Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

12. előadás

Összetett adatkezelés

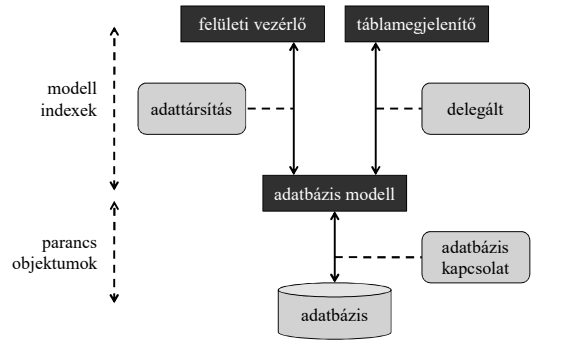
Giachetta Roberto

http://people.inf.elte.hu/groberto

Összetett adatkezelés
Adatok megjelenítése

- A Qt modell/nézet architektúrában alapvetően a nézet biztosítja az adatok megjelenítésére, amelyekből több áll rendelkezésünkre (`QListView`, `QTableView`, `QTreeView`, `QComboBox`), illetve öröklődéssel továbbiakat definiálhatunk
  - az adatokat megfelelően kell leképezni, pl. a listaszerű megjelenítő és táblamodell esetén definiálni kell a megjelenített oszlopot
  - a megjelenítést módosíthatja a delegált
- A nézet mellett azonban lehetőségünk van arra, hogy a modellbeli adatokat bármilyen felületi vezérlőből elérhessük, *adattársítás* (*data binding*) segítségével

Összetett adatkezelés
Adatok megjelenítése



Összetett adatkezelés
Adattársítás vezérlőkhöz

- Vezérlők adattársítására használható a `QDataWidgetMapper` osztály, amely egy vezérlő értékéhez tudja társítani a rekordot
  - az társítás adatforrása beállítható a `setModel(<modell>)` metódussal, míg a vezérlő beállítása az `addMapping(<vezérlő>, <oszlopszám>)` művelettel végezhető, azaz a modell egy megadott oszlopát tudjuk a vezérlőhöz kapcsolni
  - hasonlóan megszűntethetjük az adattársítást (`removeMapping`, `clearMapping`)
  - lehetőségünk van automatikusan, vagy manuálisan menteni az adatokat (`setSubmitPolicy`), előbbi esetén a vezérlőbeli módosítások azonnal megjelennek a modellben

Összetett adatkezelés
Adattársítás vezérlőkhöz

- Mivel a vezérlők csak egy adatot tudnak megjeleníteni, ezért a modellben navigálnunk kell, hogy hányadik sor megfelelő oszlopát jelenítjük meg, illetve szerkesztjük, ehhez a `QDataWidgetMapper` biztosít lépési műveleteket:
  - első (`toFirst`), utolsó (`toLast`), előző (`toPrevious`), következő (`toNext`) sorra való lépés
  - adott sorra lépés index (`setCurrentModelIndex(<index>)`), illetve sorszám (`setCurrentIndex(<sorszám>)`) alapján
  - sor váltásakor esemény is kiváltódik, amelytől megkapjuk a kiválasztott sor számát (`currentIndexChanged(<sorszám>)`)

Összetett adatkezelés
Adattársítás vezérlőkhöz

- Pl.:

```
QLineEdit lineEdit; // sorszerkesztő
QSqlTableModel model;
model.setTable("myTable"); // tábla beállítása

QDataWidgetMapper mapper;
mapper.setModel(&model); // modell beállítása
mapper.addMapping(&lineEdit, 1);
   // adattársítás a második oszlopra
mapper.toFirst(); // az első sorra lépünk

// a sorszerkesztőben megjelenik az első sor
// második oszlopa, amelyet szerkesztve az módosul
// a modellben
```

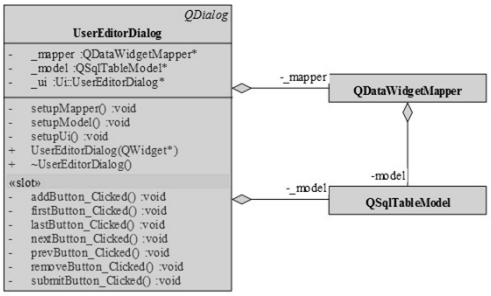
Összetett adatkezelés
Példa

Feladat: Készítsük el a felhasználók tábla szerkesztését vezérlők segítségével.

