Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások fejlesztése I

11. előadás

Adatkezelés speciális eszközökkel

Giachetta Roberto

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

Adatkezelés speciális eszközökkel
Az adatkezelés lehetőségei

- Adatbázis tartalom 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

Adatkezelés speciális eszközökkel
Változások követése

- Az adatmodellek lehetőségek 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

Adatkezelés speciális eszközökkel
Változások követése

- A változáskövetést célszerű manuális szerkesztési stratégiával használni, mivel így a változtatások visszavonhatóak (`revertAll()`) mentés előtt
- Pl.:

```
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
}
```

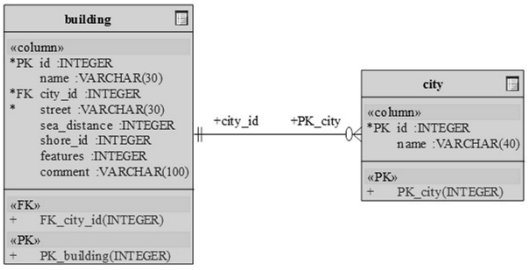

Adatkezelés speciális eszközökkel
Példa

Feladat: Módosítsuk az épületek szerkesztését úgy, hogy ellenőrizze a tengerpart távolság értékét (csak pozitív szám lehet), és mentéskor minden új sorhoz szúrja be az „új hirdetés” megjegyzést. Ezen felül lehessen a városokat is hozzáadni, törölni (feltéve, hogy a városban nincs épület).

- az ellenőrzést és az automatikus beírást a modell `beforeInsert` és `DataChanged` eseményeivel kezeljük
- a városokat külön dialógusablakban (`CityEditorDialog`) szerkeszthetjük (mivel csak a nevüket kell, elég egy `ListView` nézet) úgy, hogy a modellt az épületek modell relációjából olvassuk ki (`relationModel(2)`)
- az épület azonosítóját rejtsük el (úgyis generálódik)

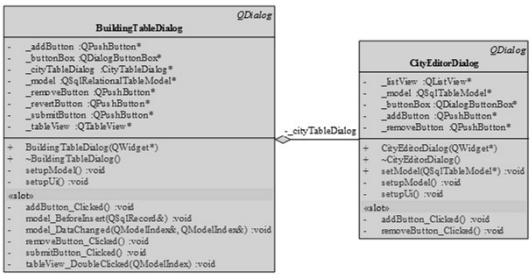
Adatkezelés speciális eszközökkel
Példa

Tervezés (adatbázis):



Adatkezelés speciális eszközökkel
Példa

Tervezés (alkalmazás):



Adatkezelés speciális eszközökkel
Példa

Megvalósítás (`buildingeditordialog.cpp`):

```
…
connect(_model, SIGNAL(beforeInsert(QSqlRecord&)),
   this, SLOT(model_BeforeInsert(QSqlRecord&)));
   // beszúrás előtti esemény társítása
connect(_model, SIGNAL(beforeUpdate(int,
   QSqlRecord&)), this,
   SLOT(model_BeforeUpdate(int, QSqlRecord&)));
   // adatváltoztatás esemény társítása
…
tableView->setColumnHidden(0, true);
// első oszlop elrejtése
…
```

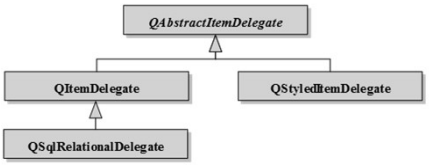

Adatkezelés speciális eszközökkel
Példa

Megvalósítás (`buildingeditordialog.cpp`):

```
void BuildingEditorDialog::
      model_BeforeInsert(QSqlRecord& record){
   if (record.value("comment").isNull())
      // ha a komment oszlop üres
      record.setValue("comment",
                      trUtf8("új hirdetés"));
}
void BuildingEditorDialog::
      tableView_DoubleClicked(QModelIndex index){
   if (index.isValid() && index.column() == 2)
      _cityEditorDialog->show();
   // megjelenítjük a városszerkesztőt
}
```


