Haladó technikák az adatmodellezésben

Adatkezelés problémái

- Adatbázistartalomnak egy 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

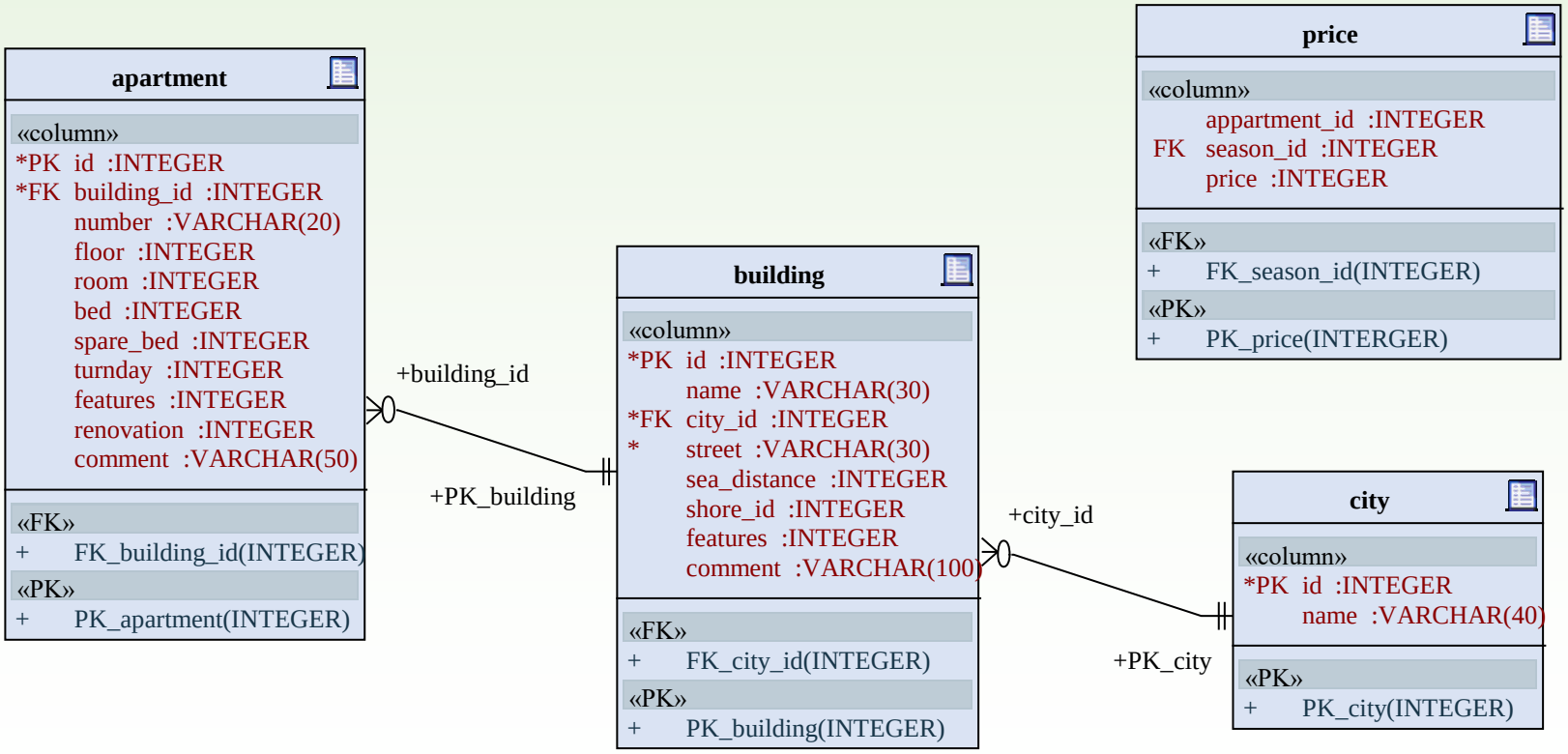
Változások követése

- Az adatmodellek lehetőséget adnak az adatokban történt változások követésére, amit felhasználhatunk ellenőrzések, vagy automatikus kitöltések végrehajtására
  - közvetlenül a változást követően a **`dataChanged(<tartomány bal felső indexe>, <jobb alsó indexe>)`** szignálokkal
  - az adatbázisba történő mentéskor (**`submit()`** és **`submitAll()`** lefutásakor) a **`beforeInsert(<rekord>)`**, **`beforeDelete(<sorszám>)`** és **`beforeUpdate(<sorszám>, <rekord>)`** szignálokkal
