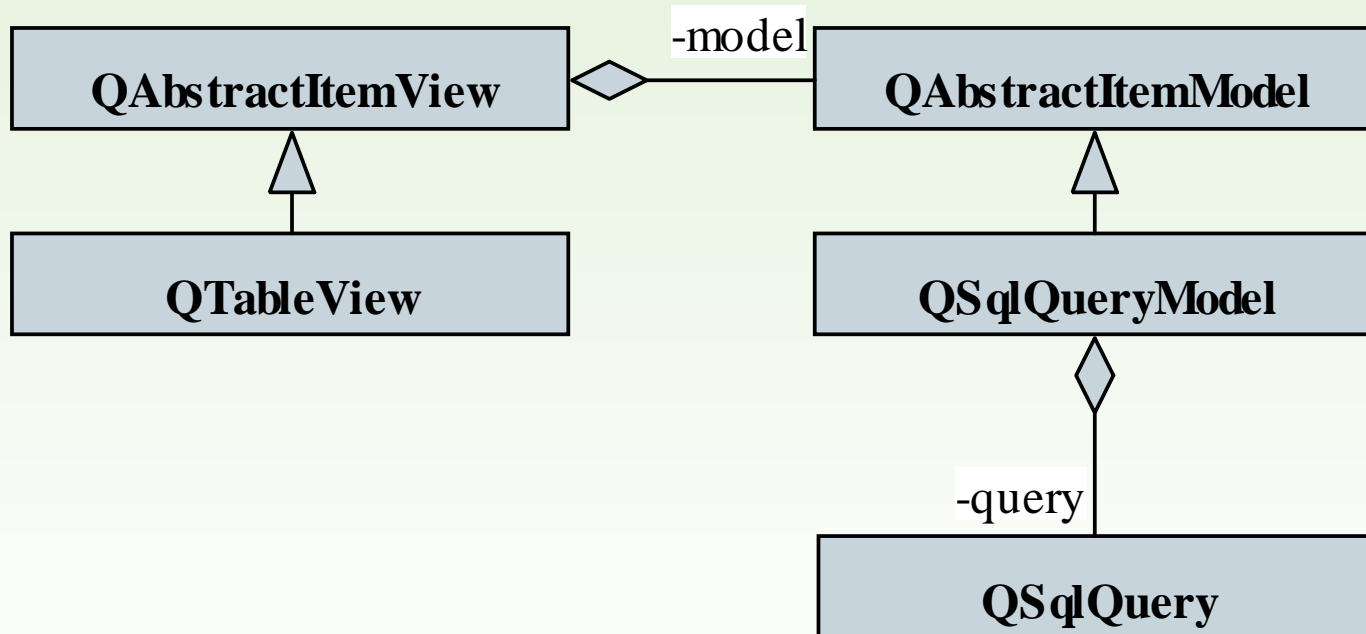
Készítsük el az apartman adatbázis épületeinek (**buildings**) grafikus megjelenítését.

- Az alkalmazáshoz nem kell egyetlen új osztályt se definiálnunk, a létező típusok felhasználásával megoldható a feladat.
- Az ablakban egy táblamegjelenítőben jelenjen meg a tábla teljes tartalma, ehhez egy **QTableView** példányt alkalmazunk.
- Az adatok betöltését egy lekérdező modellel végezzük (**QSqlQueryModel**), amely megkapja a megfelelő lekérdezést, és lefuttatja a lekérdező műveleteket (**QSqlQuery**).

1.Feladat: tervezés



1.Feladat: tervezés



1.Feladat: megvalósítás

```
int main(int argc, char *argv[]) {  
    QApplication a(argc, argv);  
    QSqlDatabase db = QSqlDatabase::addDatabase("MYSQL");  
    ...  
    QSqlQueryModel* model = new QSqlQueryModel();  
    model->setQuery("select * from building");  
    model->setHeaderData(0, Qt::Horizontal, trUtf8("Azonosító"));  
    ...  
    QTableView* tableView = new QTableView();  
    tableView->setModel(model);  
    tableView->show();  
  
    return a.exec();  
}
```

kapcsolat létrehozása

lekérdezési modell beállítása

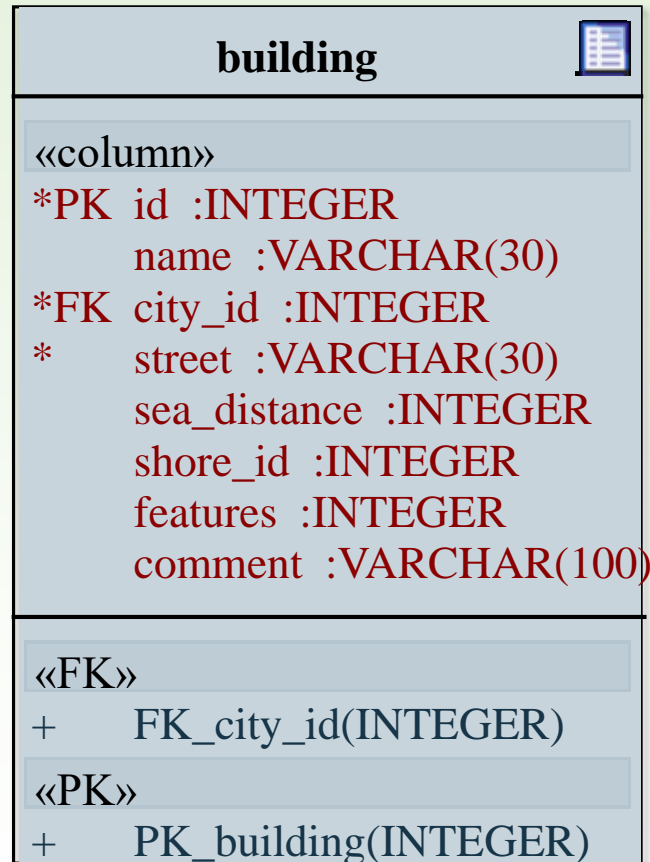
tábla megjelenítő beállítása

2.Feladat

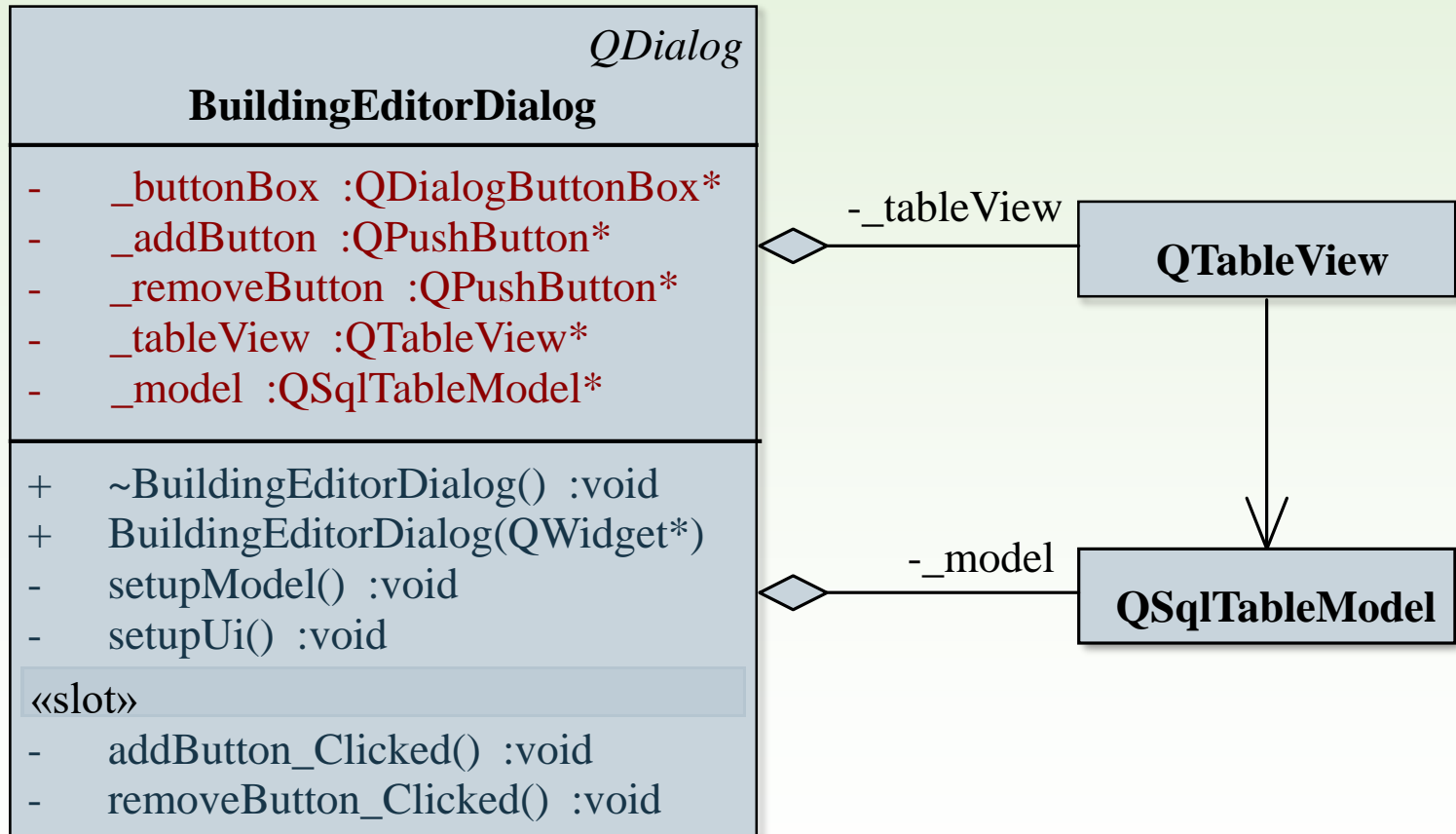
Készítsünk alkalmazást, amely alkalmas az épületek szerkesztésére, új épület létrehozására, törlésére.