- felveszünk négy szerkesztőmezőt az adatok kezelésére, és adattársításokat hozunk létre a megfelelő oszlopokra
- gombok segítségével oldjuk meg a navigációt (elsőre, utolsóra, előzőre, következőre), valamint a speciális tevékenységeket (beszúrás, törlés, mentés, visszavonás)
- beszúráskor és törléskor mindig az aktuálisan betöltött sort kezeljünk, amit a `currentIndex()` tulajdonságon keresztül érhetünk el
- törléskor ügyelnünk kell, hogy a tábla ne legyen üres

Összetett adatkezelés
Példa

Tervezés:



Összetett adatkezelés
Példa

Megoldás (`usereditordialog.cpp`):

```
void UserEditorDialog::setupMapper()
{
   _mapper = new QDataWidgetMapper(this);
   _mapper->setModel(_model);
      // modell beállítása
   _mapper->addMapping(_ui->userIdLineEdit, 0);
      // adattársításokat létrehozása
   _mapper->addMapping(_ui->nameLineEdit, 1);
   _mapper->addMapping(_ui->passwordLineEdit, 2);
   _mapper->addMapping(_ui->levelSpinBox, 3);

   _mapper->toFirst(); // első sor beállítása
}
```


Összetett adatkezelés
Példa

Megoldás (`usereditordialog.cpp`):

```
void UserEditorDialog::addButton_Clicked()
{
   int row = _mapper->currentIndex();
      // lekérjük a betöltött sor számát
   _model->insertRow(row);
      // ennek helyére beszúrjuk az újat
   _mapper->setCurrentIndex(row);
      // megjelenítjük a beszúrt (üres) sort
}
```


Összetett adatkezelés
Vezérlők és nézet közös használata

- Lehetőségünk van nézet és adattársítás egyszerre történő használatára
  - amennyiben az adatok módosítását vezérlőkkel végezzük, a nézetben kikapcsolhatjuk a szerkesztést a `setEditTriggers(<viselkedés>)` metódussal
  - célszerű szinkronban tartani a vezérlők által megjelenített, illetve a nézetben kijelölt sort
    - ehhez a megjelenítőhöz egy szelekciós modellt (`QItemSelectionModel`) kell kapcsolnunk (`setSelectionModel`)
    - amely felügyeli a nézetben történő kijelölés változását (`currentRowChanged`)

Összetett adatkezelés
Vezérlők és nézet közös használata

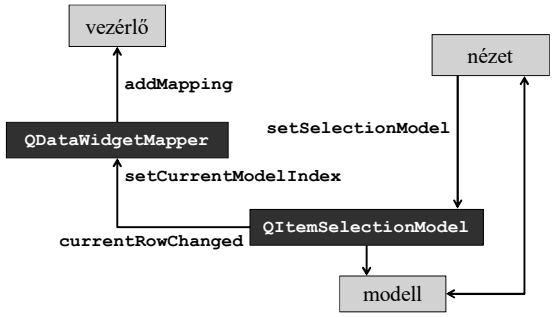
- Pl.:

```
QTableView view; // nézet
view.setModel(&model);

QItemSelectionModel selectionModel(model);
   // szelekciós modell a kijelölés követésére
view->setSelectionModel(&model);
   // a szelekciós modellt beállítjuk a
   // megjelenítőnek, innentől automatikusan össze
   // lesz kötve a modellel is
connect(model, SIGNAL(currentRowChanged(…)),
        mapper, SLOT(setCurrentModelIndex(…)));
   // kezeljük a modellbeli indexváltozásokat
```

