DIPLOMAMUNKA

Sterbinszky Nóra Viktória

Debrecen

2009

DEBRECENI EGYETEM
INFORMATIKAI KAR

ÓRAREND KÉSZÍTÉSE SZERVER OLDALON

VISSZALÉPÉSES MEGOLDÁSKERESŐ ALGORITMUS MEGVALÓSÍTÁSA AZ ADATBÁZIS-PROGRAMOZÁSBAN

Témavezető:
Kollár Lajos
Egyetemi tanársegéd

Készítette:
Sterbinszky Nóra Viktória
Programtervező informatikus MSc szakos hallgató

Debrecen
2009

TARTALOMJEGYZÉK

Bevezetés

A mai számítógépes, programvezérelt világunkban is előfordulnak olyan főiskolák és egyetemek, ahol az órarendek még kézi módszerrel készülnek. Általában az előző évi, vagy félévi órarend kerül átdolgozásra, kiegészítve az időközben felmerült igényekkel, a szükségtelen elemek kihagyásával.
Az évről évre változó új helyzet, új tárgyak, esetleg új szakok belépése, az oktatói létszám csökkenése, a termek funkciójának bővülése (számítógépterem, labor, stb.) tovább nehezíti egy minden szempontnak megfelelő komplex órarend összeállítását. Az órarend elkészítése az új félév megkezdése előtt, adminisztrátorok és oktatók összehangolt, sok egyeztetéssel járó munkáját igényli. Ezt a munkát könnyíti meg, és rövidíti le az elkészítés idejét egy órarend-készítő program.**[5]**

A Debreceni Egyetem Informatikai Karán is kézzel állítják össze a szakok órarendjeit, amelyben csak a Microsoft Excel táblázatkezelő program nyújt segítséget. A Neptun tanulmányi rendszer is tartalmaz órarend-szerkesztő modult, amely figyeli az oktató-, az időpont- és terem-ütközéseket, de a szerkesztést kézi úton kell megvalósítani.**[6]**
Az ütközések figyelése nagyon hasznos, de nem elegendő. Így merült fel bennem a teljes órarend-készítési folyamat automatizálásának lehetősége, méghozzá magában az adatbázisban.
Dolgozatomban egy szerver oldali, adatbázisban dolgozó, visszalépéses kereső algoritmust felhasználó órarend-készítő alkalmazást szeretnék ismertetni.

A mesterséges intelligencia által használt kereső algoritmusok megvalósíthatók az adatbázis-programozásban, vagyis szerver oldalon is. A kliens oldalon használt kereső algoritmusok a memóriában tárolják a futási idő alatt keletkező objektumokat, míg adatbázisban ez rekordok beszúrásával is megoldható.
A visszalépéses kereső algoritmus adatbázisban tárolt implementációjához különböző szemléletű programozási eszközök (SQL, PL/SQL, Java) együttes alkalmazására van szükség.

1. Előkészületek

1.1 A feladatnak megfelelő kereső algoritmus kiválasztása

A mesterséges intelligencia leggyakrabban használt kereső algoritmusainak megvalósításai eltérnek a hagyományos esetben és az adatbázisban tárolva. A Java programozási nyelven írt kereső algoritmusok egy gráfot építenek fel működésük során, amelyhez csomópont objektumok definiálására is szükség van. Az operátor példányokat definiáló osztály megadása után az operátor-alkalmazási előfeltételek teljesülésének vizsgálatára szolgáló kódrészletek megírásával befejeződnek az előkészületek. Ezen előkészítő tevékenységekre minden kereső algoritmusnál szükség van.

A szélességi, mélységi és A-algoritmusok egyszerű és optimalizált változatainak adatbázisbeli megvalósítása történhet a csomópontokat helyettesítő táblákban tárolt rekordok felhasználásával, vagy a kliens oldali implementációhoz hasonlóan, memóriában tárolt csomópont objektumokkal. Az utóbbi eset teljes mértékben analóg a kliens oldalon történő végrehajtással. Az előzőhöz szükségünk van egy olyan táblára, amely a Java csúcs objektumainak megfelelő sorokat tartalmaz. Ekkor minden sor rendelkezik egy referenciákból álló kollekció típusú attribútummal, amelyek a gyermek-sorokra hivatkoznak. Mivel ebben az esetben a keresés végén előálló útvonal nem veremben tárolódik, a rekordok egy szülő rekordot címző referenciával is rendelkeznek. Hierarchikus lekérdezéssel ismerhetjük meg az előállt megoldást, mint utat.