- A táblaszerkesztést egy ablakba (**TableModelDialog**) helyezzük, amelyhez felvesszük a hozzáadás és törlés gombjait, a táblakezeléshez egy **QSqlTableModel**, a megjelenítéshez egy **QTableView** példányt használunk.
- Beszúráskor lekérdezzük a kijelölt sor indexét, behelyezünk egy sort a helyére, átállítjuk a kijelölést (az indexen keresztül), majd szerkesztésre váltunk.
- Törléskor töröljük a kijelölt sort (amennyiben van kijelölés), és áthelyezzük a kijelölést.

2.Feladat: tervezés



2.Feladat: tervezés



2.Feladat: megvalósítás

```
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    db.setHostName("localhost");      db.setDatabaseName("apartments");
    db.setUserName("root");
    db.setPassword("root");
    if (db.open()) { // kapcsolat megnyitása
        db.close(); // rögtön be is zárhatjuk, a később nem kell
        BuildingEditorDialog *w = new BuildingEditorDialog();
        w->show();
    }
    else {
        QMessageBox::critical(0, QObject::trUtf8("Hiba!"),
            QObject::trUtf8("adatbázis-szerver kapcsolat nincs"));
    }
    return a.exec();
}
```

2.Feladat: megvalósítás

```
void BuildingEditorDialog::setupModel() {
    _model = new QSqlTableModel(this); // táblamodell létrehozása
    _model->setTable("building"); // tábla beállítása
    _model->setSort(1, Qt::AscendingOrder); // rendezés oszlopra
    _model->setHeaderData(0, Qt::Horizontal, trUtf8("Azonosító"));
    // fejlécek beállítása
    _model->setHeaderData(1, Qt::Horizontal, trUtf8("Név"));
    _model->setHeaderData(2, Qt::Horizontal, trUtf8("Város"));
    _model->setHeaderData(3, Qt::Horizontal, trUtf8("Utca"));
    _model->setHeaderData(4, Qt::Horizontal, trUtf8("Tenger távolság"));
    _model->setHeaderData(5, Qt::Horizontal, trUtf8("Tengerpart"));
    _model->setHeaderData(6, Qt::Horizontal, trUtf8("Jellemzők"));
    _model->setHeaderData(7, Qt::Horizontal, trUtf8("Megjegyzés"));
    _model->select(); // adatok begyűjtése
}
```

2.Feladat: megvalósítás

```
void BuildingEditorDialog::setupUi() {
    _addButton = new QPushButton(trUtf8("Beszúrás"));
    _removeButton = new QPushButton(trUtf8("Törlés"));
    _buttonBox = new QDialogButtonBox(Qt::Horizontal);
    _buttonBox->addButton(_addButton, QDialogButtonBox::ActionRole);
    _buttonBox->addButton(_removeButton, QDialogButtonBox::ActionRole);

    connect(_addButton, SIGNAL(clicked()),
            this, SLOT(addButton_Clicked()));
    connect(_removeButton, SIGNAL(clicked()),
            this, SLOT(removeButton_Clicked()));

    _tableView = new QTableView(this);
    _tableView->setModel(_model);
    _tableView->setSelectionBehavior(QAbstractItemView::SelectItems);
    _tableView->resizeColumnsToContents();
    ...
}
```


2.Feladat: megvalósítás

```
void BuildingEditorDialog::removeButton_Clicked(){
    QModelIndex index = _tableView->currentIndex(); // kijelölés indexe
    if (index.isValid()) { // ha érvényes az index
        _model->removeRow(index.row()); // töröljük a kijelölt sort
        _tableView->setCurrentIndex(_model->index(index.row()-1, 0));
        // beállítjuk a táblakijelölést az előző sorra
    } else { // ha nincs érvényes kijelölés
        QMessageBox::warning(this, trUtf8("Nincs kijelölés!"),
            trUtf8("Kérem jelölje ki előbb a törlendő sort!"));
    }
}
```

2.Feladat: megvalósítás

```
void BuildingEditorDialog::addButton_Clicked() {
    int row; // beszúrandó sor sorának száma
    if (_tableView->currentIndex().isValid()) {
        row = _tableView->currentIndex().row();
        // az aktuális kijelölés sorának száma
    } else { // ha nincs érvényes kijelölés
        row = _model->rowCount(); // utolsó sor utáni sorszám
    }

    _model->insertRow(row);
    QModelIndex newIndex = _model->index(row, 0); // index az új sorra
    _tableView->setCurrentIndex(newIndex); // táblakijelölés az indexre
    _tableView->edit(newIndex); // szerkesztés alá helyezzük az elemet
}
```

Szinkron és aszinkron kapcsolat

- ❑ Az adatkezelés szerkesztési stratégiája kétféle lehet:
 - Az automatikus szerkesztési stratégia **állandó kapcsolatú ún. szinkron modellel** dolgozik, amikor az adatbázis és a modell tartalma folyamatosan (legalábbis rekord-váltásonként) megegyezik.
 - A manuális szerkesztési stratégia **bontott kapcsolatú ún. aszinkron modellel** dolgozik, amikor az adatbázis és a modell tartalma különbözhet, és csak meghatározott pontokon egyezik meg (**select**, **submitAll**, **revertAll**).
- ❑ A gyakorlatban az aszinkron modell az elterjedtebb, mivel nem igényli állandóan az adatbázis műveletek futtatását. Ilyenkor modell a módosításokat első lépésben csak a memóriában végzi el, utána menti vissza azokat az adatbázisba.
 - Az **isDirty(<index>)** metódus mutatja (igazat ad), amennyiben a modellben tárolt adat eltér az adatbázisban tárolttól.

Szerkesztési stratégia beállítása

- ❑ A **setEditStrategy (<stratégia>)** függvényével definiálhatjuk a visszamentés módját, ez a következő lehetnek:
 - **OnFieldChange**: amint váltjuk a mezőt, automatikusan meghívja a **submit ()** utasítást
 - **OnRowChange**: amint váltjuk a sort, automatikusan meghívja a **submit ()** utasítást
 - **OnManualSubmit**: nem történik változtatás, amíg meg nem hívjuk a mentés (**submitAll ()**) vagy visszavonás (**revertAll ()**) műveletét
- ❑ A mentő műveletek hamissal térnek vissza sikertelen mentéskor, ekkor a **lastError ()** tartalmazza a hibát.
 - Egy sort, vagy adatot menteni a **submit ()**, a teljes tartalmat menteni a **submitAll ()** utasítással tudunk.
 - Lehetőségünk van változtatások visszavonására is **revert ()** és **revertAll ()** metódusokkal.

Tranzakciók

- Lehetőségünk van az adatok konzisztenciáját *tranzakciók* segítségével biztosítani (ha az adatbázis-kezelő támogatja).
 - A `transaction()` utasítás indítja a tranzakciót, amelyet a `commit()` utasítással véglegesíthetünk, a `rollback()` utasítással visszavonhatunk.
 - Amennyiben valamelyik utasítás hibásnak bizonyul, visszaállíthatjuk az adatbázis konzisztens állapotát, ezért célszerű használni a `submitAll()` utasítás esetén.

```
db.transaction();           // tranzakció indítása
if (model->submitAll())      // módosítások mentése
    db.commit();            // ha sikeres, véglegesítünk
else
    db.rollback();          // ha sikertelen, visszavonjuk
```

Kapcsolt tábla modell

- ❑ Adatbázisbeli relációk segítségével kapcsolt adatokat a `QSqlRelationalTableModel` segítségével kezelhetünk.
 - a `setRelation(<oszlop>, <reláció>)` metódussal beállíthatunk relációt egy adott oszlopra
 - a reláció típusa `QSqlRelation`, megadja a tábla nevét, a forrás (társított), valamint a cél (megjelenített) oszlopot
- ❑ A relációval kapcsolt tábla egy külön modellt hoz létre az alkalmazásban, amelyet lekérdezhetünk és szerkeszthetünk
 - a `relationModel(<oszlop>)` metódus visszaadja a csatolt táblához tartozó modellt.

```
QSqlRelationalTableModel model;  
model.setTable("myTable");  
model.setRelation(2, QSqlRelation("otherTable", 0, 1));  
QSqlTableModel *otherModel = model.relationModel(2);  
otherModel->data(...); // adat lekérdezése
```

Kapcsolt táblák megjelenítése

- ❑ A társított adatok megjelenésének módját delegált típus segítségével adhatjuk meg:
 - A nézet `setItemDelegate(<delegált>)` metódusa segítségével állíthatunk be az alapértelmezett delegálttól eltérőt.
 - A társított adatokat például legördülő menü segítségével is megjeleníthetjük a `QSqlRelationalDelegate` példányra használatával.