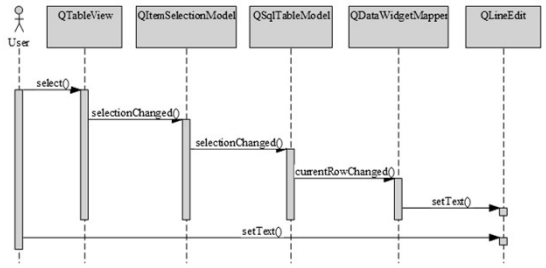
Összetett adatkezelés

Vezérlők és nézet közös használata



Összetett adatkezelés

Vezérlők és nézet közös használata



Összetett adatkezelés

Példa

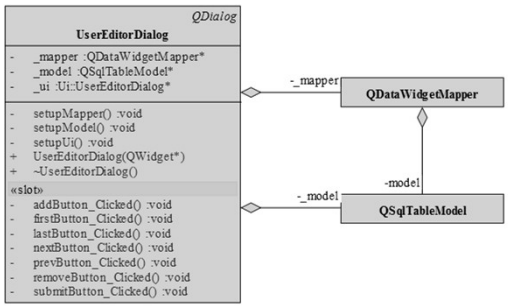
Feladat: Módosítsuk az előző alkalmazást úgy, hogy megjelenjen a teljes felhasználók tábla, de azzal ne lehessen szerkeszteni.

- a nézetet csak sorkijelölésre használjuk, ezért kikapcsoljuk a szerkeszthetőséget, valamint beállítjuk a kijelölés módját teljes sorra (`QAbstractItemView::SelectRows`)
- a táblában való navigációhoz továbbra is használjunk gombokat, amelyek állítják az adattársító aktuális sorát, illetve a nézetben is módosítja a kijelölt sort
- ugyanakkor a nézetben a sorkijelölés váltását szelekciós modell sorváltó eseménye (`currentRowChanged`) segítségével követjük

Összetett adatkezelés

Példa

Tervezés:



Összetett adatkezelés

Példa

Megoldás (`usereditordialog.cpp`):

```
void UserEditorDialog::firstButton_Clicked() {
   _mapper->toFirst(); // vezérlők beállítása
   _ui->tableView->setCurrentIndex(
      _model->index(0, 0));
      // táblamegjelenítő beállítása
}

void UserEditorDialog::setupModel() {
   …
   _ui->tableView->setSelectionBehavior(
         QAbstractItemView::SelectRows);
      // teljes sor kijelölése
```


Összetett adatkezelés

Példa

Megoldás (`usereditordialog.cpp`):

```
   _ui->tableView->setEditTriggers(
         QAbstractItemView::NoEditTriggers);
      // a táblaszerkesztés kikapcsolása a
      // megjelenítőn
   …
   QItemSelectionModel* model = new
         QItemSelectionModel(_model);
   _ui->tableView->setSelectionModel(model);
      // szelekciós modellt beállítjuk a
      // megjelenítőnek
}
```

Összetett adatkezelés

Tulajdonságok kezelése adattársítás során

- Adattársításkor a vezérlő alapértelmezett tulajdonságát kezeljük (olvassuk, illetve írjuk), ám lehetőségünk van tetszőleges tulajdonság kezelésére az **`addMapping(<vezérlő>, <oszlopszám>, <tulajdonság>)`** metódussal
- Pl.:

```
QComboBox comboBox; // legördülő menü
… // legördülő menü feltöltése
mapper.addMapping(&comboBox, 2, "currentIndex");
   // adattársítás a harmadik oszlopra, a
   // legördülő menü aktuális indexére (így a
   // tényleges adat helyett a sorszámmal
   // operálunk)
```


