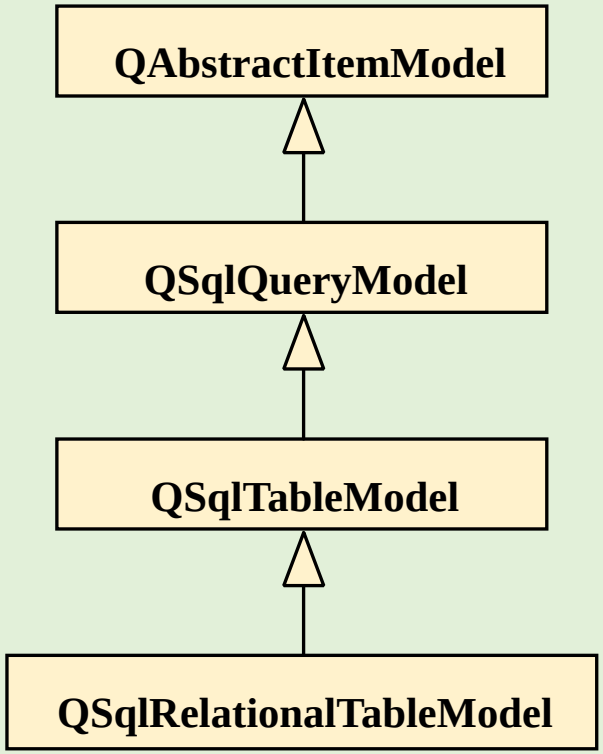
Adatbázis-kezelő modellek

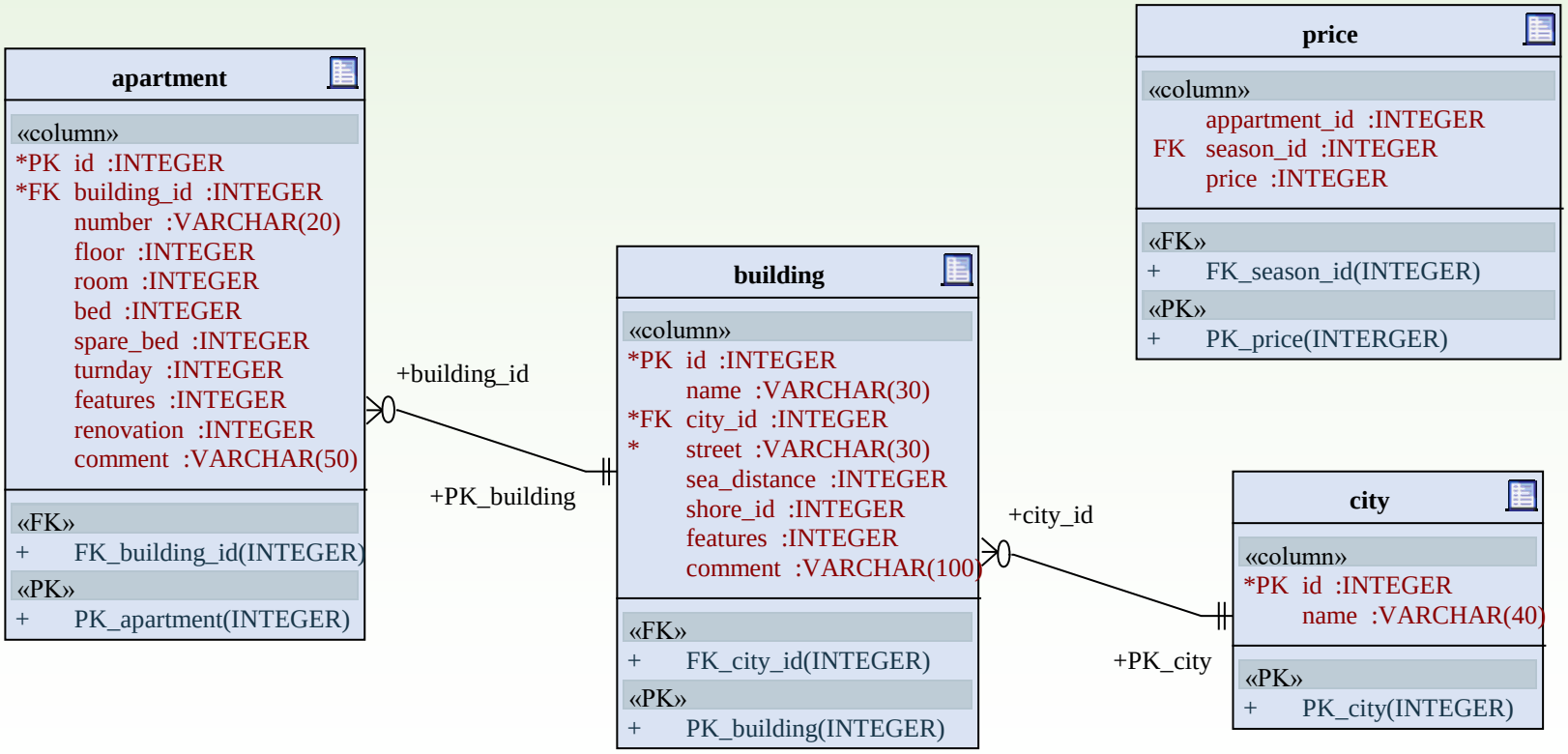
Adatbázis-kezelő modellek

- 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)



Keret probléma

A következőkben megoldott feladatok egy olyan összetett alkalmazásnak a részei, amelyet egy utazási iroda használ az általa kiközvetített apartmanok bérbeadásához.