A keresés folyamata egy tevékenység, amely nem írható le deklaratív programozási eszközökkel, hiszen a végrehajtás módját szeretnénk megadni. A legtöbb adatbázis-kezelő rendszer biztosítja az ehhez szükséges paradigmát.

Az általam választott, a feladat megoldásához a legjobb felszereltséggel rendelkező rendszer az Oracle Database 11g. Ennek használatakor nem csupán a procedurális szemléletmódot képviselő PL/SQL nyelv áll rendelkezésre, hanem az objektum-orientált Java 1.5-ös verziója is. Az Oracle Database 11g adatbázis-szerver beépített Java Virtuális Gépet is tartalmaz, így nem okoz gondot a Java osztályok futtatása a szerver oldalon.

Egy feladatnak a mesterséges intelligenciában használt kereső algoritmusokkal történő megoldását abban az esetben érdemes áthelyeznünk szerver oldalra, ha az eredményt a keresés végén adatbázisban szeretnénk tárolni. Azért is dönthetünk ilyen megoldás mellett, mert a szervergépek általában sokkal jobb teljesítményűek az asztali számítógépeknél. Az is előfordulhat, hogy a feladat kívánja meg e módszer alkalmazását, mint ahogy az általam készített alkalmazás esetében is.
Az adatbázisban tárolt csomópontok nagy előnye, hogy ha valamilyen okból kifolyólag rákényszerülünk a keresési folyamat megszakítására, de egy későbbi időpontban azt folytatni szeretnénk, a merevlemezen tárolt információ megőrződik a két időpont között is, ellentétben egy memóriában felépített útvonallal.
Amikor egy, az A-algoritmusok családjába tartozó kereső használata mellett döntünk, a heurisztikát lekérdezésekkel is megbecsülhetjük, és egy újabb attribútum felvétele után a rekordokban eltárolhatjuk.

A visszalépéses kereső esetében két úton is eljuthatunk a megvalósításhoz. Az egyik lehetőség az előzőekben ismertetett, a csúcsok rekordokban történő tárolása, illetve a memóriában felépített csomópont objektumokból álló gráf egyik, kezdőállapotból valamelyik célállapotba vezető útjának az előállítása. A második megoldás teljes mértékben kihasználja az adatbázis-kezelő rendszer nyújtotta előnyöket. A szerver oldali megvalósítás minimálisra csökkenti a tranzakciók megszakadásának esélyét, így megtehetjük, hogy a keresés teljes futási ideje alatt egyetlen nagy tranzakcióval dolgozunk. Ekkor minden egyes „csúcs”, azaz rekord beszúrása után létre kell hoznunk egy mentési pontot, amelyek a rekordokkal együttesen képesek átvenni a csomópont objektumok szerepét. A létrehozott mentési pontokat egy verem adatszerkezetben tároljuk. Visszalépés esetén a tranzakció visszagörgetési utasítása a verem tetején lévő mentési pont objektumig hatástalanítja az elvégzett műveleteket. Ha talál egy megoldást az algoritmus, csak akkor véglegesíthetjük a tranzakciót.

Egy órarend elkészítésére az adatbázisban a fenti módszer a legalkalmasabb. Ha megfelelő mennyiségű információ áll rendelkezésre, a feladatnak lesz megoldása. (Kézzel készítve is előbb-utóbb összeáll egy órarend.) Az adatok előkészítése után a legoptimálisabb variáció állhat elő. A gráfban az összes lehetséges megoldás egyforma mélyen van, hiszen csak akkor fejeződhet be a keresés, ha már minden órát elhelyeztünk az órarendben. Körök nem

fordulnak elő ennek a feladatnak a megoldása során, mivel minden óra egyedi a hozzá tartozó időpontokkal együtt. A behelyezést az előfeltételeinek teljesülése után tesszük meg. Az előfeltételek vizsgálatát nem teljesen „szabályos módon” oldottam meg az órák egymáshoz viszonyított időpontjait illetően.

Az alkalmazás működésének részletes leírása előtt az iménti feltételek (oktatói és hallgatói igények) ismertetése következik. A webes felületek jelenleg még nem készültek el, de az előzetes terveik már készen állnak. A program lényegi részét, a magját a visszalépéses megoldáskereső algoritmus képezi, amely köré felépíthetők a webes interfészeket kezelő Java szervlet osztályok.