Összetett adatkezelés

Egyedi tulajdonságok

- A tulajdonságok olyan attribútumok, amelyekhez kívülről is hozzáférhetünk lekérdező és beállító műveletek segítségével
  - lehetőségünk van saját tulajdonságok létrehozására a **`Q_PROPERTY`** makró segítségével
  - a tulajdonságnak megadjuk típusát és nevét, valamint az író (**`WRITE`**) és olvasó (**`READ`**) műveleteket
- Pl.:

```
class Demo : public Qobject {
   Q_PROPERTY(int prop READ prop WRITE setProp)
   …
   int prop(); // olvasó művelet
   void setProp(int value); // író művelet
```


Összetett adatkezelés

Példa

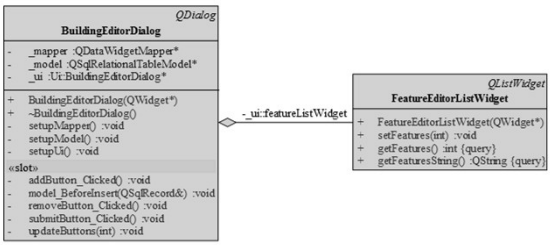
Feladat: Készítsük el az épületek tábla karbantartását csak vezérlők segítségével.

- használjunk egyszerű relációs modellt a táblakezelésre
- a város kiválasztását legördülő menüvel kell megadnunk, amelyet feltöltünk a városok tábla modelljével, továbbá meg kell adnunk a társításhoz a sorszámot (**`currentIndex`**), mint tulajdonságot
- a jellemzőket az egyedi vezérlőkkel (**`FeatureEditorListWidget`**) tudjuk szerkeszteni, ám ahhoz, hogy adattársítást tudjunk végezni, meg kell valósítanunk egy egyedi tulajdonságot a jellemzőkre (**`int features`**)

Összetett adatkezelés

Példa

Tervezés:



Összetett adatkezelés

Példa

Megoldás (`buildingtabledialog.cpp`):

```
void BuildingTableDialog::initMapper(){
   …
   _mapper = new QDataWidgetMapper(this);
      // vezérlő adattársítása
   _mapper->setModel(_model);
   …
   _mapper->addMapping(
      ui->shoreTypeComboBox, 5, "currentIndex");
      // társított tulajdonság
   _mapper->addMapping(
      ui->featureListWidget, 6, "features");
   …
}
```


Összetett adatkezelés

Példa

Megoldás (`featureeditorlistwidget.hpp`):

```
class FeatureEditorListWidget : public QListWidget
   // egyedi szerkesztő a jellemzőkre, egy
   // speciális listamegjelenítő
{
   Q_OBJECT
   Q_PROPERTY(int features READ getFeatures
                           WRITE setFeatures)
   // tulajdonságkezelés hozzárendelése
public:
   FeatureEditorListWidget(QWidget *parent = 0);
   …
}
```

Összetett adatkezelés
Egyszerű szűrések

- A vezérlők arra is használhatók, hogy szűrjük a megjelenített tábla tartalmát
  - modellek szűrése legkönnyebben a modell **`setFilter(<szűrőfeltétel>)`** műveletével végezhető el, amely lényegében a tábla tartalmának lekérdezésekor megfogalmazott **`where`** feltételt szabályozza, ezért a megszokott SQL szintaxist alkalmazhatjuk
  - a szűrés a kiválasztás (**`select`**) végrehajtásakor módosul, vagy amikor beállítottuk a szűrőt
  - pl.:

```
model.setTable("myTable");
model.setFilter("name = ''"); // szűrés
```

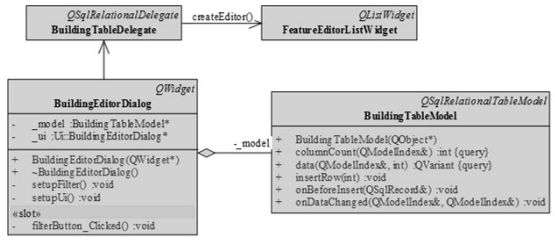

Összetett adatkezelés
Példa

Feladat: Készítsük el az épületek tábla szűrését a név, illetve a város alapján.