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.

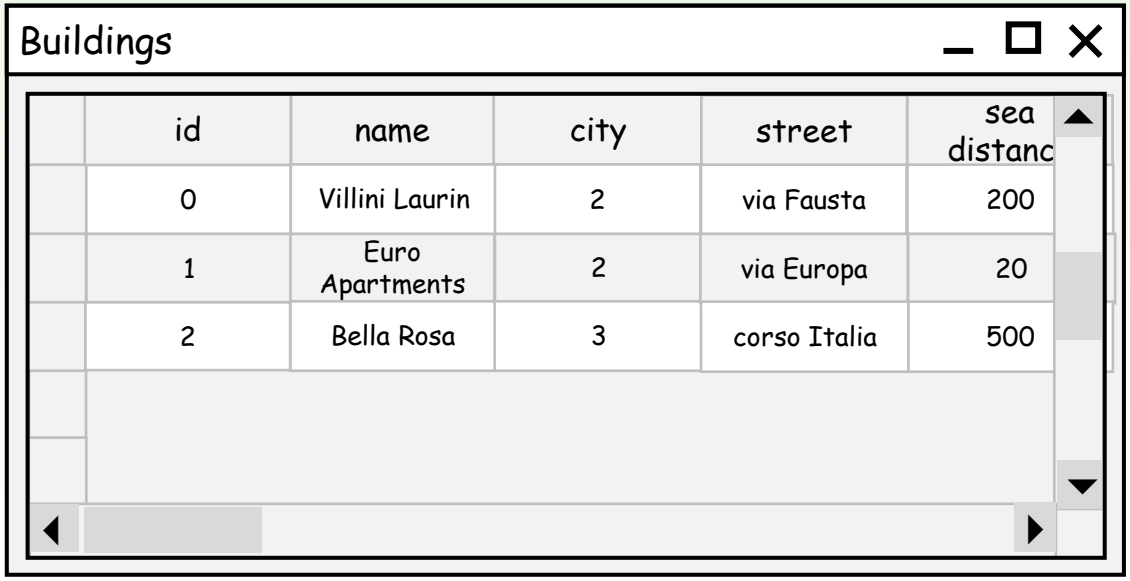
Lekérdezés modell nézete

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

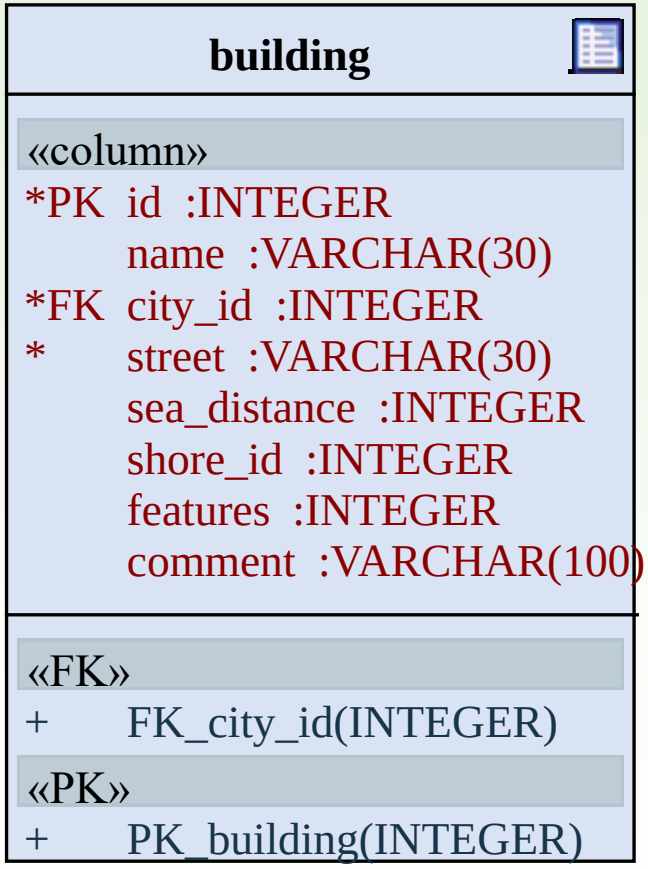
```
QSqlQueryModel model; // modell
model.setQuery("select …"); // lekérdezés
model.setHeaderData(0, Qt::Horizontal, "Id");
        // oszlop fejlécének beállítása
…
QTableView view; // nézet
view.setModel(model); // modell beállítása
view.show(); // megjelenítés
```