Adatkezelés speciális eszközökkel
Speciális adatmegjelenítés

- A modell és a megjelenítő közötti kapcsolatot a *delegált* (`QAbstractItemDelegate` leszármazott) osztályok biztosítják, amelyek szabályozzák a megjelenítés módját
  - az alap megjelenítést a `QItemDelegate`, a relációk kezelését a `QSqlRelationalDelegate`, egyedi megjelenítést pedig a `QStyledItemDelegate` szolgáltatja



Adatkezelés speciális eszközökkel
Egyedi delegáltak létrehozása

- Lehetőségünk van bármely osztályból történő származtatással további speciális megjelenítési módok definiálásra
  - a delegált `paint(…)` művelete szolgál a kirajzolás végrehajtására, ezt kell felüldefiniálnunk
  - paraméterben megkapja a kirajzoló objektumot (`QPainter`), a kirajzolási stílust (`QStyleOptionViewItem`), valamint a kirajzolandó adatot (`QModelIndex`)
  - a stílusban megfogalmazhatunk különböző módokat (tagolás, igazítás), illetve méretet
  - a `paint` műveletben a `drawDisplay` művelettel tudjuk a felületet rajzolni, a `drawFocus` művelettel pedig rárajzolni a fókuszt

Adatkezelés speciális eszközökkel
Egyedi delegáltak létrehozása

- Pl.:

```
class MyDelegate : public QItemDelegate {
   …
   void paint(QPainter *painter, …, const
              QModelIndex &index) const {
      if (index.column() == 2) // kettes oszlopra
      {
         … // speciális rajzolást végzünk
         drawDisplay(…); // adat kirajzolása
         drawFocus(…); // fókusz kirajzolása
      } else { // a többire az alapértelmezettet
          QItemDelegate::paint(…);
      }
   …
```

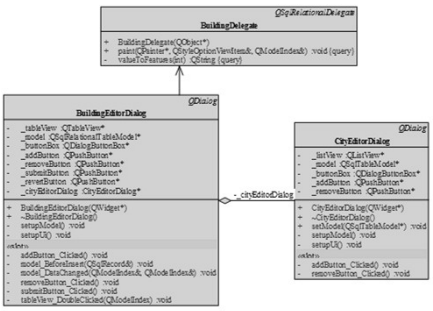
Adatkezelés speciális eszközökkel
Példa

Feladat: Módosítsuk az épületek kezelését úgy, hogy a tengerpart távolságnál a szám után a mértékegységet (méter) is odaírjuk, illetve jelenítsük meg a jellemzőket szám helyett szövegesen

- ehhez egy egyedi delegált osztályt (**`BuildingDelegate`**) származtatunk a **`QSqlRelationalDelegate`**-ből, és felüldefiniáljuk a **`paint`** műveletet
- a távolságnak csak hozzá kell vennünk az „ m” szöveget a számhoz, illetve 1 esetén azt írjuk ki, hogy „közvetlen”
- a jellemzőknél a számérték alapján fűzzük össze a megjelenítendő szöveget (ehhez bitenként kell feldolgoznunk a számot)

Adatkezelés speciális eszközökkel
Példa

Tervezés:



Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingdelegate.cpp`):

```
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();
          // adat lekérdezése
          if (shoreDistance == 1)
             text = "közvetlen";
          else
             text = QString::number(shoreDistance)
                    + " m";
```


Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingdelegate.cpp`):

```
          QStyleOptionViewItem optionViewItem =
              option; // kiírás módjának beállítása
          optionViewItem.displayAlignment =
             Qt::AlignRight | Qt::AlignVCenter;
             // jobbra és középre tagolt
          drawDisplay(painter, optionViewItem,
             optionViewItem.rect, text);
             // adat kirajzolása
          drawFocus(painter, optionViewItem,
             optionViewItem.rect);
             // fókusz kirajzolása
          break;
    ...
```


Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingdelegate.cpp`):

```
       default: // alapértelmezett rajzolás
          QItemDelegate::paint(painter, option,
             index);
          break;
       ...

QString BuildingDelegate::valueToFeatures(int
      value) const {
   QString result;
   if (value % 2 == 1) result += trUtf8("főút, ");
    if ((value >> 1) % 2 == 1)
        result += trUtf8("parti szolgálat, ");
   ...
```