1.2 Az alkalmazás felhasználói szemszögből

Az órarend készítésével foglalkozó adminisztrátornak jelentős időbeli megtakarítást eredményez egy olyan órarend-készítő program használata, amely kézi szerkesztést nem igényel. Az így felszabaduló idő jelentőségét növeli, ha figyelembe vesszük, hogy az órarend-készítés és a félév kezdésével járó egyéb tanulmányi ügyek intézésének ideje egybeesik.
Az órarend-készítő program használata az oktatók számára is előnyös. Nem kell bemenniük a kar tanulmányi osztályára egyeztetni, hogy mely időpontok felelnek meg számukra az óráik megtartására, hanem egyszerűen, webes felületen keresztül, böngészőből (vékony klienssel) tudják az igényeiket eljuttatni a megfelelő helyre. Az adminisztrátornak nem szükséges ismernie, hogy konkrétan melyek ezek az időpontok, hiszen ezek az adatbázisban tárolódnak, és később az algoritmus használja fel őket.
Amikor elkészült a kari órarend, az oktató lekérdezheti a sajátját, így ellenőrizheti, hogy megfelelő időpontokba kerültek-e az órái.
Az adminisztrátorok a kész órarend minden részét lekérdezhetik, szakonként illetve évfolyamonként is.

Amennyiben szükséges változtatni a már elkészült órarenden, ez megtehető „kézzel” is az adminisztrációs felületről. Ezt akkor célszerű ilyen módon elvégezni, ha csupán egy-két módosításról van szó. Nagy számban jelentkező változtatási igény esetén, vagy ha későn derül ki, hogy szükség lenne még néhány gyakorlati órára, érdemes automatizáltan végrehajtatni a

módosítást. Az „OK” órarend-készítő alkalmazással a tervek szerint ez szintén megvalósítható.

Egy későbbi célkitűzés lehet az elkészült órarend egy az egyben történő átvitele a tanulmányi rendszer adatbázisába. A jelenleg használt tanulmányi rendszerben az ennek megfelelő kézi adatbevitel tizenegyszeri gombnyomást és 4-5 percet vesz igénybe minden egyes behelyezendő tanóra esetén.

1.3 Az igények felmérése

Magam is dolgoztam a Debreceni Egyetem Informatikai Karának Tanulmányi Osztályán, így első kézből szerezhettem információt az órarend-készítésről, és az oktatói igények természetéről. A következő leírás nem teljes, de igyekeztem a leggyakoribb igényeket figyelembe venni.
Az igények két csoportra oszthatók: oktatói és hallgatói igényekre. Az oktatói igényeket maga az oktató közli, a hallgatói igényeket pedig az adminisztrátor fogja össze.

1.3.1 Oktatói igények

A következő felsorolás pontjai az oktató egy vagy több tanórájára vonatkoznak:

- Az óra egy konkrét, adott időpontban legyen. Például hétfőn 10 órától 12-ig.
- Az óra adott időpontokban ne legyen. Abban az esetben, ha az oktatónak egyéb elfoglaltságai miatt néhány időpont semmilyen körülmények között nem felel meg.
- Az óra adott időintervallumba essen. Az intervallum lehet horizontális vagy vertikális. Ez azt jelenti, hogy több napon, de mindig azonos időpontban, vagy a kezdő és befejező időpont nem lényeges, de ezek ugyanazon a napon legyenek.
- Az oktató saját órái közül néhány, vagy akár az összes közvetlenül egymás után legyen megtartva, az órarendjét ne tarkítsák lyukasórák.
- Az oktató órái mindig egyforma időpontokban legyenek (különböző napokon).
- Az órák ugyanabban a teremben legyenek. Előfordulhat, hogy egy óra megtartásához olyan szoftverre van szükség, amely nincs minden gépteremben telepítve, vagy egyéb olyan körülmények állnak fenn, amelyek megkövetelik, hogy az órák ugyanabban a teremben legyenek megtartva.

- Az óra meghatározott teremben legyen. Az oktató konkrétan megadja, hogy melyik teremben szeretné az órát megtartani.
- A terem adott létszám befogadására legyen alkalmas. Az órán részt vevő hallgatók létszáma ne haladja meg a terem férőhelyeinek számát. Géptermek esetében minden hallgató használhasson számítógépet.
- A terem speciális tulajdonságokkal rendelkezzen. Előfordulhat például, hogy egy szoftver csak az első emeleti géptermekben van telepítve, mivel csak ott vannak olyan teljesítményű számítógépek, ahol az adott szoftver az elvárásoknak megfelelően futtatható. Ez a pont azt is magában foglalja, hogy egy óra megtartása géptermet vagy hagyományos termet igényel-e.
- Az oktatónak bármelyik időpont megfelel.
- Az oktatónak bármelyik terem megfelel.