1.Feladat

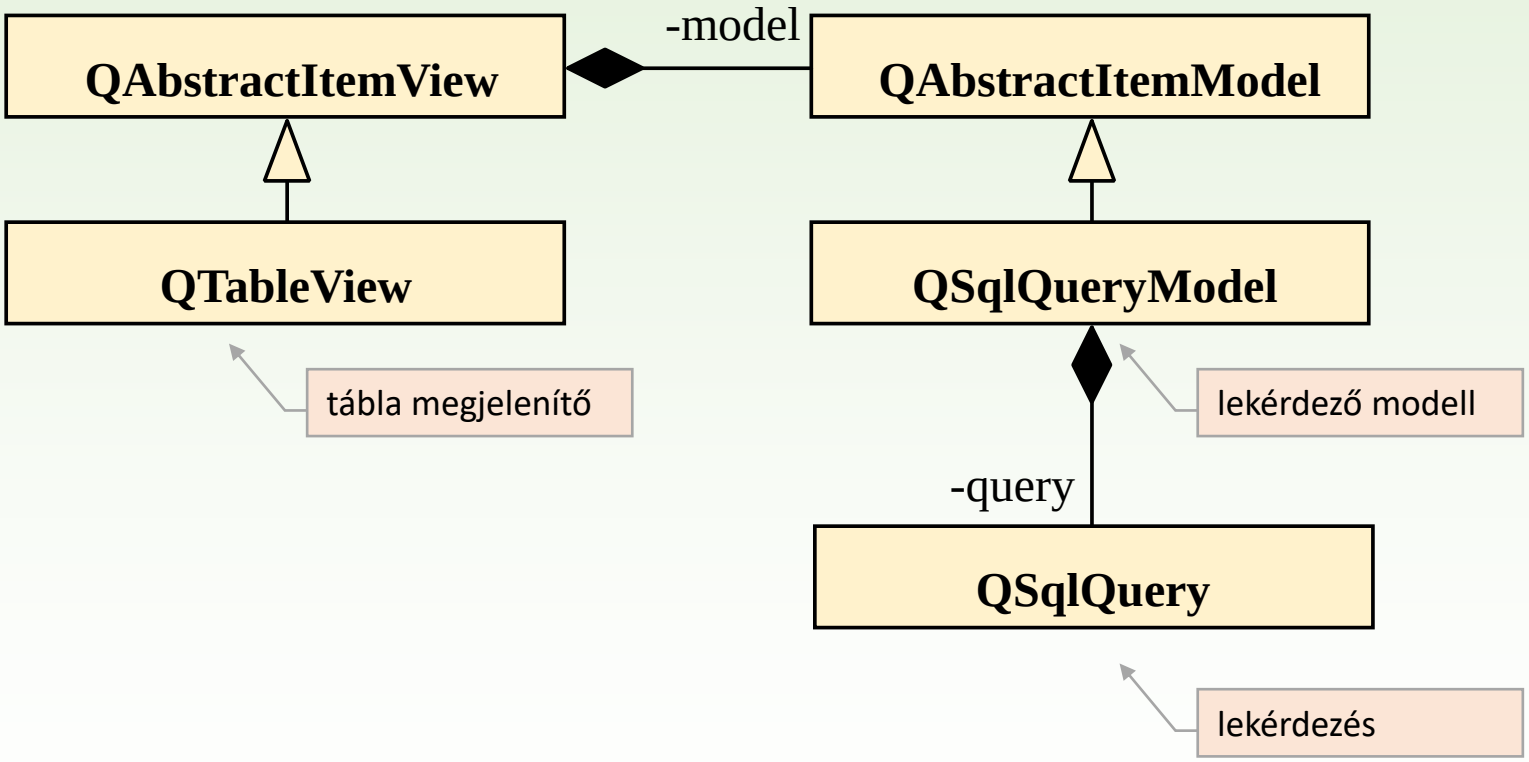
Jelenítsük meg az apartman adatbázis épületeit (**`buildings`**).