Adatkezelés speciális eszközökkel
Egyedi szerkesztőmezők

- Az egyedi delegált segítségével nem csak a kiírást, de a szerkesztés módját is változtathatjuk
  - az alapértelmezett szerkesztőmezőt tetszőlegesre cserélhetjük
  - a **`createEditor(…)`** művelet felelős a szerkesztőmező létrehozásáért, ebben visszaadhatunk egy tetszőleges **`QWidget`** leszármazottat, vagyis bármilyen vezérlőt behelyezhetünk szerkesztőnek
  - a **`setEditorData(…)`** felelős a szerkesztőmező kitöltéséért, hogy az megfeleljen a modellbeli adatnak
  - a **`setModelData(…)`** felelős a szerkesztőmezőben történt módosítás visszaírásáért a modellbe

Adatkezelés speciális eszközökkel

Példa

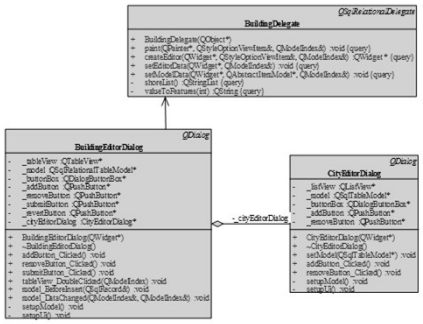
Feladat: Módosítsuk az épületek kezelését úgy, hogy a tengerpart típust egy legördülő menü segítségével lehessen kijelölni.

- tengerpart típusok: homokos (0), sziklás (1), kavicsos (2), apró kavicsos (3)
- módosítjuk a **`BuildingDelegate`** osztályt az egyedi vezérlővel, amely **`QComboBox`** típusú lesz, ennek elemeit egy konstans lista (**`shoreList`**) segítségével töltjük fel, amely tartalmazza a parttípusokat
- 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

Adatkezelés speciális eszközökkel

Példa

Tervezés:



Adatkezelés speciális eszközökkel

Példa

Megoldás (`buildingdelegate.cpp`):

```
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->addItems(shoreList());
      // felvesszük a lista által tartalmazott
      // elemeket
      return shoreComboBox;
   ...
```


Adatkezelés speciális eszközökkel

Példa

Megoldás (`buildingdelegate.cpp`):

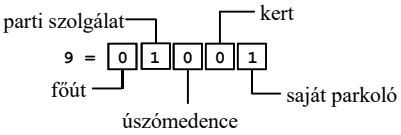
```
void BuildingDelegate::setEditorData(Qwidget
     *editor, const QModelIndex &index) const
{
   if (index.column() == 5)
   {
      int i = index.data().toInt();
      QComboBox *shoreComboBox =
         qobject_cast<QComboBox*>(editor);
      // konverzió szükséges
      shoreComboBox->setCurrentIndex(i);
      // szerkesztőmező elemének beállítása
   }
   ...
```


Adatkezelés speciális eszközökkel

Példa

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

- a jellemzők bináris formában tárolják az épület tulajdonságait (kert, parkoló, …) annak érdekében, hogy a későbbiekben könnyen fel tudjunk venni új tulajdonságokat a táblaszerkezet módosítása nélkül
- pl.:



Adatkezelés speciális eszközökkel

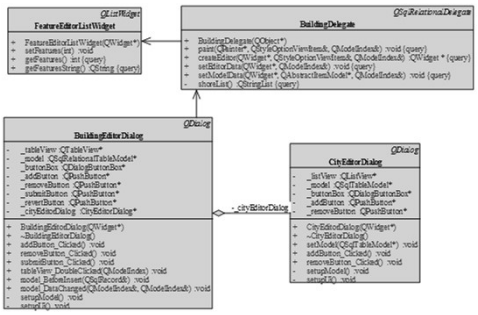
Példa

Tervezés:

- mivel az adatbázisban továbbra is a szám lesz eltárolva, szükségünk lesz egy egyedi lista vezérlőre (**`FeatureEditorListWidget`**), amely a szám-szöveg konverziót (**`setFeatures`**, **`getFeatures`**), illetve a szöveges formában történő kiírást (**`getFeaturesString`**) elvégzi, és listaszerűen jeleníti meg az adatokat
- ehhez a **`QListWidget`** vezérlőből származtatunk, amelyben lehetőség van az elemek kijelölésére, így közvetlenül tárolhatjuk a jellemzők állapotát
- az egyedi vezérlőnket a delegált (**`BuildingDelegate`**) segítségéhez helyezzük a szerkezetbe

Adatkezelés speciális eszközökkel
Példa

Tervezés:



Adatkezelés speciális eszközökkel
Példa

Megoldás (`featureeditorlistwidget.cpp`):

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


Adatkezelés speciális eszközökkel
Példa

Megoldás (`featureeditorlistwidget.cpp`):

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


Adatkezelés speciális eszközökkel
Számított adatok kezelése

- 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, amelyben a pozíció (index) alapján pontosan megállapíthatjuk a megjeleníteni kívánt adatot
    - az oszlopok számát megadó `columnCount()` metódust, ahol általában növeljük az értéket
  - mindkét műveletben tovább hívhatjuk az ősbeli örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk

Adatkezelés speciális eszközökkel
Számított adatok kezelése

- Pl.:

```
class MyTableModel : public QSqlTableModel {
   Qvariant data(const QModelIndex&, int) const;
   int columnCount() const;
};

QVariant MyTableModel::data(const QModelIndex&
     index, int role) const {
   if (index.column() == 6)
     // ha a számított oszlopban vagyunk
     … // kiszámítjuk az értéket
   else // különben az ősbeli értéket adjuk vissza
      return QSqlTableModel::data(index, role);
}
```


Adatkezelés speciális eszközökkel
Számított adatok kezelése

```
int MyTableModel::columnCount() const {
   return QSqlTableModel::columnCount() + 1;
   // egy oszloppal bővítettük a táblát
}
```

- A `data` metódusban szerep (`role`) paraméter állapítja meg, milyen információ lekérdezése kapcsán hívták meg a műveletet, a leggyakoribb szerepek:
  - `Qt::DisplayRole`: a modell által megjelenített érték (amelyet tovább változtathatunk a delegáltban)
  - `Qt::EditRole`: a szerkesztett érték, amely általában megegyezik a megjelenítettel (meghatározott esetekben különbözhet, pl. relációk esetén)

Adatkezelés speciális eszközökkel
Számított adatok kezelése

- `Qt::ToolTipRole`: előugró üzenet
- `Qt::CheckStateRole`: az adott elem kijelölési állapota (lehet kijelölt, kijelöletlen), ennek használatával a cella alapértelmezetten kijelölő mezőként fog megjelenni a nézetben
- `Qt::SizeHintRole`: szabályozza a megjelenítendő cella méretét, egyedi szerkesztőmezők esetén módosítható
- `Qt::TextAligmentRole`: szövegigazítás az adathoz
- `Qt::ForegroundRole`, `Qt::BackgroundRole`, `Qt::TextColorRole`, …: különböző megjelenítési beállítások, amelyek szabályozhatóak a modell szintjén, illetve a delegált szintjén is

Adatkezelés speciális eszközökkel
Számított adatok kezelése

- Pl.:

```
QVariant MyTableModel::data(…) const {
   if (index.column() == 6){
      switch (role){
         case Qt::TestAlignmentRole: // igazítás
            return QVariant(Qt::AlignLeft |
                            Qt::AlignVCenter);
         case Qt::DisplayRole:
         case Qt::EditRole: // megjelenítendő adat
            return …;
         case Qt:ToolTipRole: // előugró üzenet
            return QVariant("You cannot modify
                             this!");
         …
```

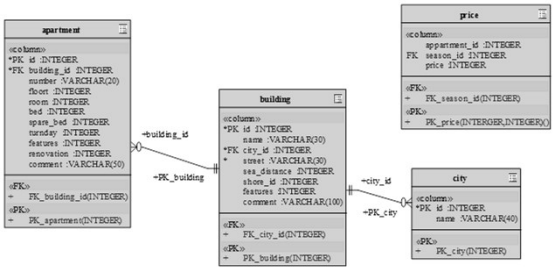

Adatkezelés speciális eszközökkel
Példa

Feladat: Egészítsük ki az épületek táblát három számított oszloppal, az épületben lévő apartmanok számával, valamint a minimális, és maximális árral.