- A változáskövetést célszerű manuális szerkesztési stratégiával ötvözni, mivel így a változtatások visszavonhatók (**`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
}
```

Keret probléma

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

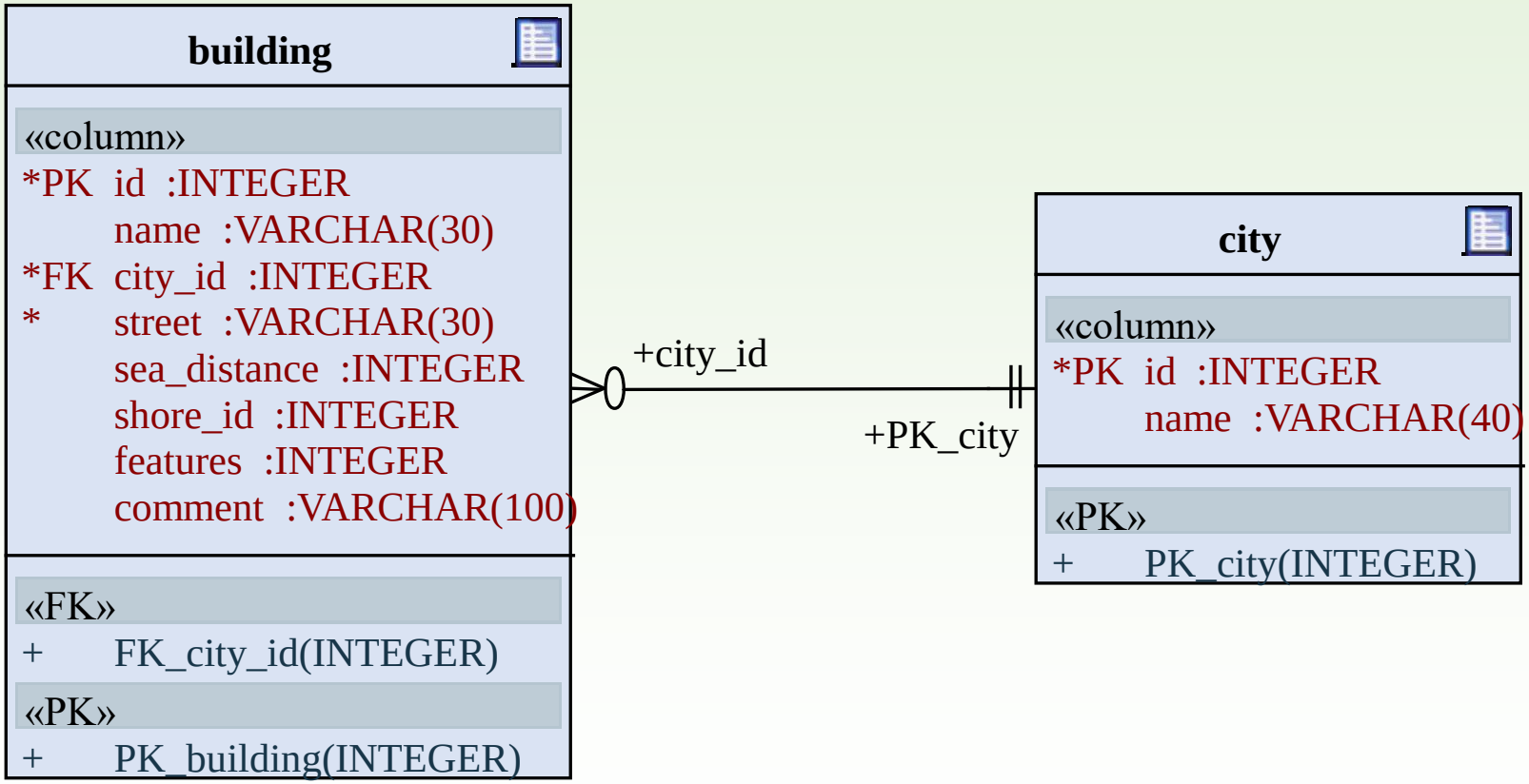


4.Feladat

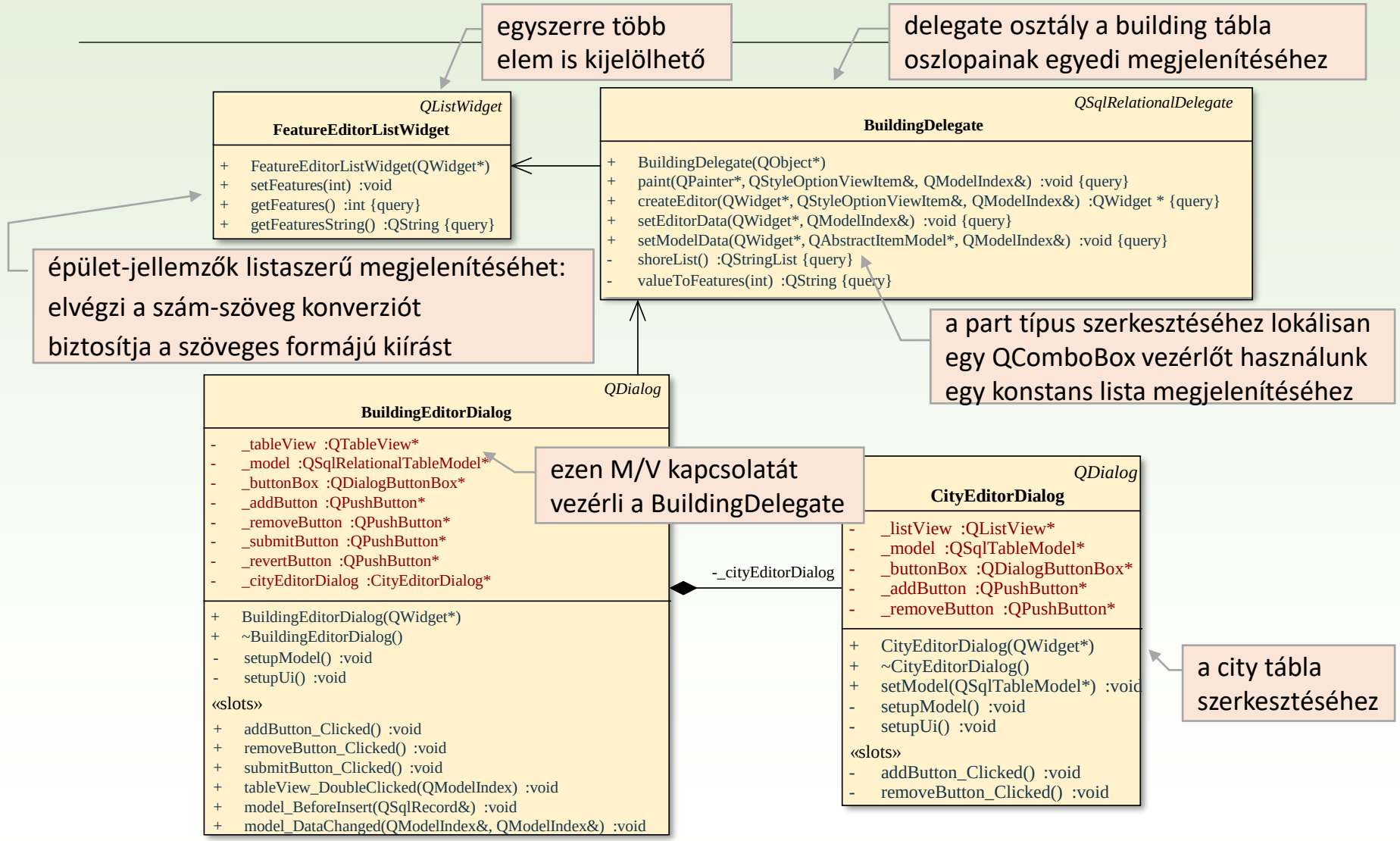
Módosítsuk az épületek kezelését úgy, hogy egy épület adatainak megjelenése minél beszédesebb legyen. (Ennek egyik részletét, hogy a település kód helyett a település neve látszódjon, az idegenkulcs kapcsolat alapján már megoldottuk.)

- A tengerparttól vett távolság (*sea_distance*) megjelenítéséhez vegyük hozzá az „m” (méter) szöveget a számhoz, illetve 10 méter alatt írjuk ki azt, hogy „közvetlenül”.
- A tengerpartot jellemző egész szám (*shore*) jelentése: homokos (0), sziklás (1), kavicsos (2), apró kavicsos (3). Szám helyett a szöveget jelenítsük meg, amelyet szerkesztéskor egy legördülő menüből választhassunk ki.
- Az épület jellemzőit (*features*) egy-egy logikai ítélet írja le amely a tulajdonság meglétét vagy hiányát mutatja. Ezen logikai ítéleteknek, (biteknek) a sorozata egy egész számot határoz meg, és ezt tároljuk. Ezen szám megjelenítése helyett a meglevő tulajdonságok szöveges kiírásával érthetőbb lenne, szerkesztéskor pedig a jellemzőket egy listából kellene kiválasztani.

4.Feladat: adatbázis



4.Feladat: tervezés



4.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 < 10)text = "next to";
        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

4.Feladat: part típusának kiírása