1.3.2 Hallgatói igények

Itt többféle igény (akár személyre szabott órarend elkészítésének az igénye is) felmerülhet. A kari órarend elkészítésénél csak azokat az igényeket célszerű figyelembe venni, amelyek általánosan minden hallgatót érintenek.
Tipikusan ilyen lehet az, hogy egymástól távol eső helyszíneken tartandó órák időben semmi esetre se kövessék egymást közvetlenül. Sőt, a legjobb megoldás a különböző helyszínek közötti utazást is minimalizálni azáltal, hogy az egy helyszínen tartandó órák mindegyike ugyanarra a napra kerüljön. Például a DE IK mérnök informatikus hallgatói a város három, egymástól viszonylag távol eső pontját látogatják minden héten. Ha az utazás minimalizálását nem vesszük figyelembe az órarendjük elkészítésénél, előfordulhat, hogy egy nap mindhárom helyre el kell jutniuk, lehet, hogy nem is egyszer. Ezt mindenképpen kívánatos elkerülni.

A további hallgatói igények „testre szabhatók”, azaz a hallgató a saját felületén tudja ezeket beállítani.

További hallgatói igények

- Az órák a hét elején legyenek.
- Az órák a hét második felében legyenek.

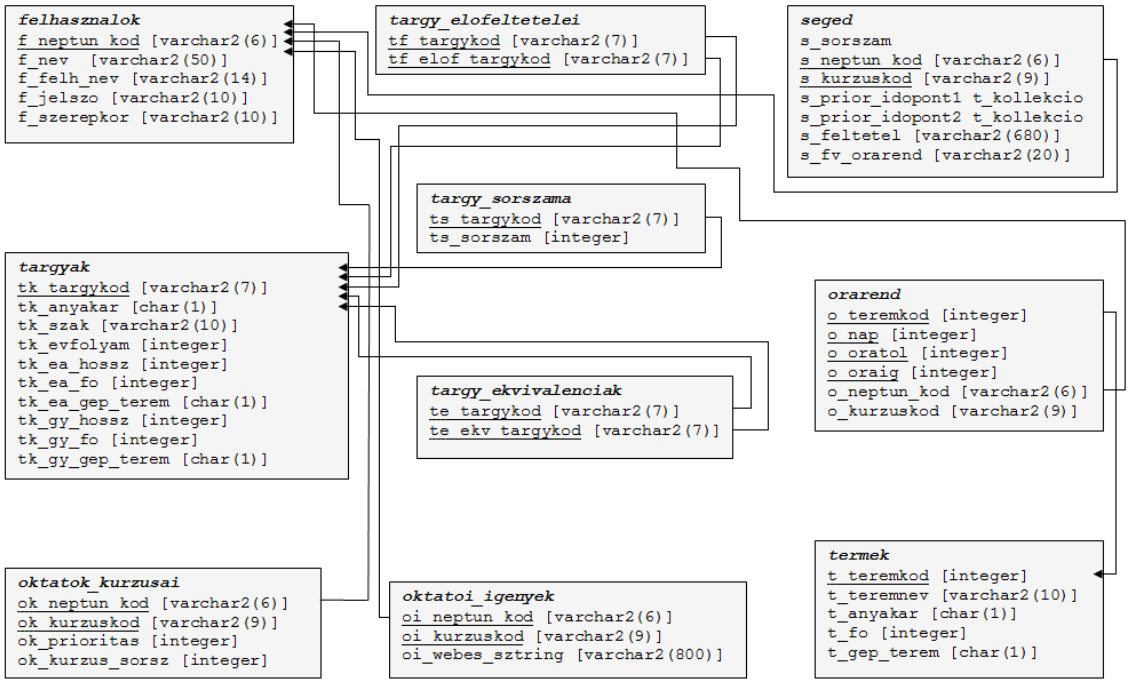
- Ne legyen óra délelőtt.
- Ne legyen óra délután.
- Ne legyen óra meghatározott időpontban (például 12 és 14 óra között).
- Egyéb igények.

Az igényeket minden felhasználó (oktató, hallgató, adminisztrátor) maga állíthatja be a saját felületén.

A minden szempontnak megfelelő program elkészítése, az alkalmazás tesztelése a gyakorlatban sok időt igénylő, összetett feladat. Jelenleg még csak az alkalmazás magja készült el, a webes felületek egyelőre tervezési szakaszban tartanak. A továbbiakban az adatbázisban történő, visszalépéses megoldáskeresővel végzett órarend-készítés bemutatása, és a tervezett, még meg nem valósított modulok leírása következik.