- a várost válasszuk ki egy legördülő menüből, amely megjeleníti a „mind” feliratot, valamint a városok neveit
- a név szűrésénél lehessen részszöveg megadni
- a szűrőt egy gombbal lehessen be, illetve kikapcsolni, bekapcsolt állapotban ne lehessen módosítani a szűrő feltételeket, és ne lehessen szerkeszteni a táblát

Összetett adatkezelés
Példa

Tervezés:



Összetett adatkezelés
Példa

Megoldás (`buildingtabledialog.cpp`*):*

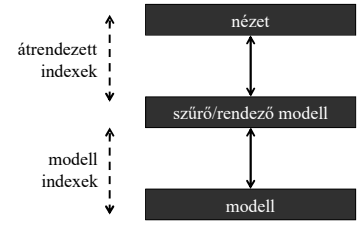
```
void BuildingTableDialog::filterButton_Clicked() {
    ...
    QString filterString = "name LIKE '%" +
       ui->nameLineEdit->text() + "%'";
    if (_ui->cityComboBox->currentText()
        != "mind"){
       QSqlQuery query;
       ...
       if (query.next())
          filterString += " AND city_id = " +
             query.value(0).toString();
       buildingTableModel->setFilter(filterString);
    ...
```


Összetett adatkezelés
Szűrő modellek

- A **`setFilter`** művelet csak az adatbázisban tárolt adatok szűrését teszi lehetővé, számított adatokat nem szűrhetünk
- Amennyiben speciálisabb szűrést szeretnénk végrehajtani, használhatjuk a dedikáltan szűrésre, illetve rendezésre készült **`QSortFilterProxyModel`** típust
  - egy olyan modell, amely ténylegesen nem tárolja az adatokat, de képes egy másik modell számára az indexeket átrendezni és megszűrni
  - az eredeti modellt a **`setSourceModel(<modell>)`** metódussal adhatjuk át
  - lehetőséget ad tetszőleges modellbeli tartalom szűrésére

Összetett adatkezelés
Szűrő modellek

- A nézet és a tényleges modell közé helyezzük, így megfelelően továbbítja az indexeket a két irányba
  - a nézet az átrendezett, szűrt indexeket kapja meg



Összetett adatkezelés
Szűrő modellek

- A szűrő/rendező modell különböző szűrőfeltételek tud fogadni:
  - `setFilterFixedString(<szöveg>)`: azonos tartalomra
  - `setFilterRegExp(<kifejezés>)`: reguláris kifejezésre
  - `setFilterWildcard(<minta>)`: adott mintára
- A szűrést különböző tényezők befolyásolhatják:
  - a megszűrendő oszlop (`setFilterKeyColumn`)
  - kis-/nagybetű kezelés (`setFilterCaseSensitivity`)
  - megjelenítési szerep (`setFilterRole`)
  - azonnali frissítés, ha a csatolt modell tartalma változik (`setDynamicSortFilter`)

Összetett adatkezelés
Szűrő modellek

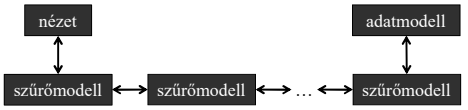
- Pl.:

```
QSortFilterProxyModel filterModel;
   // szűrőmodell létrehozása
filterModel.setSourceModel(&model);
   // beállítjuk, melyik modellt szűri meg
filterModel.setFilterKeyColumn(1);
   // beállítjuk a szűrés céloszlopát
filterModel.setFilterFixedString("");
   // az üres szövegre szűrünk

view.setModel(&filterModel);
   // a nézetnek a szűrőmodellt állítjuk be, így
   // eleve megszűrt tartalmat fog megjeleníteni
```


Összetett adatkezelés
Szűrő modellek

- A rendezést is hasonló paraméterek alapján végezhetjük el adott oszlopra
- Mivel teljes értékű modell, ezért a megszokott függvények (`data`, `columnCount`, `index`) is használhatóak
- Amennyiben több oszlopra is szeretnénk szűrést végrehajtani, célszerű a modelleket egymásra építeni, így az eredeti adat több feltételen is keresztül tud menni