```
…
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(tr("sandy"));
     list.append(tr("rocky"));
     list.append(tr("pebbly"));
     list.append(tr("gritty"));
     return list;
}
```

4.Feladat: épület jellemzők kiírása

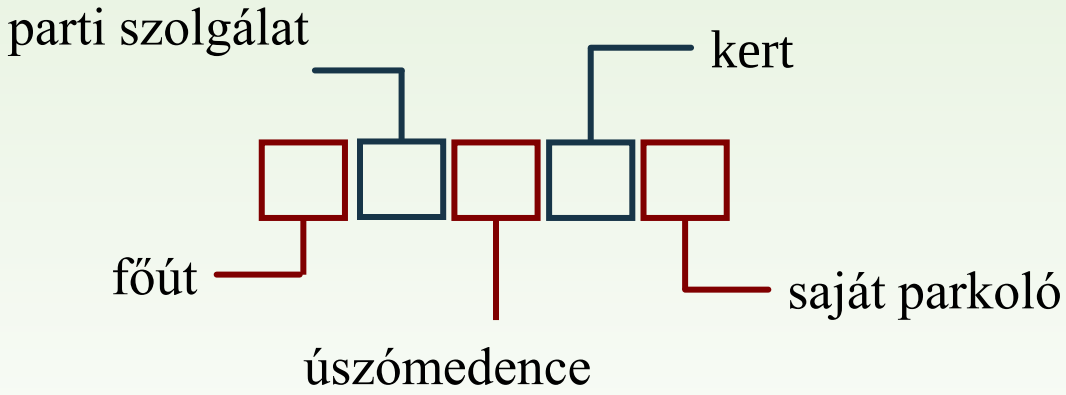
```
    …
    case 6: // jellemzők oszlop
        QString text;
        if (index.data().isNull() || index.data().toInt() == 0) {
            text = "none";
        } else {
            text = valueToFeatures(index.data().toInt());
        }
        QStyleOptionViewItem optionViewItem = option;
        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

4.Feladat: épületek jellemzői

Az épületeket több „hozzátartozik /nem tartozik hozzá” tulajdonság jellemzi.



- Az adatbázisban ezt egy olyan szám kódolja, amely annak a hat hosszú bitsorozatnak az értéke, amely a van/nincs állításokat írja le.
- Ebből a számból kell legenerálni az épületre jellemző tulajdonságok neveinek szövegszerű felsorolását, és a szerkesztésnél azt a listát, amely a választható jellemzők neveit sorolja fel, amelyből egyszerre többet is ki lehet jelölni.

4.Feladat: épület jellemzők

```
QString BuildingDelegate::valueToFeatures(int value) const
{
    QString result;
    if (value % 2 == 1)          result += tr("next to busy road, ");
    if ((value >> 1) % 2 == 1) result += tr("beach service, ");
    if ((value >> 2) % 2 == 1) result += tr("swimming pool, ");
    if ((value >> 3) % 2 == 1) result += tr("garden, ");
    if ((value >> 4) % 2 == 1) result += tr("private parking, ");
    if (result.size() > 0)       return result.left(result.size() - 2);
    else   return result;
}
```

a jellemzők lekérdezését bitenkénti eltolással oldjuk meg

4.Feladat: szerkesztés létrehozása

```
QWidget* BuildingDelegate::createEditor( Qwidget *parent,
    const QStyleOptionViewItem &option, const QModelIndex &index) const
{
    switch (index.column()) {
    case 5:
        QComboBox *shoreComboBox = new QComboBox(parent);
        shoreComboBox->addItems(shoreList());
        return shoreComboBox; break;
    case 6:
        FeatureEditorListWidget* featureEditorlistWidget =
                new FeatureEditorListWidget(parent);
        return featureEditorlistWidget;  break;
    default:
        return QSqlRelationalDelegate::createEditor(parent, option, index);
        break;
    }
}
```