2. Az adatbázis

Az alkalmazáshoz használt adatbázis tíz táblából áll. Ez a szám változhat a további fejlesztés során, lehet valamennyivel kevesebb is, de nagy valószínűséggel inkább növekedni fog.
A táblák felépítése és funkciói a következő szakaszokban kerülnek bemutatásra. A közöttük lévő relációkat a ***2.1 ábra*** szemlélteti.



***2.1 ábra**: Az „OK” alkalmazás által használt táblák és a közöttük lévő relációk*

2.1 A `felhasznalok` tábla

Ez a tábla négy oszlopot tartalmaz, a felhasználó Neptun-kódját (`f_neptun_kod`), a nevét (`f_nev`), a bejelentkezési nevét (`f_felh_nev`), a jelszavát (`f_jelszo`), és utolsó attribútumként a felhasználó lehetséges szerepköreinek megjelölését (`f_szerepkor`).

A Neptun-kód mindig hat karakterből áll. Ennek az attribútumnak az értéke egy karakterlánc, csakúgy, mint a névnek, bejelentkezési névnek, felhasználói névnek és a jelszónak is.

A szerepkörök azonosítására egy-egy karakter szolgál. Például a tanulmányi adminisztrátor azonosítója ’T’, a rendszeradminisztrátoré ’R’, az oktatóé ’O’, a hallgatóé ’H’. Mivel egy személyhez több szerepkör is tartozhat, ennek az attribútumnak az értékei is karakterláncok. Például, ha valaki oktató is és tanulmányi adminisztrátor is, akkor a szerepkör értéke ’OT’ lesz.

A felhasználók tábla az alkalmazás felhasználóinak alapvető adatait, és a bejelentkezéshez (a rendszerbe való belépéshez) szükséges információkat tartalmazza.

Ha új felhasználó regisztrál, ebben a táblában kerülnek rögzítésre az adatai.

2.2 Az `oktatok_kurzusai` *tábla*

Ez a tábla tartalmazza a kurzusok oktatókhoz rendelését.

Az oktatókat a Neptun-kódjuk azonosítja. Itt csak olyan oktató szerepelhet, aki a `felhasznalok` táblában is benne van. Az első attribútum az `ok_neptun_kod`.

Ezután következnek a kurzusok kódjai. Az `ok_kurzuskod` attribútum nem egyedi, mint ahogyan a Neptun-kód sem ebben a táblában, de az elsődleges kulcsot ezek együtt alkotják. A kurzusok kódjai egy sorszámot kapnak, de minden oktató esetében újra kezdődik ez a sorszámozás. A kurzuskód 9 karakter hosszú sztringből áll, amelynek első 7 karaktere a tárgykódot azonosítja. A fennmaradó két karakter azt jelzi, hogy a tárgy előadásáról, illetve gyakorlatairól van-e szó (’E’ és ’G’ karakterek). Ezután következik a sorszám, amelynek egyetlen számjegy (karakter) nagyságú hely áll rendelkezésére. Ha igény van rá, meg lehet növelni az attribútum befogadó képességét, de általános esetben erre nincs szükség, mivel valószínűtlen, hogy egy oktató egy tárgyból 10 vagy annál több gyakorlatot tartson.

Ezután következik az `ok_prioritas` attribútum. Ennek értéke egy egész szám, amely a kurzus prioritását adja meg. Ez határozza meg, hogy milyen sorrendben érdemes az egyes kurzusokat behelyezni az órarendbe. Az a megközelítés a leginkább célravezető, ha először azokat a kurzusokat tesszük be, amelyek a legtöbb hallgatót érintik. Ezek változtatása, mozgatása – bár megoldható, de – nehézkes lenne, és tovább bonyolítaná az egyébként sem egyszerű feladatot. Az elsőéves hallgatók előadásai jó példák erre. Utánuk következnek a magasabb évfolyamok előadásai, amelyeken már kisebb a létszám.

A gyakorlatok prioritása mindig kisebb, mint az előadásoké, hiszen az egy-egy előadáshoz kapcsolódó gyakorlatok száma 6-8 körül mozog. Ez a módszer általában megfelel a célnak, de lehetőség nyílik a változtatásra is.

Az adminisztrációs felületről kurzusonként megadható a hozzájuk rendelt prioritás. Ha az adminisztrátor ezt nem állítja be, az alapértelmezés szerinti értékek kerülnek be a táblába, amelyeket a ***2.2 ábra*** tartalmazza.