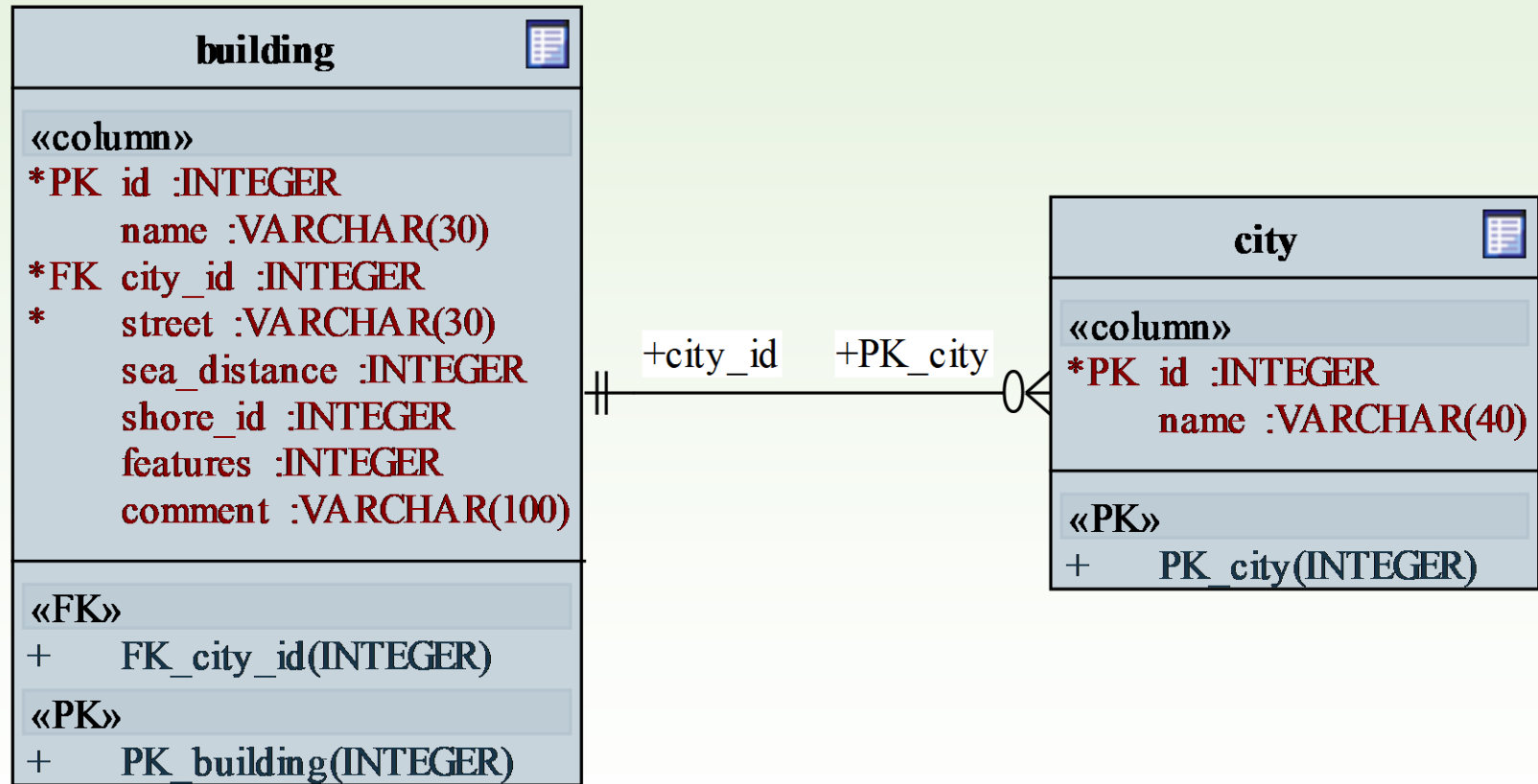
```
QTableView view;  
view.setModel(model);  
view.setItemDelegate(new QSqlRelationalDelegate());
```

3.Feladat

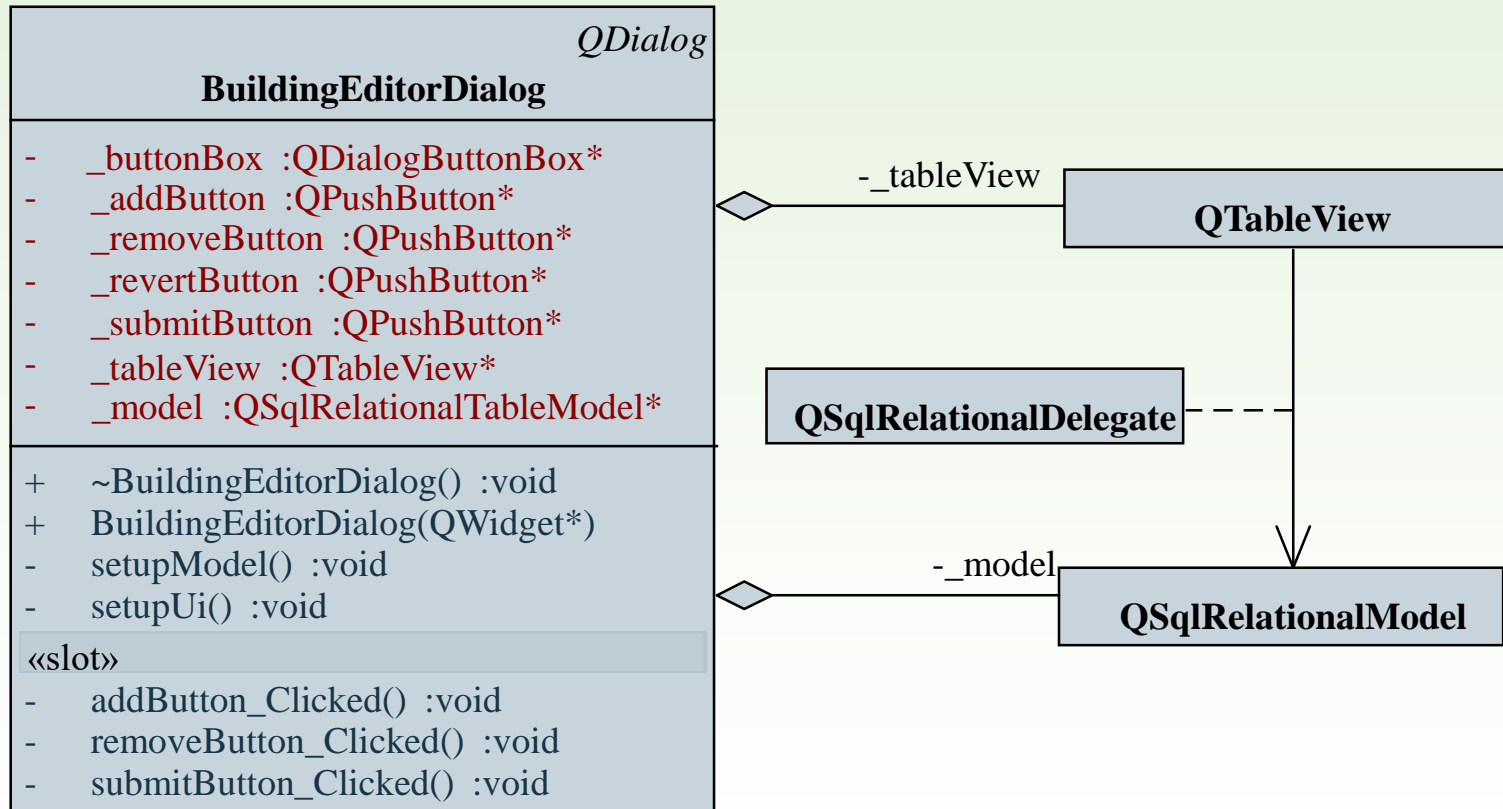
Módosítsuk az épületek szerkesztését úgy, hogy a városokat hozzácsatoljuk az megjelenítéshez.

- Ehhez relációs adatmodellt kell használnunk, amely létrehozza a relációt a városok táblával (**city**), az épületek táblabeli azonosítót (**city_id**) kötve az azonosítóhoz (**id**), és helyette megjelenítve a nevet (**name**).
- A megjelenítéshez lecseréljük a delegáltat is, így legördülő menü fog megjelenni.
- Az adatok mentését manuálisan valósítjuk meg tranzakciók segítségével egy külön gombbal.