1.Feladat: adatbázis



1.Feladat: tervezés



1.Feladat: megvalósítás

```
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
    …
    QSqlQueryModel* model = new QSqlQueryModel();
    model->setQuery("select * from building");
    model->setHeaderData(0, Qt::Horizontal,tr("id"));
    model->setHeaderData(1, Qt::Horizontal,tr("name"));
    model->setHeaderData(2, Qt::Horizontal,tr("city"));
    model->setHeaderData(3, Qt::Horizontal,tr("street"));
    model->setHeaderData(4, Qt::Horizontal,tr("sea distance"));
    …
    QTableView* tableView = new QTableView();
    tableView->setModel(model);
    tableView->show();

    return a.exec();
}
```

kapcsolat létrehozása

lekérdezési modell beállítása

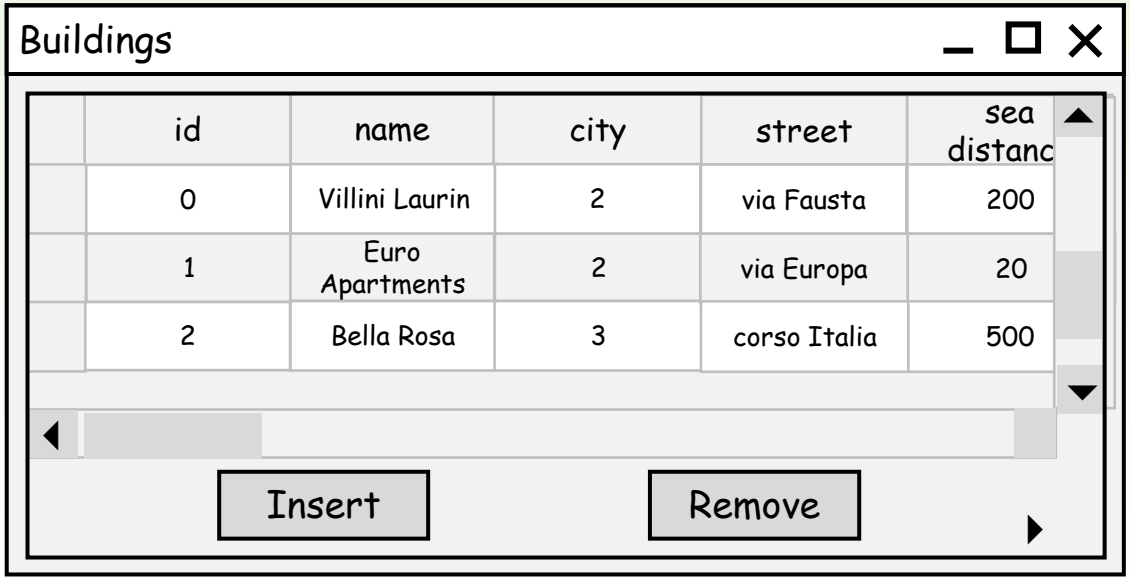
tábla megjelenítő beállítása

Tábla modell

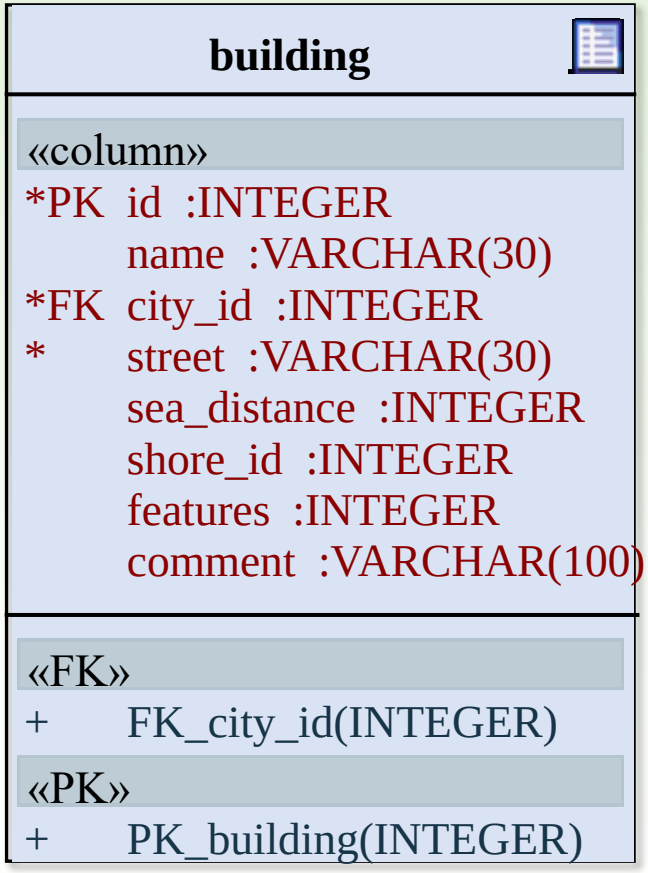
- Az adatbázisok kezelésénél a leggyakrabban használt modell a tábla.
- Egy tábla lekérdezését és szerkesztését a **`QSqlTableModel`** osztály biztosítja.
  - A **`setTable(<táblanév>)`** művelettel állíthatunk be egy táblát adatforrásnak, a **`select()`** művelet szolgál az adatok lekérdezésére.
  - 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(<index>)`** töröl egy sort.