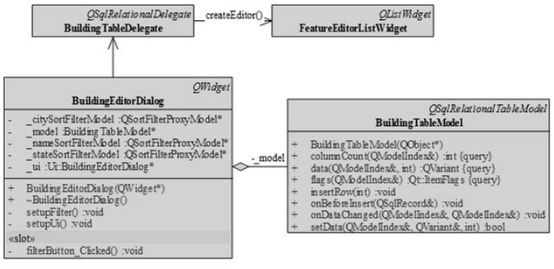
Összetett adatkezelés
Példa

Feladat: Szűrjük az épületek táblát a korábbiak mellett az állapotra is.

- használjunk szűrőmodellt a szűrés végrehajtására, mivel több feltételünk is, ezért egymásba ágyazva, de ügyeljünk arra, hogy nem feltételünk alkalmazzuk egyszerre valamennyi szűrőt, ezért elágazásokkal léptetjük a szűrmodellek egymásba ágyazását
- a szűrés törlésekor csak vissza kell állítanunk az eredeti modellt a táblamegjelenítőnek
- az állapotot legördülő menü segítségével választjuk ki, amit manuálisan töltünk fel, és a szűrőt az index alapján állítjuk be

Összetett adatkezelés
Példa

Tervezés:



Összetett adatkezelés
Példa

Megoldás (`buildingtabledialog.cpp`):

```
void BuildingTableDialog::filterButton_Clicked(){
   ...
   QAbstractItemModel *currentModel =
      _model;
      // léptetjük az aktuális modellt a
      // megszűrtek között
   if (_ui->nameLineEdit->text() != "") {
     // ha van szöveg a szövegdobozban
     _nameSortFilterModel->
        setSourceModel(currentModel);
     // forrás tábla
     nameSortFilterModel->setFilterKeyColumn(1);
        // szűrés oszlopa
```

Összetett adatkezelés
Példa

Megoldás (`buildingtabledialog.cpp`):

```
            _nameSortFilterModel->
               setFilterCaseSensitivity(
               Qt::CaseInsensitive);
            _nameSortFilterModel->setFilterRegExp(
               _ui->nameLineEdit->text());
                  // szűrés reguláris kifejezéssel
            currentModel = _nameSortFilterModel;
        }
        …
        _ui->tableView->setModel(currentModel);
        // szűrések után kialakult modell beállítása
    …
}
```


Összetett adatkezelés
Egyedi szűrések

- Amennyiben speciálisabb szűréseket szeretnénk végezni, a `QSortFilterProxyModel` osztályból származtathatunk egyedi megoldásokat
  - új szűrésmód bevezetéséhez sorok kezelésére a `filterAcceptRow(<oszlop>, <szülő index>)`, oszlopok kezelésére a `filterAcceptColumn(<oszlop>, <szülő index>)` műveleteket definiálhatjuk felül
  - új rendezéshez a `lessThan(<bal index>, <jobb index>)` műveletet kell felüldefiniálnunk
  - a szűrés/rendezés alkalmazásához használhatjuk az `invalidate`, és `invalidateFilter` metódusokat

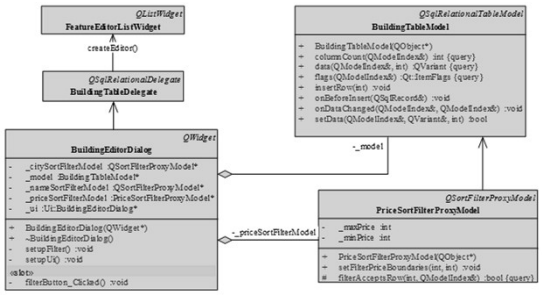
Összetett adatkezelés
Példa

Feladat: Egészítsük ki az épületek szűrését a minimum/maximum árhatár meghatározásával, és csak azokat az épületeket jelenítsük meg, amelyek szélső árai a megadott tartományba esnek.