	Kötelező előadás	*Választható előadás*	*Kötelező gyakorlat*	*Választható gyakorlat*
5. évfolyam	**12**	**11**	**10**	**9**
4. évfolyam	**14**	**13**	**8**	**7**
3. évfolyam	**16**	**15**	**6**	**5**
2. évfolyam	**18**	**17**	**4**	**3**
1. évfolyam	**20**	**19**	**2**	**1**

***2.2 ábra**: Kurzusokhoz rendelt prioritások táblázatos formában*

Amennyiben a tantárgyak nem csupán két részre oszthatók, hanem vannak kötelező, kötelezően választható és szabadon választható tárgyak is, ezek a prioritás-értékek tovább finomíthatók.

Az adatbázis-tábla következő oszlopa az `ok_kurzus_sorsz`, amely a kurzusokhoz rendelt abszolút sorszámokat tartalmazza, miáltal egy kurzus az oktatója Neptun-kódja nélkül is azonosíthatóvá válik. Ez a funkció a jelenlegi implementációban még nincs kihasználva.

2.3 Az `oktatoi_igenyek` tábla

Ez a tábla tartalmazza az oktatói webes felületről nyert információt.

Az oktató Neptun-kódja és a kurzus azonosítója (`oi_neptun_kod` és `oi_kurzuskod`) együtt képezik az elsődleges kulcsot, így azonosítható egyértelműen minden egyes kurzus.

A harmadik és egyben utolsó attribútum a webről érkező információt tartalmazza sztring formájában. Ebbe vannak „becsomagolva” az alkalmas időpontok, a feltételek, valamint a figyelembe veendő másik órarend, amely óráinak az időpontjaival el kell kerülni a kurzusunk időpontjának ütközését. Ez akkor lehet hasznos, ha az oktató még maga is hallgató egy felsőbb évfolyamon. Ekkor, ha megadja az évfolyamát és a szakját, amivel azonosít egy órarendet, az általa tartott órák nem kerülnek olyan időpontokra, amikor a saját órái (amelyeken hallgatóként vesz részt) lesznek majd.

A gyakorlatok közül egyszerre csak egyet fog felvenni a demonstrátor, ezért feleslegesnek tűnhet az összes lehetséges gyakorlat-időpont kiiktatása. Az elmúlt évek tanúsága alapján, a

bolognai-képzés következtében, a mesterképzésben évente mindössze egy-egy csoportnyi hallgató vesz részt. Ez azt jelenti, hogy a negyedik és az ötödik évfolyamokon egyetlen gyakorlat kerül csak meghirdetésre, ezért az imént említett módszer megfelelő.
Abban az esetben, ha mégis több csoport van a felsőbb évfolyamokon, több gyakorlat tartozik egy-egy tantárgyhoz. Igaz, hogy ezek közül csak az egyik lesz fontos a demonstrátor számára, mégis e módszer szerint az összes gyakorlat időpontja ütközésnek minősül. Ez csak első ránézésre jelent hátrányt, ugyanis ha a demonstrátor már tudja, hogy mikor lesznek hallgatóként az órái, az oktatói felületen letilthatja ezek időpontjait. Ha még nem ismeri ezeket, akkor az összes szóba jövő gyakorlat-időpont tiltólistára kerül. Így hallgatóként megmarad a szabad választása, amelyet az általa tartott órák nem befolyásolnak.
Az oktató több olyan órarendet is megjelölhet, amelyeknek óráival szeretné elkerülni az ütközést.

A webről érkező információ következő darabja a feltétel-sztring. Ez egyfajta formulákat tartalmaz. Például, az oktatói felületről az „*órák legyenek egymás után adott sorrendben*" feltétel ezen formulákkal felírva a következőképpen néz ki: `((Ad==Bc)&&(Bd==Cc))`.
A kifejezésnek teljesen zárójelezettnek kell lennie.
Amikor megtudjuk a konkrét értékeket a kurzusokra vonatkozóan, a formulába behelyettesítődnek ezek kódjai, például: `((indk301g1d==indk301g2c)&&(indk301g2d==inbk301g1c))`.
Az előbbi formulában tehát a nagybetűk jelölik a kurzus azonosítóját, az azonosító végén lévő kisbetű pedig a kurzus egyik adatbázis-táblabeli attribútumát.