2.Feladat

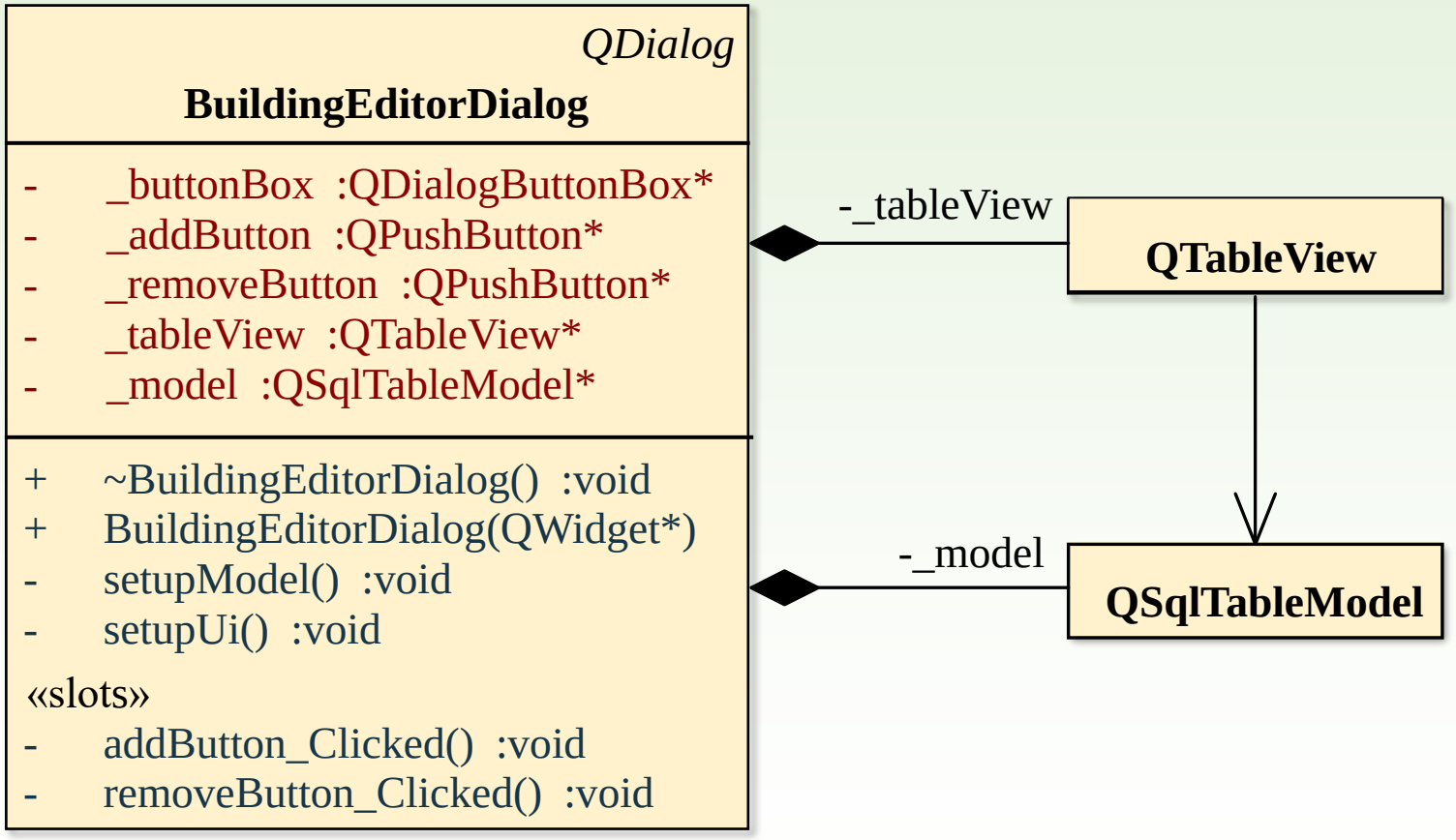
Készítsünk alkalmazást az apartman adatbázis épületeinek (**`buildings`**) szerkesztésére, új épület létrehozására, törlésére.



2.Feladat: adatbázis



2.Feladat: tervezés



2.Feladat: modell megvalósítása

```
void BuildingEditorDialog::setupModel() {
    _model = new QSqlTableModel(this); // táblamodell létrehozása
    _model->setTable("building"); // tábla beállítása
    _model->setSort(1, Qt::AscendingOrder); // rendezés oszlopra
    // fejlécek beállítása
    model->setHeaderData(0, Qt::Horizontal,tr("id"));
    model->setHeaderData(1, Qt::Horizontal,tr("name"));
    model->setHeaderData(2, Qt::Horizontal,tr("city"));
    model->setHeaderData(3, Qt::Horizontal,tr("street"));
    model->setHeaderData(4, Qt::Horizontal,tr("sea distance"));
        …
    _model->select(); // adatok begyűjtse
}
```

2.Feladat: nézet megvalósítása

```
void BuildingEditorDialog::setupUi() {
    _addButton = new QPushButton(tr("Insert"));
    _removeButton = new QPushButton(tr("Remove"));
    _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: törlés funkció

Törléskor töröljük a kijelölt sort (amennyiben van kijelölés), és áthelyezzük a kijelölést.