3.Feladat: tervezés



3.Feladat: tervezés



3.Feladat: megvalósítás

```
void BuildingEditorDialog::setupModel()
{
    _model = new QSqlRelationalTableModel(this);
    _model->setTable("building");
    _model->setSort(1, Qt::AscendingOrder);
    _model->setEditStrategy(QSqlTableModel::OnManualSubmit);
    ...
    _model->setRelation(2, QSqlRelation("city", "id", "name"));
    // reláció beállítása egy oszlophoz
    _model->select();
}
```

3.Feladat: megvalósítás

```
void BuildingEditorDialog::setupUi()
{
    ...
    connect(_addButton, SIGNAL(clicked()),
            this, SLOT(addButton_Clicked()));
    connect(_removeButton, SIGNAL(clicked()),
            this, SLOT(removeButton_Clicked()));
    connect(_submitButton, SIGNAL(clicked()),
            this, SLOT(submitButton_Clicked()));
    connect(_revertButton, SIGNAL(clicked()),
            _model, SLOT(revertAll())); // visszavonás
    _tableView = new QTableView(this);
    _tableView->setModel(_model);
    _tableView->resizeColumnsToContents(); // automatikus oszlopméret
    _tableView->setItemDelegate(new QSqlRelationalDelegate());
                                // megjelenítés módjának definiálása
    ...
}
```

3.Feladat: megvalósítás

```
void BuildingEditorDialog::submitButton_Clicked()
{
    _model->database().transaction(); // átváltunk tranzakciós üzemmódba
    if (model->submitAll()) { // mentés
        _model->database().commit();
    } else { // amennyiben sikertelen volt
        _model->database().rollback(); // visszavonjuk
        QMessageBox::warning(this,
            trUtf8("Hiba történt a mentéskor!"),
            trUtf8("Az adatbázis a következő hibát jelezte: %1").
                arg(model->lastError().text()));
    }
}
```

Adatkezelés problémái

- ❑ Adatbázistartalom alkalmazáson keresztüli kezelése számos problémát felvet, amit figyelembe kell vennünk, pl.:
 - adatok helyességének ellenőrzése (pl. tartomány, formátum)
 - adatok meglétének ellenőrzése (kötelezően kitöltendő mezők esetén), esetleges kitöltése alapértelmezett értékkel
 - speciális adatmegjelenítés (pl. mértékegységek)
 - kapcsolt táblák adatainak együttes, vagy külön kezelése, szerkesztése
 - adatbázisban indirekt tárolt adatok megjelenítése (pl. aggregált információk kapcsolt táblából), esetlegesen szerkesztése

Változások követése

- ❑ Az adatmodellek lehetőséget adnak az adatokban, illetve a szerkezetben történt változások követésére, amelyeket felhasználhatunk ellenőrzések, vagy automatikus kitöltések végrehajtására, pl.:
 - kezelhetjük adatok változását közvetlenül a változást követően a `dataChanged(<tartomány bal felső indexe>, <jobb alsó indexe>)` eseménnyel
 - kezelhetjük a sorok (rekordok) változását az adatbázisba történő mentéskor (`submit()` és `submitAll()` lefutásakor) a `beforeInsert(<rekord>)`, `beforeDelete(<sorszám>)` és `beforeUpdate(<sorszám>, <rekord>)` eseményekkel
- ❑ A változáskövetést célszerű manuális szerkesztési stratégiával használni, mivel így a változtatások visszavonhatóak (`revertAll()`) mentés előtt.

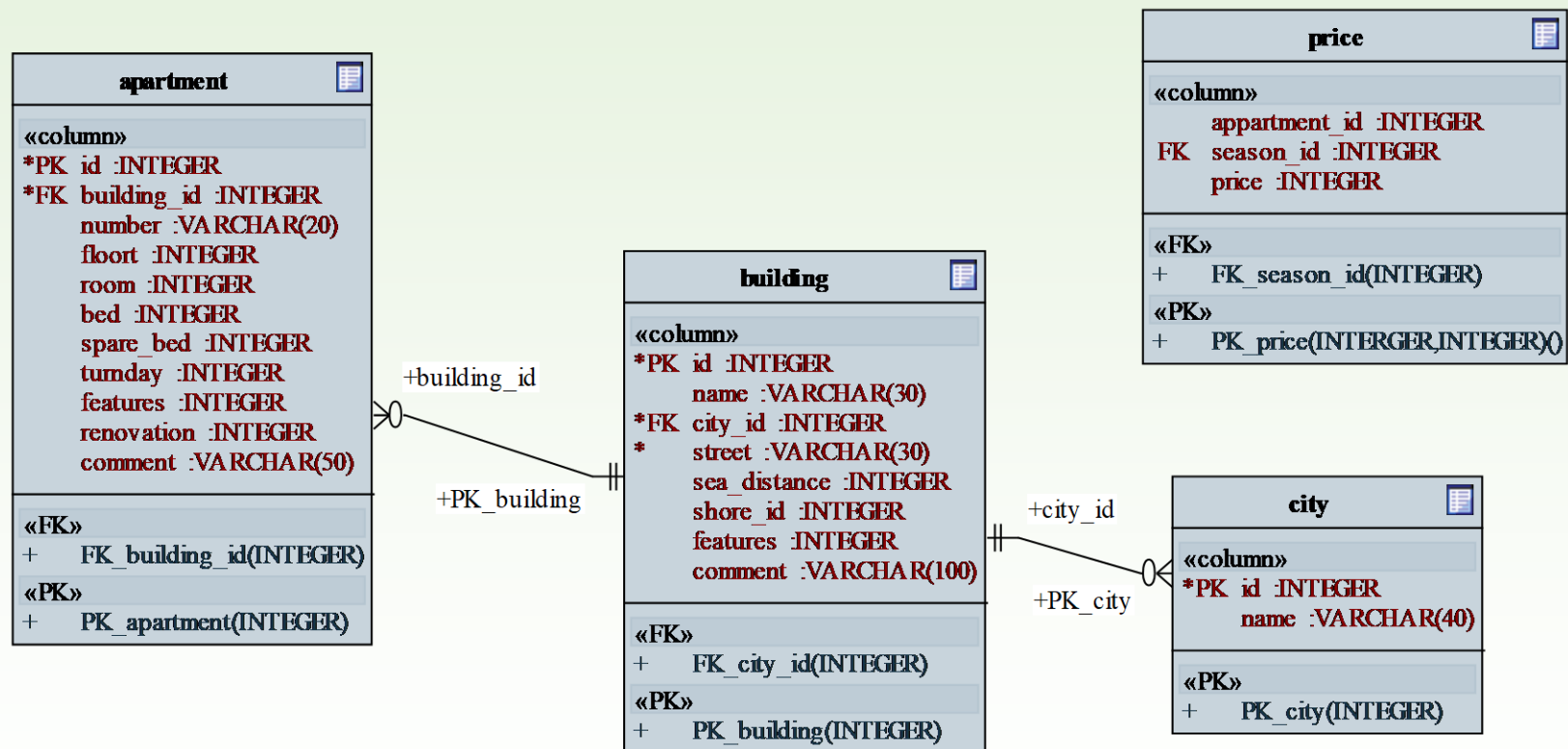
```
model_dataChanged(const QModelIndex topLeft, ...) {  
    if (topLeft.column() == 3 && model.value(topLeft).isNull())  
        // ha a 3-as oszlopban vagyunk, és elfelejtettük kitölteni  
        model.setData(topLeft, 0); // utólag kitölthetjük 0-ra  
}
```

1.Feladat

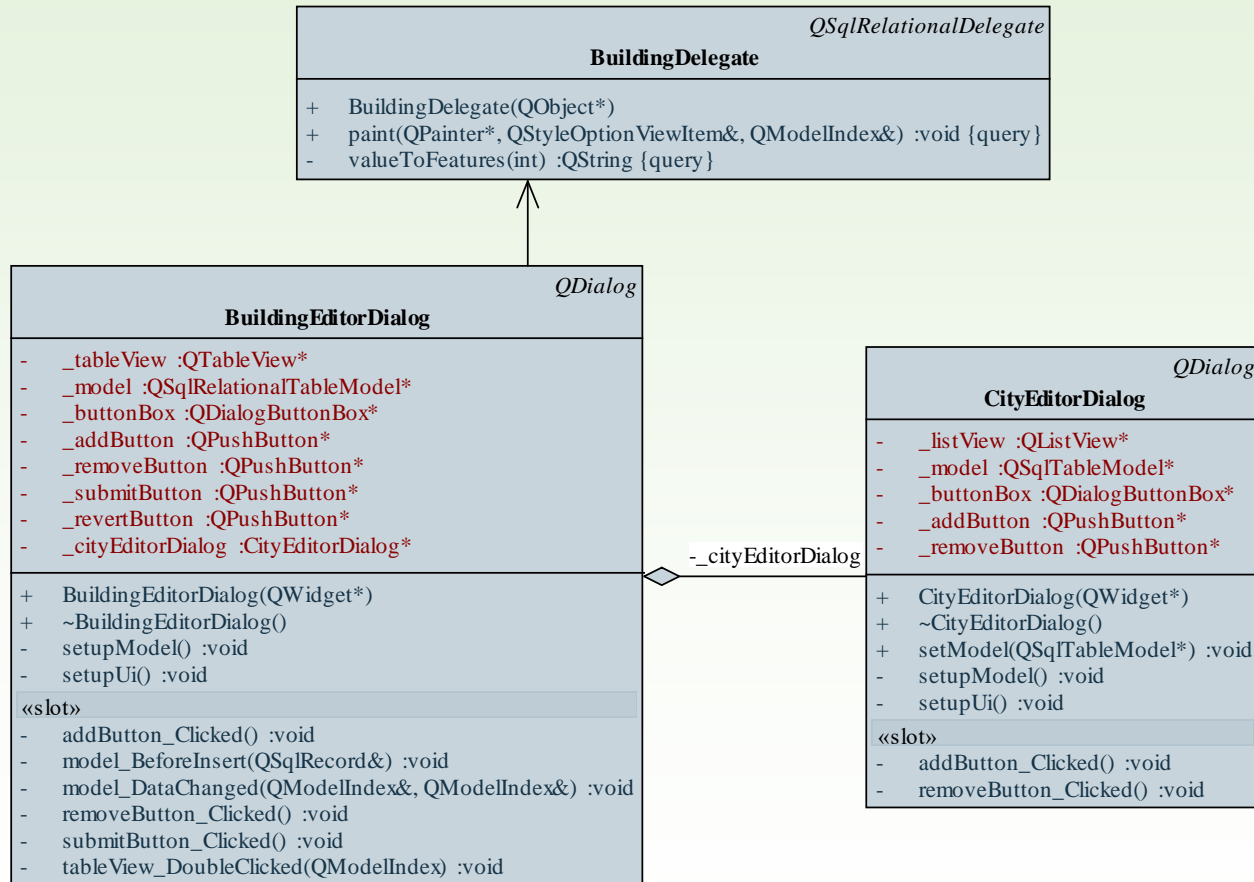
Módosítsuk az épületek kezelését úgy, hogy egy épület adatainak megjelenése minél **beszédesebb** legyen. (Ennek egyik részletét, hogy a település kód helyett a település neve látszódjon, az idegenkulcs kapcsolat alapján már megoldottuk.)