A fenti feltétel-sztring a következőképpen olvasandó: az *indk301g1* befejező időpontja essen egybe az *indk301g2* kezdő időpontjával, és az *indk301g2* befejező időpontja essen egybe az *inbk301g1* kezdő időpontjával.
Így biztosítható a három kurzus egymás után következése adott sorrendben. Ennek a feltételnek teljesülnie kell, amikor az órák bekerülnek a helyükre.

Az alkalmazás még nem tartalmaz olyan lehetőséget, hogy az adminisztrátor által megadott formulák –, amelyek eredetileg nem szerepeltek benne, de a gyakori használatuk miatt indokolt felvenni őket a rendszerbe – opció formájában jelenjenek meg a felületeken. A

tervek szerint ez a fejlesztés így oldható meg: létrehozunk egy olyan táblát az adatbázisban, amely ezeket a formulákat tartalmazza. Ezek a rendszerbe való belépéskor lekérdezhetők, és a dinamikusan generált weboldalakon megjelenhetnek.
Minden szerepkör rendelkezik a létrehozás jogával, mindössze annyi a megkötés az új formulákkal szemben, hogy csak adminisztrátorok publikálhatják őket, a többi felhasználó számára csak így tehetők elérhetővé. Az oktatók és a hallgatók csak a saját felületükön hozhatnak létre új, általuk gyakran használt formulákat. Ez a lehetőség további táblákat igényel az adatbázisban, de a rendszer ezzel bővíthetővé válik.

A webről érkező sztring első része tartalmazza az alkalmas időpontokat. A „tiltott" időpontok már nem is szerepelnek ebben a karakterláncban. Az időpontok prioritás szerinti csökkenő sorrendben kerülnek be az adatbázisba. A feldolgozás, vagyis az óra behelyezése majd ennek a sztringnek az elején kezdődik, és ez biztosítja, hogy az órák a legnagyobb valószínűséggel kerüljenek a legnagyobb prioritású helyre.
A webről érkező sztring hosszát minimalizálandó, az intervallumok (ha azonos prioritással rendelkeznek a bennük lévő időpontok) rövidítve kerülnek rögzítésre. Például, ha az oktató 5-ös prioritással megadja a *kedd 12-14*, *kedd 14-16* és *kedd 16-18* időpontokat, ez a *kedd 12-18* formában érkezik be tárolásra. Az óra hosszát ezzel nem veszítjük el, mert a `targyak` tábla minden tárgyhoz tartalmazza az órák hosszát.

Az `igenyek` tábla összegyűjti a felhasználóktól a szükséges információkat, amelyek „nyugalmi állapotban" maradnak egy megadott határidőig. Eddig az időpontig kell eljuttatni minden igényt az adatbázisba. Ha ez megtörtént, kezdődhet az adatok feldolgozása, és az órarend összeállítása.

2.4 A `targyak` tábla

Ennek a táblának az első attribútuma a `tk_targykod`, amely egyúttal az elsődleges kulcs is. Ez egy 7 karakter hosszú sztring, hiszen itt nem szükséges tárolnunk a kurzusok adatait.
A következő oszlop a `tk_anyakar`, amely azt a kart tárolja, amelyhez a tárgy tartozik. Egy hasonló attribútummal rendelkezik a `termek` tábla is. A kapcsolat annak ellenőrzésére szolgál, hogy egy tárgy megfelelő terembe került-e. Például a mérnök informatikus szak

órarendje a fizika tárgyköréhez tartozó órákat is tartalmaz, amelyekhez a földrajzilag máshol található, ezeket a tárgyakat kezelő kar termei kell, hogy rendelődjenek.
Ezután következik a `tk_szak`, amely karakterlánc típusú. Ez tárolja a szakok azonosítóit. Például, ha egy tárgy csak programtervező informatikusok számára van meghirdetve, annak 'P' a kódja, de ha ezt mérnök informatikus hallgatók is teljesíthetik, akkor a kódja 'MP' lesz.
A tábla következő attribútuma a `tk_felev`. Ez tárolja tárgyanként, hogy a tantervi háló szerint mely félévben ajánlott azt teljesíteni.
Ezután három olyan oszlop következik, amelyeknek az értékei a tárgy előadására vonatkoznak. A `tk_ea_hossz`, a `tk_ea_letszam` és a `tk_ea_gep_terem` rendre a tárgyhoz tartozó előadás hosszát, a rajta részt vevő hallgatók létszámát, illetve a hozzá szükséges terem „minősítését" tartalmazzák.
Először szokatlannak tűnhet egy előadás gépterem-igénye, de vannak olyan előadások, amelyeket 20-30 fő számára hirdetnek meg, és számítógépes termet igényelnek.
A soron következő három attribútum az imént felsorolt funkciót látja el a tárgy gyakorlataira vonatkozóan.