tengerpart fajta kiválasztásához hozza létre

épület jellemző szerkesztéséhez hozza létre

tengertől való távolság (int) szerkesztése alapértelmezett módon történik

4.Feladat: szerkesztés megjelenítése

```
void BuildingDelegate::setEditorData(
     Qwidget *editor, const QModelIndex &index) const
{
     switch (index.column()) {
     case 5:
          int i = index.data().toInt();
          QComboBox *shoreComboBox = qobject_cast<QComboBox *>(editor);
          shoreComboBox->setCurrentIndex(i);
          break;
     case 6:
          FeatureEditorListWidget* featureEditorlistWidget =
               qobject_cast<FeatureEditorListWidget *>(editor);
          featureEditorlistWidget->setFeatures(index.data().toInt());
          break;
     default:
          QSqlRelationalDelegate::setEditorData(editor, index);
          break;
     }
}
```

megkapja a szerkesztéshez használandó widget-et

cast kell, mert alapértelmezetten nem tudja, hogy legördülő menü

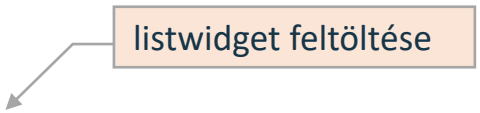
4.Feladat: szerkesztés mentése

```
void BuildingDelegate::setModelData(QWidget *editor,
    QAbstractItemModel *model, const QModelIndex &index) const
{
    switch (index.column()) {
    case 5:
        QComboBox *shoreComboBox = qobject_cast<QComboBox *>(editor);
        model->setData(index, shoreComboBox->currentIndex());
        break;
    case 6:
        FeatureEditorListWidget* featureEditorlistWidget =
            qobject_cast<FeatureEditorListWidget *>(editor);
        model->setData(index, featureEditorlistWidget->getFeatures());
        break;
    default:
        QSqlRelationalDelegate::setModelData(editor, model, index);
        break;
    }
}
```

adat visszaírása a modellbe

4.Feladat: jellemzők szerkesztése

```
FeatureEditorListWidget::FeatureEditorListWidget(QWidget *parent)
    :  QListWidget(parent)
{
    addItem(tr("next to busy road"));
    addItem(tr("beach service"));
    addItem(tr("swimming pool"));
    addItem(tr("garden"));
    addItem(tr("private parking"));
}
```



4.Feladat: jellemzők konvertálása

```
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);
    }
}
```

a listwidget azon elemeinek kijelölése, amelyekkel a modellből kapott adat (int) alapján rendelkezik az épület

```
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;
}
```

a listwidget kiválasztott épület jellemzők alapján a modellbe írt adat (int) kiszámítása

5.Feladat

Egészítsük ki az épületek táblát a végén három számított oszloppal

- az épületben lévő apartmanok számával
- az apartmanok árai közül a legkisebbel
- az apartmanok árai közül legnagyobbal,

és szúrjunk be az ötödik (4.) oszlopként egy állapot oszlopot, amely jelöli, hogy van-e tatarozás az épületben. Ez az állapot

- „normál”, ha mindegyik apartman kiadható,
- „lezárt”, ha mindegyik apartman tatarozás alatt van,
- „felújítás alatt” egyébként.

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

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 ősosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