- Az épület tengerparttól vett távolságnak (*sea_distance*) megjelenítésénél vegyük hozzá az „ m” szöveget a számhoz, illetve 1 érték esetén írjuk ki azt, hogy „közvetlen”.
- A tengerpartot jellemző egész szám (*shore*) helyett annak jelentését írjuk ki: homokos (0), sziklás (1), kavicsos (2), apró kavicsos (3)
- Az épület jellemzésére használt egész számot (*features*), amelynek bitjei az épület valamilyen tulajdonságának meglétét vagy hiányát kódolják, a meglevő tulajdonságok szöveges leírásainak összefűzésével jelenítjük meg.

Adatbázis



1.Feladat: tervezés



1.Feladat: távolság megjelenítése

```
BuildingDelegate::BuildingDelegate(QObject *parent) :  
    QSqlRelationalDelegate(parent) {}  
  
void BuildingDelegate::paint( QPainter *painter,  
    const QStyleOptionViewItem &option, const QModelIndex &index) const  
{  
    switch (index.column()) {  
    case 4: // tengerpart távolság oszlop  
        QString text;  
        int shoreDistance = index.data().toInt();  
        if (shoreDistance == 1) text = "közvetlen";  
        else text = QString::number(shoreDistance) + " m";  
        QStyleOptionViewItem optionViewItem = option;  
        optionViewItem.displayAlignment = Qt::AlignRight | Qt::AlignVCenter;  
        drawDisplay(painter, optionViewItem, optionViewItem.rect, text);  
        drawFocus(painter, optionViewItem, optionViewItem.rect);  
        break;  
    ...  
}
```

kiírás módja

adat lekérdezése

kiírás

1.Feladat: part megjelenítése

```
...
case 5: // tengerpart típus oszlop
    QString text = shoreList().at(index.data().toInt());
    QStyleOptionViewItem optionViewItem = option;
    optionViewItem.displayAlignment = Qt::AlignLeft | Qt::AlignVCenter;
    drawDisplay(painter, optionViewItem, optionViewItem.rect, text);
    drawFocus(painter, optionViewItem, optionViewItem.rect);
    break;
...
```

a szöveget egy listából kérdezzük le

```
QStringList BuildingDelegate::shoreList() const
{
    QStringList list;
    list.append(trUtf8("Homokos"));
    list.append(trUtf8("Sziklás"));
    list.append(trUtf8("Kavicsos"));
    list.append(trUtf8("Apró kavicsos"));
    return list;
}
```

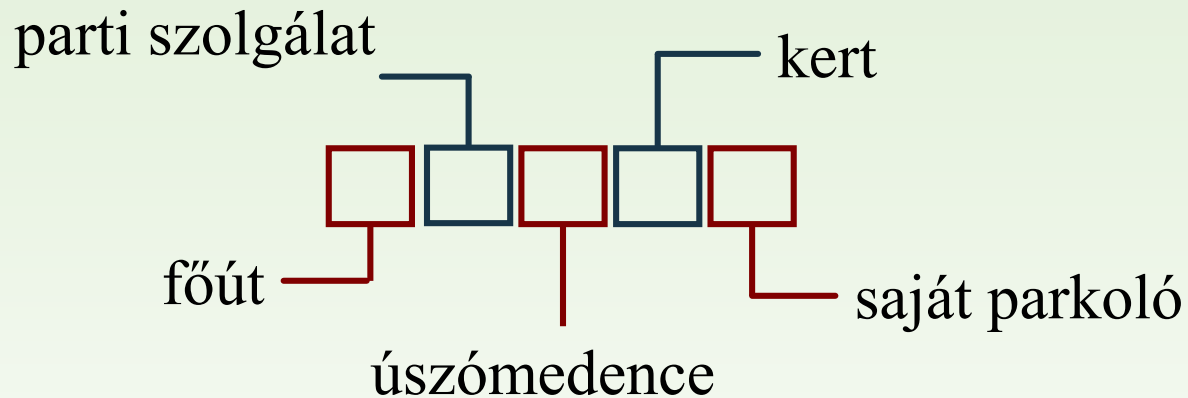
1.Feladat: jellemzők kiírása

```
...
case 6: // jellemzők oszlop
    QString text;
    if (index.data().isNull() || index.data().toInt() == 0) {
        text = "nincsenek";
    } else {
        text = valueToFeatures(index.data().toInt());
    }
    QStyleOptionViewItem optionViewItem = option;
    optionViewItem.displayAlignment = Qt::AlignLeft | Qt::AlignVCenter;
    drawDisplay(painter, optionViewItem, optionViewItem.rect, text);
    drawFocus(painter, optionViewItem, optionViewItem.rect);
    break;
default: // különben az alapértelmezett kirajzolást végezze
    QSqlRelationalDelegate::paint(painter, option, index);
    break;
}
}
```

lekérdezett adatok átalakítása

kiírás

1.Feladat: megvalósítás



```
QString BuildingDelegate::valueToFeatures(int value) const
{
    QString result;
    if (value % 2 == 1)          result += trUtf8("főút, ");
    if ((value >> 1) % 2 == 1) result += trUtf8("parti szolgálat, ");
    if ((value >> 2) % 2 == 1) result += trUtf8("úszómedence, ");
    if ((value >> 3) % 2 == 1) result += trUtf8("kert, ");
    if ((value >> 4) % 2 == 1) result += trUtf8("saját parkoló, ");
    if (result.size() > 0)      return result.left(result.size() - 2);
    else    return result;
}
```

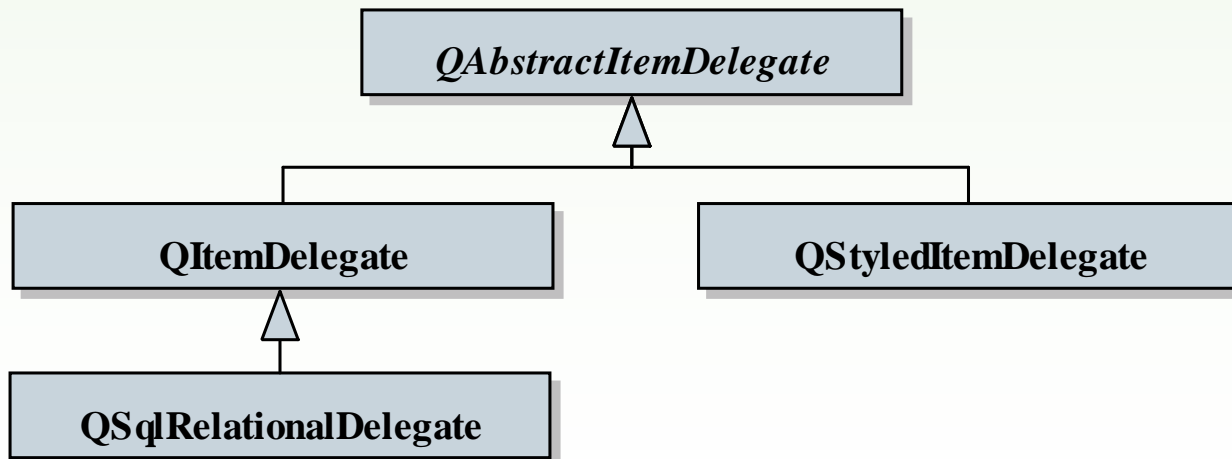
a jellemzők lekérdezését
bitenkénti eltolással oldjuk meg

A szerkesztés egyedi megjelenítése

- ❑ A saját, származtatott delegált osztályokkal az adatelemek értékének egy adatmezőben történő megjelenését nemcsak azok kiírása, de szerkesztési módját is egyedire szabhatjuk.
 - a **createEditor(...)** művelet felelős a szerkesztőmező tetszőleges **QWidget**-ként való létrehozásáért, amely akkor jelenik meg az adatelem mezőjében, amikor azt szerkeszteni akarjuk
 - a **setEditorData(...)** felelős azért, hogy a szerkesztőmező widget-jében a megfelelő modellbeli adat értéke jelenjen meg.
 - a **setModelData(...)** felelős a szerkesztőmezőben történt módosítás visszaírásáért a modellbe.
- ❑ A szerkesztést táblázat esetén oszloponként is szabályozhatjuk, de hívhatjuk közvetlenül az ősoosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

Adatmegjelenítés szabályozása

- ❑ Az adatok csoportos megjelenítéshez a nézet számos osztályt (**QListView**, **QTableView**, **QTreeView**) biztosít, amelyekből származtatással továbbiakat definiálhatunk. Ezek adatmezőiben az adatelemek értékeinek megjelenési módját a *delegált* (**QAbstractItemDelegate** leszármazott) osztályok biztosítják.
- ❑ Az előre definiált delegált osztályok közül az alap megjelenítést a **QItemDelegate**, a relációk kezelését a **QSqlRelationalDelegate**, egyedi megjelenítést pedig a **QStyledItemDelegate** szolgáltatja.