```
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, tr("Nincs kijelölés!"),
        tr("Kérem jelölje ki előbb a törlendő sort!"));
    }
}
```

2.Feladat: beszúrás funkció

Beszúráskor lekérdezzük a kijelölt sor indexét, behelyezünk egy sort a helyére, átállítjuk a kijelölést (az indexen keresztül), majd szerkesztésre váltunk.

```
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 **`revetAll()`** metódusokkal.

Tranzakciók

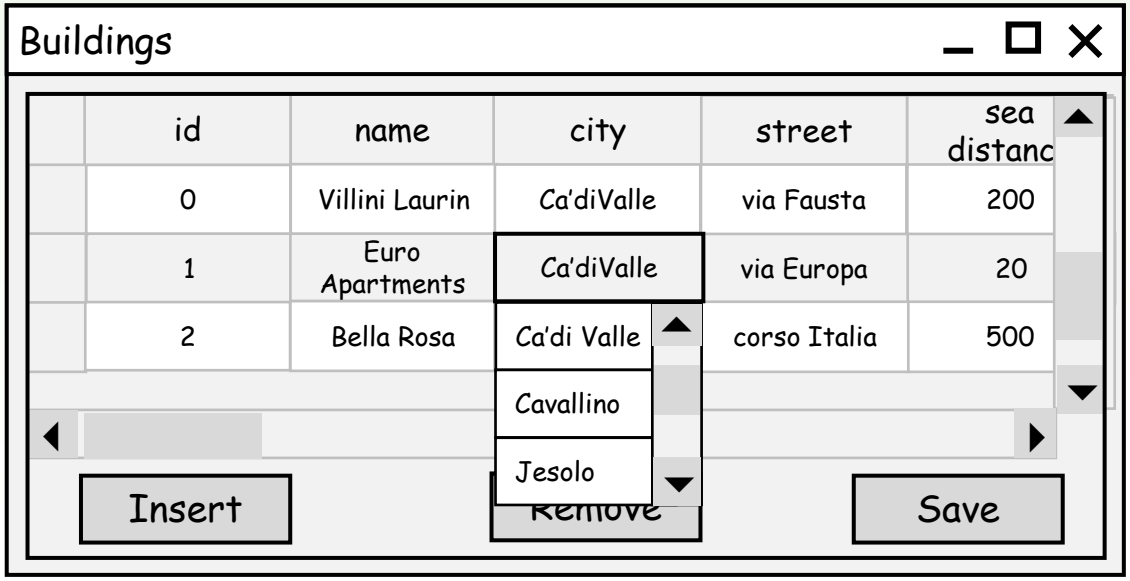
- Lehetőségünk van az adatok konzisztenciáját *tranzakció*k segítségével biztosítani (ha az adatbázis-kezelő támogatja).
  - A **`transaction()`** utasítás indítja a tranzakciót, amelyet a **`commit()`** utasítással véglegesíthetünk, a **`rollback()`** utasítással visszavonhatunk.
  - Amennyiben valamelyik 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
```