Számított adatok szerkesztése

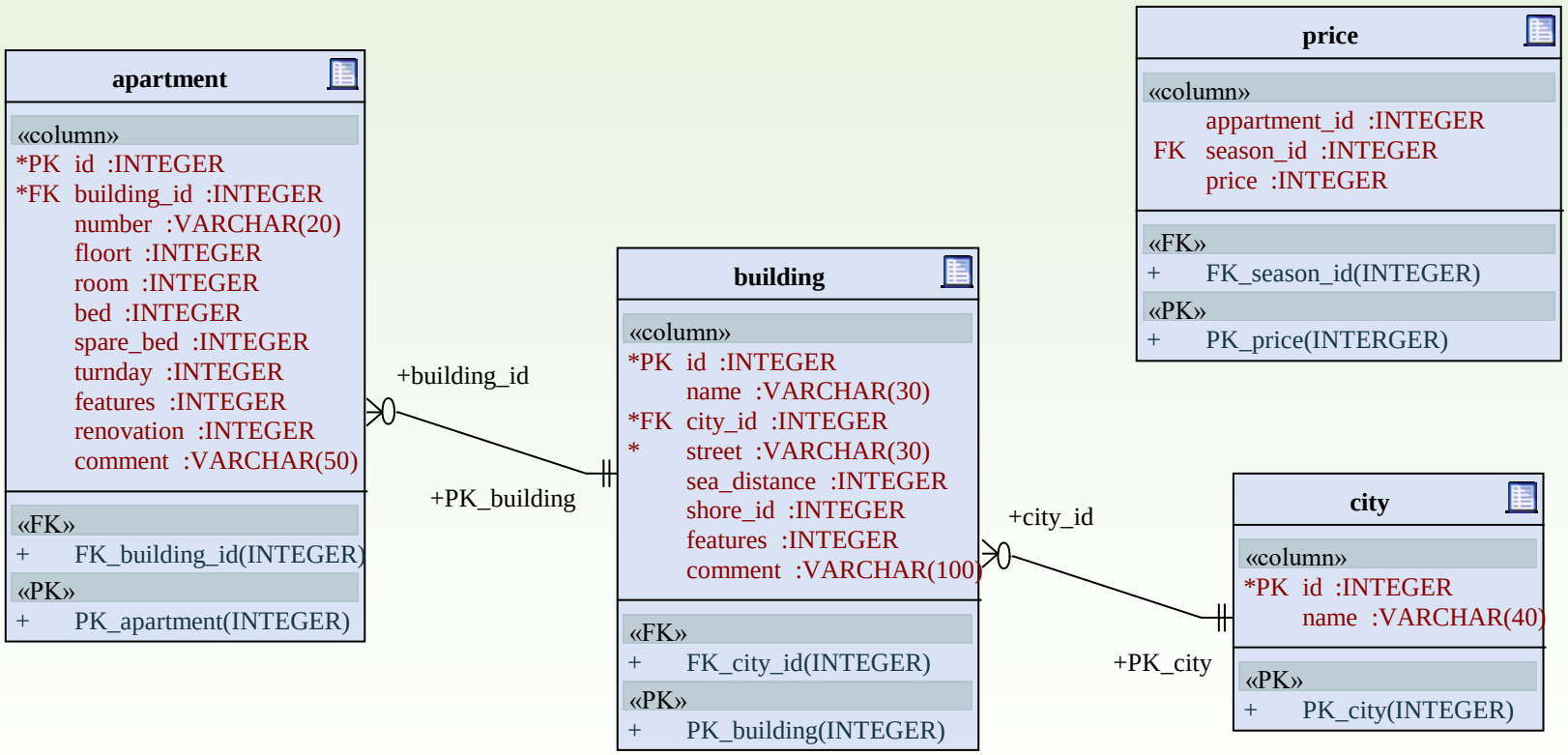
- A táblamodell nemcsak a számított adatok lekérdezését, de szerkesztését is lehetővé teszi.
  - A **`setData(<index>, <érték>, <szerep>)`** művelet felüldefiniálásával az adatok szerkesztését specializálhatjuk, amelyben megadhatjuk a számított adatok módosításának tényleges tevékenységét.
  - A számított 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: tervezés