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

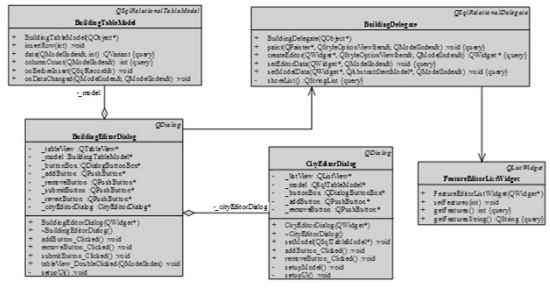
Adatkezelés speciális eszközökkel
Példa

Tervezés (adatbázis):



Adatkezelés speciális eszközökkel
Példa

Tervezés (alkalmazás):



Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingtablemodel.cpp`):

```
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 && index.column() <= 10
       && role == Qt::TextAlignmentRole )
      // szövegigazítás
      return QVariant(Qt::AlignRight |
                      Qt::AlignVCenter);
```

Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingtablemodel.cpp`):

```
if (index.column() == 8 && ( role ==
    Qt::DisplayRole || role == Qt::EditRole)) {
   // apartmanok számának számítása
   QSqlQuery query;
   query.exec("select count(*) 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(0);
 ...
```


Adatkezelés speciális eszközökkel
Számított adatok szerkesztése

- A táblamodell nem csak a számított adatok lekérdezését, de szerkesztését is lehetővé teszi
  - az adatok szerkesztését specializálhatjuk a **`setData(<index>, <érték>, <szerep>)`** műveletet felüldefiniálásával, 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
    - a számított adott oszlopnak kiválaszthatónak (**`ItemIsSelectable`**) és szerkeszthetőnek (**`ItemIsEditable`**) kell lennie

Adatkezelés speciális eszközökkel
Számított adatok szerkesztése

- Pl.:

```
class MyTableModel : public QSqlTableModel {
   …
   bool setData(const QModelIndex&, const
                QVariant&, int);
   Qt::ItemFlags flags(const QModelIndex&) const;
};

Qt::ItemFlags MyTableModel::flags(const
                QModelIndex&) const {
   …
   if (<számított oszlopban vagyunk>)
      return Qt::ItemIsEditable | …
}
```

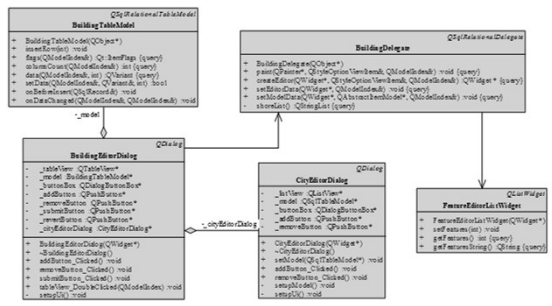

Adatkezelés speciális eszközökkel
Példa

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”. Lehessen állítani az értéket úgy, hogy normál, vagy lezárt állapotba tudjuk helyezni az épületet.

- felveszünk a számított oszlopot, amely az adatot a 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

Adatkezelés speciális eszközökkel
Példa

Tervezés:



Adatkezelés speciális eszközökkel
Példa

Megoldás (`buildingtablemodel.cpp`):

```
Qt::ItemFlags BuildingTableModel::flags(const
      QModelIndex& index) const {
   Qt::ItemFlags flag =
      QSqlTableModel::flags(index);
      // lekérdezzük az alap állapotjelzőt
   if (index.column() == 4)
      // a negyedik oszlop számított
      flag |= Qt::ItemIsSelectable |
              Qt::ItemIsEditable;
         // szerkeszthetővé tesszük

   return flag;
}
```

Adatkezelés speciális eszközökkel

Példa

Megoldás (`buildingtablemodel.cpp`*):*

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