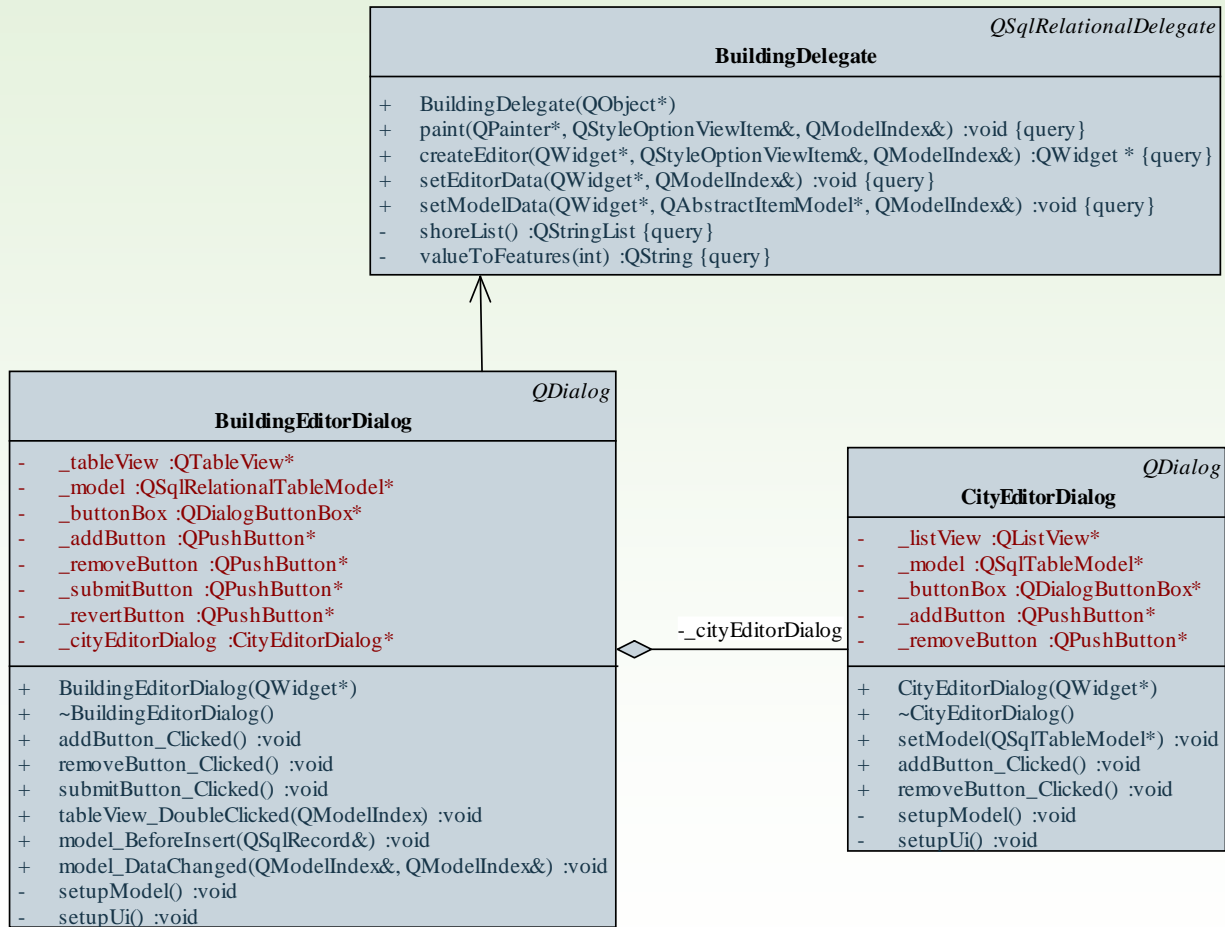


2.Feladat

Módosítsuk az épületek kezelését úgy, hogy a tengerpart típusát (homokos, sziklás, kavicsos, apró kavicsos) egy **legördülő menü** segítségével lehessen kijelölni.

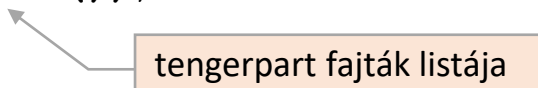
- Vezessünk be a **BuildingDelegate** osztályban egy **QComboBox** típusú egyedi vezérlőt, amely elemeit a parttípusokat tartalmazó konstans lista (**shoreList**) segítségével töltjük fel.
- A listában az index segítségével állítjuk a parttípust, így könnyen számolható a legördülő menüben kiválasztott elem (a **currentIndex** segítségével), valamint az adatbázisban visszaírandó érték is.

2.Feladat: tervezés



2.Feladat: szerkesztés létrehozása


```
QWidget* BuildingDelegate::createEditor( QWidget *parent,
    const QStyleOptionViewItem &option, const QModelIndex &index) const
{
    if (index.column() == 5) { // a tengerpart oszlopnál legördülő menü
        QComboBox *shoreComboBox = new QComboBox(parent);
        shoreComboBox->addItem(shoreList());
        return shoreComboBox;
    } else
        return QSqlRelationalDelegate::createEditor(parent,
                                                    option, index);
}
```



tengerpart fajták listája

2.Feladat: szerkesztés megjelenítése

```
void BuildingDelegate::setEditorData(  
    QWidget *editor, const QModelIndex &index) const  
{  
    if (index.column() == 5) {  
        int i = index.data().toInt();  
        QComboBox *shoreComboBox = qobject_cast<QComboBox*>(editor);  
        shoreComboBox->setCurrentIndex(i);  
    }  
    else QSqlRelationalDelegate::setEditorData(editor, index);  
}
```



szerkesztőmező elemének beállítása

2.Feladat: szerkesztés mentése

```
void BuildingDelegate::setModelData(QWidget *editor,  
    QAbstractItemModel *model, const QModelIndex &index) const  
{  
    if (index.column() == 5) {  
        QComboBox *shoreComboBox = qobject_cast<QComboBox *>(editor);  
        model->setData(index, shoreComboBox->currentIndex ());  
    }  
    else QSqlRelationalDelegate::setModelData(editor, model, index);  
}
```

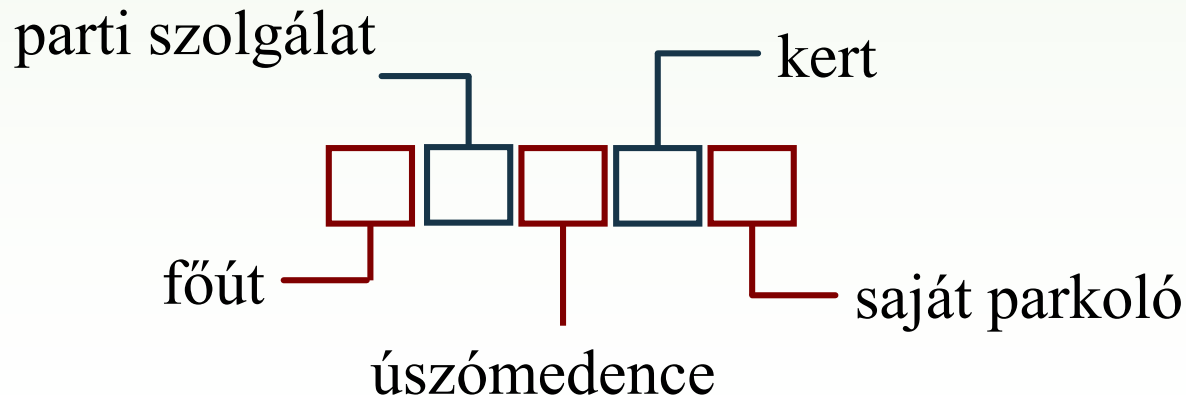
adat visszaírása a modellbe

3.Feladat

Az eddigiek mellett oldjuk meg az **épületek jellemzőinek** kényelmes szerkesztését.

- Megjeleníteni az épületek jellemzőit – azok felsorolásával – már szépen tudjuk, de szerkesztésnél egy egész számot kellett beírni, amelynek 1-es bitjei utalnak a meglévő tulajdonságokra.