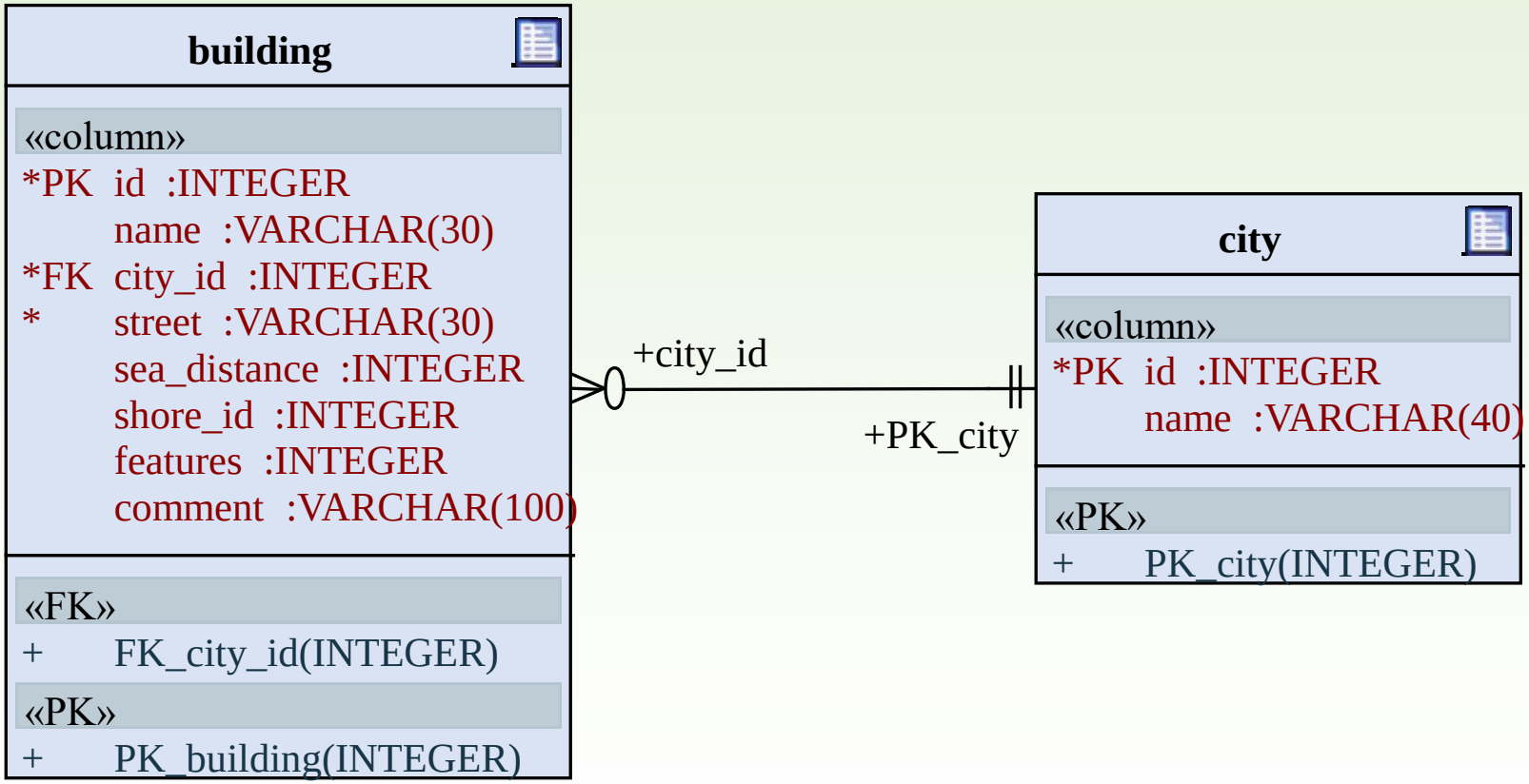
3.Feladat

Módosítsuk az épületek szerkesztését úgy, hogy

- a város azonosítók helyett a város neveket jelenítjük meg.
- a város oszlop szerkesztésekor az adott mezőben egy legördülő menü ajánlja fel a választható települések nevét
- az adatok mentését manuálisan valósítjuk meg tranzakciók segítségével egy külön gombbal.



3.Feladat: adatbázis



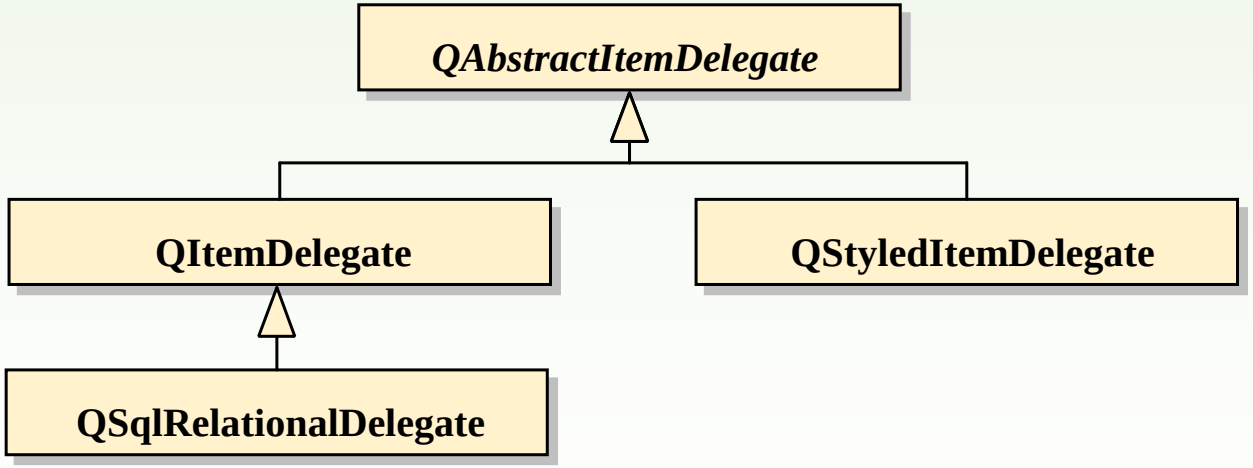
Kapcsolt tábla modell

- Adatbázisbeli relációk mentén kapcsolt adatokat a **`QSqlRelationalTableModel`** segítségével kezelhetjük.
  - 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
```

Adatmegjelenítés szabályozása

- Az előre definiált delegált osztályok közül az alap megjelenítést a **`QItemDelegate`**, a relációk kezelését a **`QSqlRelationalDelegate`**, egyedi megjelenítést pedig a **`QStyledItemDelegate`** szolgáltatja.