- az árakat két `QSpinBox` vezérlő segítségével módosítjuk
- definiálnunk kell egy egyedi szűrő modellt az árkezelésre, amiben felüldefiniáljuk a sorokra vonatkozó szűrőfeltételt, illetve kiegészítjük egy, az árakat beállító, és a szűrést végrehajtó metódussal
- az árkezeléshez pusztán lekérdezzük a sornak megfelelő adatokat a szűrőmodellben

Összetett adatkezelés
Példa

Tervezés:



Összetett adatkezelés
Példa

Megoldás (`pricesortfilterproxymodel.hpp`):

```
class PriceSortFilterProxyModel :
   public QSortFilterProxyModel {
   …
public slots:
  void setFilterPriceBoundaries(int min, int max);
protected:
   bool filterAcceptsRow(int sourceRow, const
         QModelIndex &sourceParent) const;
      // sorok elfogadásának felüldefiniálása
private:
    int _minPrice;
    int _maxPrice;
};
```


Összetett adatkezelés
Példa

Megoldás (`pricesortfilterproxymodel.cpp`):

```
void PriceSortFilterProxyModel::
       setFilterPriceBoundaries(int min, int max) {
   _minPrice = min; _maxPrice = max;
   invalidateFilter(); // szűrő újra alkalmazása
}
bool PriceSortFilterProxyModel::filterAcceptsRow(
       int sourceRow,
       const QModelIndex &sourceParent) const {
   int min = sourceModel()->index(sourceRow, 10,
             sourceParent).data().toInt();
   … // lekérdezzük az ár oszlopokat
   return (_minPrice <= min && max <= _maxPrice);
}
```

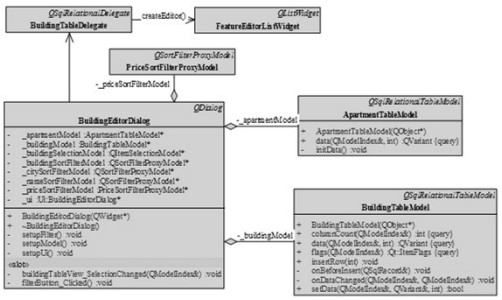
Összetett adatkezelés
Példa

Feladat: Jelenítsük meg az apartman táblát is a felületen, és legyen lehetőség épület kijelölése alapján szűrni az apartmanokat.

- felveszünk egy új egyedi modellt, valamint egy szűrőmodellt az apartman táblára, a foglalhatóságot kijelölő mezővel jelezzük (a felújítás helyett)
- a kijelölés kezelésére felveszünk egy szelekciós modellt az épületekre, ahol sorkijelölés váltásának eseményére váltjuk az apartman tábla szűrését
- mivel az épületek modelljét is szűrjük, szűréskor újra létre kell hozni, és be kell állítani a szelekciós modellt (mivel az csak egy meghatározott modellre állítható rá)

Összetett adatkezelés
Példa

Tervezés:



Összetett adatkezelés
Példa

Megoldás (`buildingeditordialog.hpp`):

```
void BuildingEditorDialog::filterButton_Clicked()
{
   …
   // a szelekciós modellt át kell állítani a
   // táblamegjelenítőben beállított modellre
  disconnect(_buildingSelectionModel,
     SIGNAL(currentRowChanged(QModelIndex,
     QModelIndex)), this,
     SLOT(buildingTableView_SelectionChanged(
     QModelIndex)));

   delete _buildingSelectionModel;
```


Összetett adatkezelés
Példa

Megoldás (`buildingeditordialog.hpp`):

```
    _buildingSelectionModel =
       new QItemSelectionModel(
          _ui->buildingTableView->model());

    _ui->buildingTableView->setSelectionModel(
       _buildingSelectionModel);

    connect(_buildingSelectionModel, SIGNAL(
       currentRowChanged(QModelIndex,
       QModelIndex)), this,
       SLOT(buildingTableView_SelectionChanged(
       QModelIndex)));
}
```