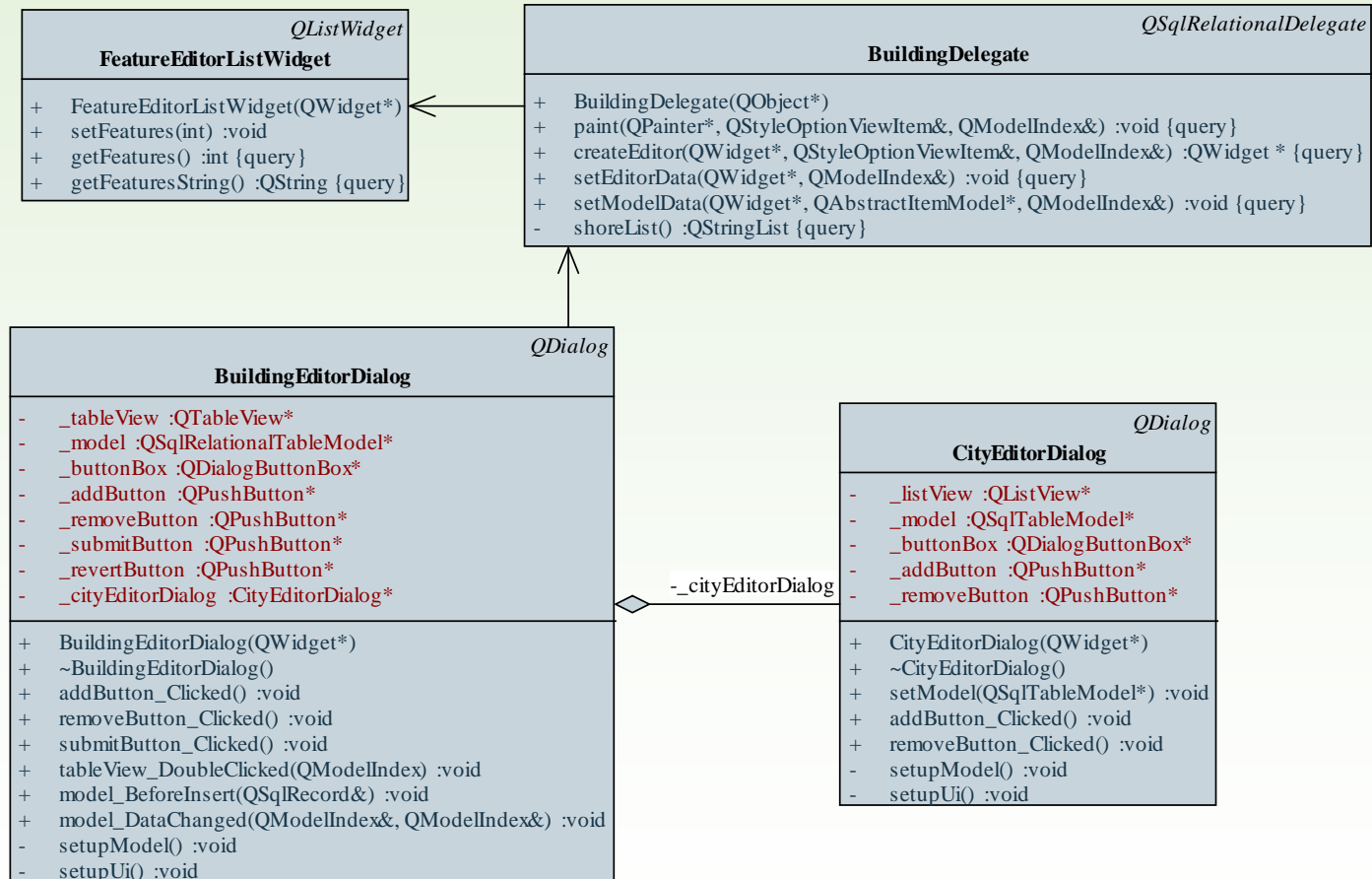
Javítsunk az épületek jellemzőinek módosításán úgy, hogy ne egy számot kelljen beírni, hanem egy listából lehessen kiválasztani az érvényes jellemzőket.



3.Feladat: tervezés

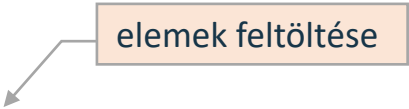
- ❑ Mivel az adatbázisban továbbra is a szám lesz eltárolva, szükségünk lesz egy egyedi lista vezérlőre (**FeatureEditorListWidget**), amely
 - elvégzi a szám-szöveg konverziót (**setFeatures**, **getFeatures**),
 - biztosítja a szöveges formájú kiírást (**getFeaturesString**),
 - listaszerűen jeleníti meg az adatokat.
- ❑ Ezt a **QListWidget** vezérlőből származtatjuk, amelyben lehetőség van több elem egyidejű kijelölésére, így közvetlenül tárolhatjuk a jellemzők állapotát.
- ❑ Az egyedi vezérlőnket a delegált (**BuildingDelegate**) segítségével helyezzük a szerkezetbe.

3.Feladat: tervezés



3.Feladat: megvalósítás

```
FeatureEditorListWidget::FeatureEditorListWidget(QWidget *parent)
    : QListWidget(parent)
{
    addItem(trUtf8("főút"));
    addItem(trUtf8("parti szolgálat"));
    addItem(trUtf8("úszómedence"));
    addItem(trUtf8("kert"));
    addItem(trUtf8("saját parkoló"));
}
```



elemek feltöltése

```
QString FeatureEditorListWidget::getFeaturesString() const
{
    QString result;
    for (int i = 0; i < 5; i++)
        if (item(i)->checkState() == Qt::Checked)
            result += item(i)->data(Qt::DisplayRole).toString() + ", ";
    if (result.size() > 0) return result.left(result.size() - 2);
    else return "nincsenek";
}
```

3.Feladat: megvalósítás

```
void FeatureEditorListWidget::setFeatures(int features)
{
    for (int i = 0; i < 5; i++){
        if (((features >> i) % 2 == 1))
            item(i)->setCheckState(Qt::Checked);
        else
            item(i)->setCheckState(Qt::Unchecked);
    }
}
```

kijelölés beállítása a bit értéke szerint

```
int FeatureEditorListWidget::getFeatures() const
{
    int featuresInt = 0;
    for (int i = 0; i < 5; i++)
        if (item(i)->checkState() == Qt::Checked)
            featuresInt += pow(2, i);
    return featuresInt;
}
```

megfelelő hatványozás a beíráshoz

Számított adatok megjelenítése

- ❑ Lehetőségünk van a modellben a tényleges adatbázisbeli tartalom mellett, vagy helyett tetszőleges **számított adat** megjelenítésére.
 - Ehhez egy új, speciális modellt kell származtatnunk, amelyben felüldefiniáljuk az
 - adatlekérdezést végző **data(<index>, <szerep>)** metódust, amely a pozíció (index) alapján határozza meg a megjeleníteni kívánt adatot
 - valamint az oszlopok számát megadó **columnCount()** metódust, amelynek általában növeljük az értéket
- ❑ Mindkét műveletben hívhatjuk az őssosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

Adat-kezelési szerepek

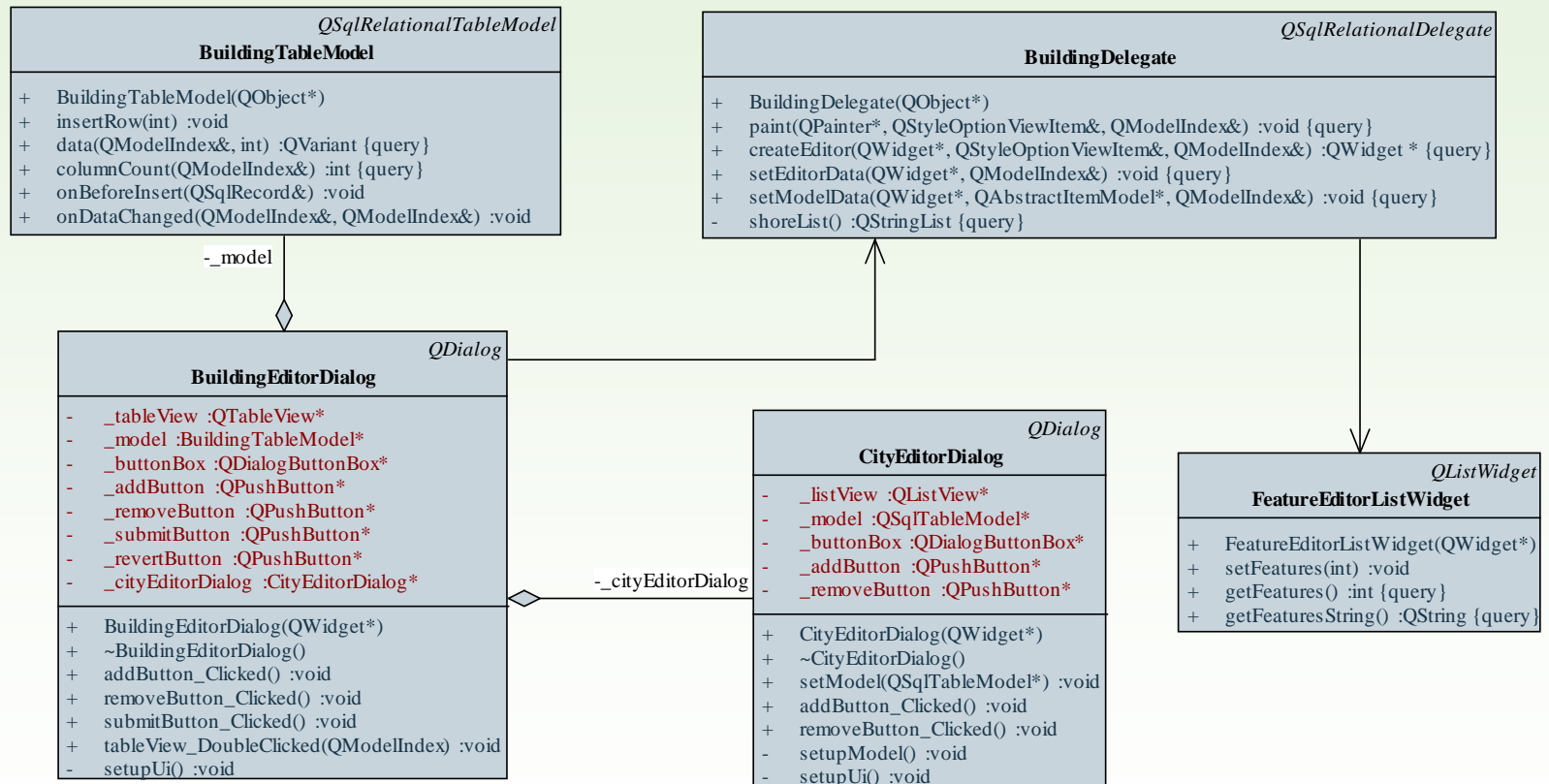
- ❑ A **data** metódus szerep (**role**) paramétere mutatja, hogy milyen információ lekérdezése céljából hívják meg a metódust.
- ❑ A leggyakoribb szerepek:
 - **Qt::DisplayRole**: megjelenítés céljából kért (tárolt vagy számított modellbeli) érték (amelyet tovább változtathatunk a delegáltban)
 - **Qt::EditRole**: szerkesztés céljából kért (tárolt vagy számított modellbeli) érték, amely általában megegyezik a megjelenítés céljából lekérttel
 - **Qt::ToolTipRole**: előugró üzenet
 - **Qt::TextAlignmentRole**: szövegigazítás az adathoz
 - **Qt::TextColorRole**, ...: különböző megjelenítési beállítások, amelyek szabályozhatóak a modell szintjén, illetve a delegált szintjén is

4.Feladat

Egészítsük ki az épületek táblát három **számított oszloppal**, az épületben lévő apartmanok számával, valamint az apartmanok árai közül a legkisebb és a legnagyobbval.