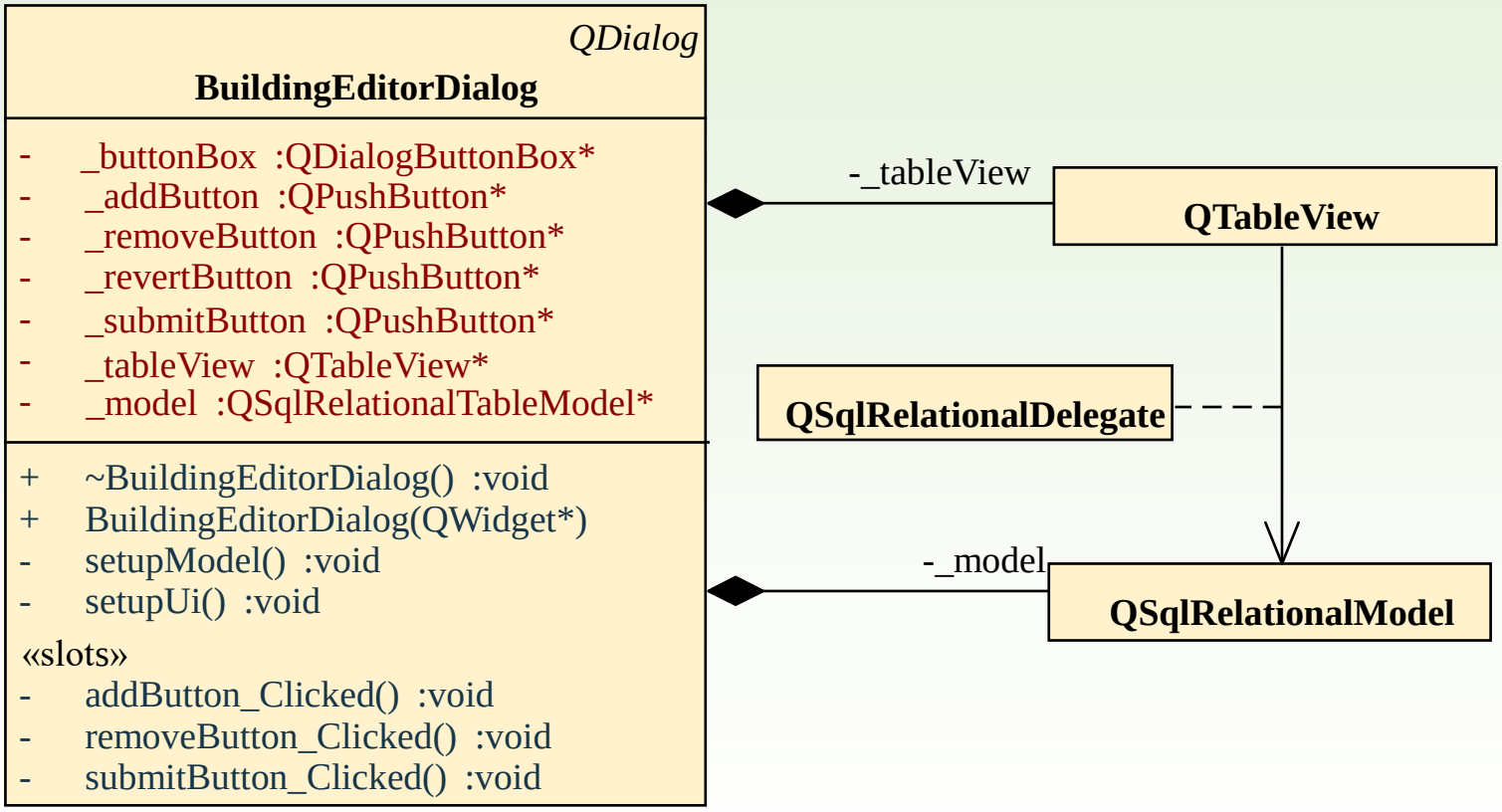


Kapcsolt táblák megjelenítése

- 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: tervezés



3.Feladat: modell megvalósítása

```
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"));

    _model->select();
}
```

reláció beállítása a building 2. oszlopához:

az oszlopban tárolt bármelyik érték helyén a city tábla azon sorának „name” értéke jelenik meg, amelyik sornak „id” értéke a 2. oszlopban tárolt szóban forgó érték.

3.Feladat: nézet megvalósítása

```
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
    …
}
```

szerkesztéskor egy lenyíló menü mutatja a választható településeket

3.Feladat: mentés funkció

```
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,
            tr("Hiba történt a mentéskor!"),
            tr("Az adatbázis a következő hibát jelezte: %1").
             arg(model->lastError().text()));
    }
}
```