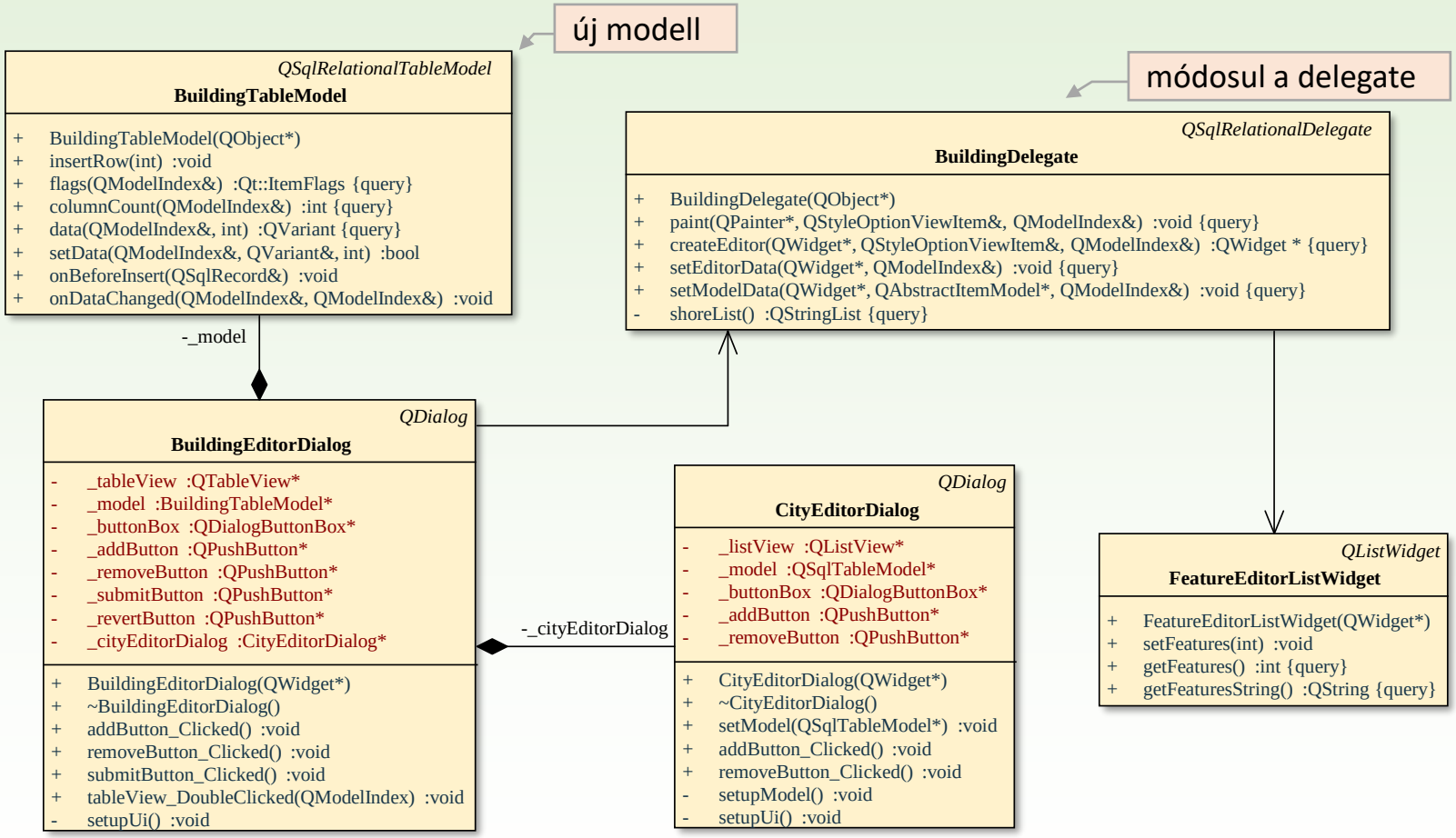
A relációs modellből egy egyedi modellt (**`BuildingTableModel`**) származtatunk, amelyben felüldefiniáljuk az adatlekérdezést, az oszlopok számát, illetve az új sor beszúrását (az alapértelmezett értékek beszúrása végett).

- A három új számított értéket megfelelő lekérdezések segítségével számoljuk ki (pl. ár esetén az **`apartment`** és a **`price`** tábla alapján).
  - A delegált osztályban az árak megjelenítését kiegészítjük a pénznem megjelölésével is.
- A tatarozás állapothoz az adatot az apartmanok táblából gyűjtjük ki.
  - A szerkesztéshez egy legördülő menüt használunk, amely csak két értéket kap meg (normál vagy lezárt).
  - Felüldefiniáljuk az adatbeállítást, amely a beállítástól függően módosítást végez az apartman táblában.

5.Feladat: adatbázis



5.Feladat: tervezés



5.Feladat: BuildingTableModel

```
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 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);
    }
    …
```

apartmanok számának meghatározása

5.Feladat: BuildingTableModel

```
    ...
    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);
}
```

minimum ár

maximum ár

Adat-kezelési szerepek

- A **`data`** metódus szerep-paramétere (**`role`**) 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::TextAligmentRole`**: szövegigazítás az adathoz
  - **`Qt::TextColorRole`**, …: különböző megjelenítési beállítások, amelyek szabályozhatóak a modell szintjén, illetve a delegált szintjén is

5.Feladat: BuildingTableModel

```
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 appartment" +
            "set renovation = 0 where building_id = " +
            this->data(this->index(index.row(),0)).toString()));
            …
```

apartment táblát kell módosítanunk