- Származtatunk a relációs modellből egy egyedi modellt (**BuildingTableModel**), amelyben felüldefiniáljuk az adatlekérdezést, az oszlopok számát, illetve az új sor beszúrását (az alapértelmezett értékek beszúrása végett).
- A három új értéket megfelelő lekérdezések segítségével hozzuk létre (pl. ár esetén az **apartment** és a **price** tábla alapján).
- A delegált osztályban az árak megjelenítését kiegészítjük a pénznem megjelölésével is.

4.Feladat: tervezés



4.Feladat: megvalósítás

```
QVariant BuildingTableModel::data(const QModelIndex &index, int role) const
{
    if (!index.isValid()) return QVariant();
    // ha nem érvényes az index, üres adatot adunk vissza
    ...
    if (index.column() == 8 &&
        (role == Qt::DisplayRole || role == Qt::EditRole)) {
        QSqlQuery query;
        query.exec( "select count(*) from apartman where id = " +
                    this->data(this->index(index.row(), 0)).toString() );
        if (query.next())
            return QVariant(query.value(0).toInt());
        else
            return QVariant(0);
    }
    ...
}
```

apartmanok számának meghatározása

4.Feladat: megvalósítás

```
...
else if (index.column() == 9 &&
        ( role == Qt::DisplayRole || role == Qt::EditRole)){ // minimum ár
    QSqlQuery query;
    query.exec("select min(price) from price where apartment_id in
                (select id from apartment where building_id = "
                + this->data(this->index(index.row(), 0)).toString() + ")");
    if (query.next()) return QVariant(query.value(0).toInt());
    else               return QVariant("?");
}
else if (index.column() == 10 &&
        ( role == Qt::DisplayRole || role == Qt::EditRole)) { // maximum ár
    QSqlQuery query;
    query.exec("select max(price) from price where apartment_id in
                (select id from apartment where building_id = "
                + this->data(this->index(index.row(), 0)).toString() + ")");
    if (query.next()) return QVariant(query.value(0).toInt());
    else               return QVariant("?");
}
else return QSqlRelationalTableModel::data(index, role);
}
```


Számított adatok szerkesztése

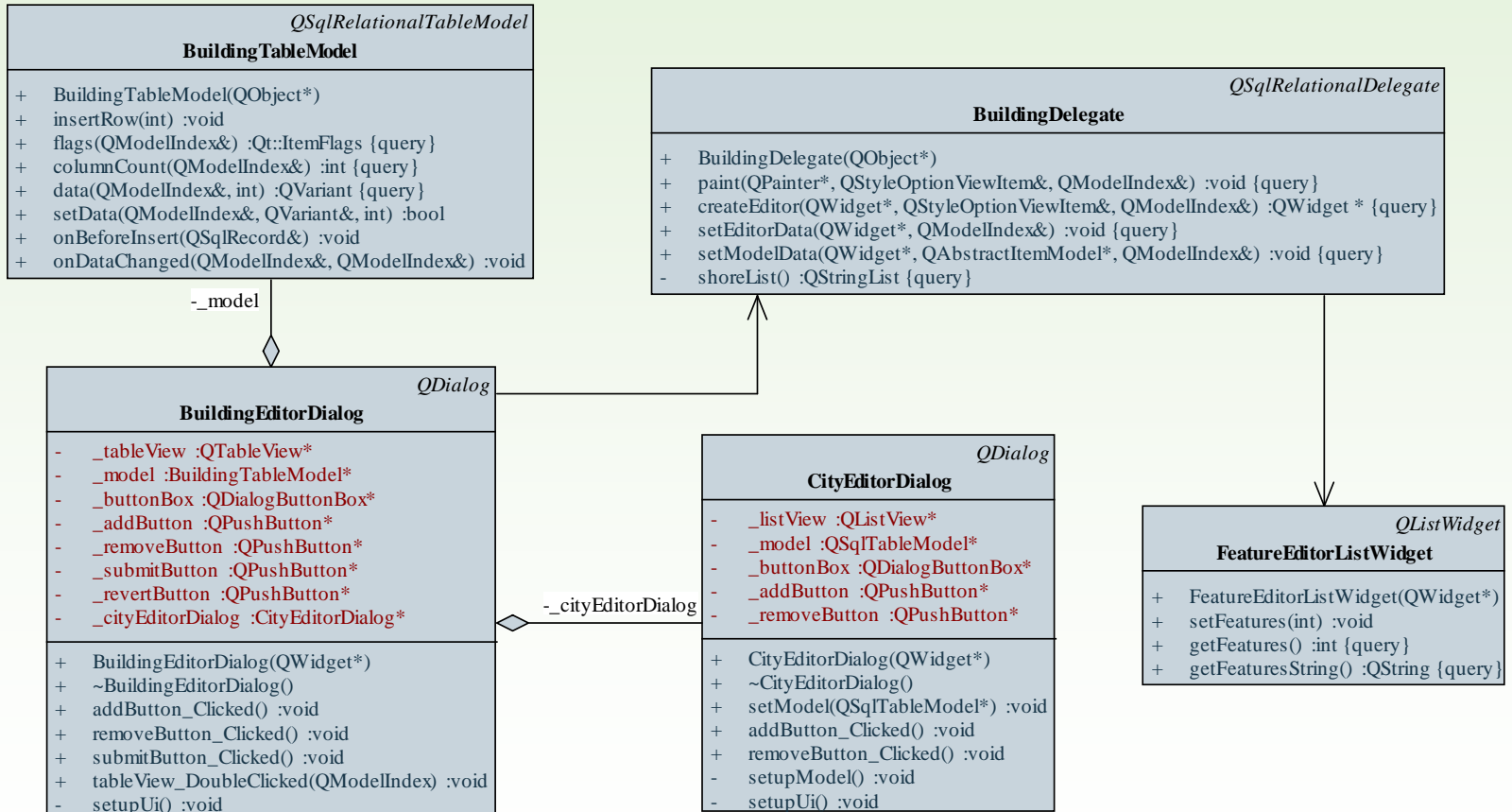
- ❑ A táblamodell nemcsak a számított adatok lekérdezését, de szerkesztését is lehetővé teszi.
 - A **setData(<index>, <érték>, <szerep>)** művelet felüldefiniálásával az adatok szerkesztését specializálhatjuk, amelyben megadhatjuk a számított adatok módosításának tényleges tevékenységét.
 - A számított oszlopot a szerkesztés előtt szerkeszthetővé kell tenni, ehhez felül kell definiálni az oszlopok állapotjelzőit visszaadó **flags(<index>)** műveletet.
 - Az adott számított oszlopnak kiválaszthatónak (**ItemIsSelectable**) és szerkeszthetőnek (**ItemIsEditable**) kell lennie.

5.Feladat

Egészítsük ki az épületek táblát egy állapot oszloppal, amely jelöli, hogy van-e tatarozás az épületben. Az állapot „normál”, ha mindegyik apartman kiadható, „lezárt”, ha mindegyik apartman tatarozás alatt van, egyébként „felújítás alatt”. Lehesse állítani az értéket úgy, hogy normál, vagy lezárt állapotba tudjuk helyezni az épületet.

- Felveszünk egy számított oszlopot, amely az adatot az apartmanok táblából gyűjti.
- A megjelenítéshez egy legördülő menüt használunk, amely csak két értéket kap meg, nem mind a hármat.
- Felüldefiniáljuk az adatbeállítást, ahol az értékeket az apartman táblába írjuk.

5.Feladat: tervezés



5.Feladat: megvalósítás

```
Qt::ItemFlags BuildingTableModel::flags(  
    const QModelIndex& index) const  
{  
    Qt::ItemFlags flag = QSqlTableModel::flags(index);  
    if (index.column() == 4) // számított oszlop  
        flag |= Qt::ItemIsSelectable | Qt::ItemIsEditable;  
    return flag;  
}
```

lekérdezzük az alap
állapotjelzőt

kiválaszthatóvá is, és
szerkeszthetővé is tesszük

```
bool BuildingTableModel::setData(  
    const QModelIndex& index, const QVariant& value, int role)  
{  
    ...  
    if (index.column() == 4) {  
        if (value.toInt() == 0) {  
            QSqlQuery query;  
            return (query.exec("update apartment" +  
                "set renovation = 0 where building_id = " +  
                this->data(this->index(index.row(), 0)).toString()));  
            ...  
        }  
    }  
}
```

apartment táblát kell módosítanunk