A DE Informatikai Karán ismereteim szerint nincs olyan tárgy, amelyhez tantermi és laborgyakorlat is tartozik egyidejűleg, ezért nem láttam szükségesnek ezt az esetet kezelni. Ha mégis igény lenne rá, egy újabb attribútum-hármas felvételével megoldható a probléma.

2.5 A `targy_ekvivalenciak` tábla

Ez a tábla tartalmazza a tárgyekvivalenciákat.
Az előbbi táblát egészíti ki olyan szempontból, hogy egy tárgyról nem csak azt tudhatjuk majd, hogy több szakon is meghirdetésre került-e, hanem azt is, hogy mi a kódja a többi szakon.
Csak olyan tárgyazonosítók szerepelhetnek benne, amelyek a `targyak` táblában is jelen vannak.

2.6 A `targy_elofeltetelei` tábla

Ez a tábla tartalmazza az egyes tantárgyak előfeltételeit. A jelenlegi alkalmazás ezt még nem veszi igénybe.

2.7 A `targy_sorszama` tábla

Ebben a táblában tárolhatjuk a kurzusokhoz rendelt egyedi sorszámokat, így azok önmagukban is egyértelmű azonosítást tesznek lehetővé. Jelenleg ez a funkció sincs használatban, mert minden táblában, amelyben előfordulnak a kurzuskódok, mindig ott van mellettük a Neptun-kód is, és a két attribútum értéke együtt egyedi.

2.8 A `termek` tábla

A termek adatait tároló tábla.

Minden terem rendelkezik egy kóddal is a nevén kívül, amely azonosítja a termet. A `t_teremkod` és `t_teremnev` attribútumok mellett itt is van egy `t_anyakar` nevű oszlop, amely a `targyak` táblában lévő, hasonló nevű attribútumhoz kapcsolódik. Így tudjuk ellenőrizni, hogy egy óra megfelelő terembe került-e.

Minden teremhez tartozik még egy szám, amely megmondja, hogy hány fő fér el benne. Ez a `t_fo` attribútum.

A tábla utolsó oszlopa a `t_gep_terem`, amely egyetlen karaktert tartalmaz: a ’G’-t, ha gépteremről van szó, és a ’T’-t, ha „hagyományos” a terem.

2.9 A `seged` tábla

Az előfeldolgozás után ebbe a táblába kerülnek be a webről kapott információk a megfelelő formában, és közvetlenül ebből a táblából dolgozik majd a megoldáskereső algoritmus is.

Az első attribútum az `s_sorszam`, amely a rekord sorszámát mutatja, és egyedi minősítéssel bír.

Az `s_neptun_kod` és az `s_kurzuskod` együtt alkotják az elsődleges kulcsot. Ezek értékeiből pontosan tudhatjuk, hogy melyik kurzust kell behelyezni az órarendbe.

A következő két attribútum, az `s_prior_idopont1` és az `s_prior_idopont2` egy-egy kollekciót tartalmaznak. Mindkét kollekció egyforma, beágyazott tábla típusú, amely objektumokból áll. Ezek az objektumok reprezentálják az alkalmas időpontokat.
Az objektum típus négy mezőt tartalmaz, amelyek mindegyike egész típusú. Az első a prioritást, a második a napot azonosítja, a harmadik a kezdés időpontját, míg a negyedik a befejezés időpontját szolgáltatja.

Kezdetben a `seged` tábla `s_prior_idopont2` attribútuma nem tartalmaz egyetlen értéket sem. Ez arra szolgál majd, hogy a már alkalmazott operátorokat, vagyis a már kipróbált időpontokat tárolja. A párjából, az `s_prior_idopont1` attribútum értékeiből kerülnek át a keresés során a felhasznált időpontok. Így ez a kollekció egyre csökken, míg a másik egyre növekszik.

A következő oszlop a táblában az `s_feltetel`. Ide kerül a webről érkező sztring feltételt tartalmazó része. Ebbe a feltételbe helyettesítődnek majd be a konkrét attribútum-értékek a megfelelő pozíciókra, így válik ellenőrizhetővé a feltétel.
Az utolsó attribútum az `s_fv_orarend`, amely a figyelembe veendő órarendet tartalmazza karakterlánc formájában. Ez a sztring a szak kódját és az évfolyamot (félévet) foglalja magában.

2.10 Az `orarend` tábla

Ez az utolsó tábla, amely az órarendbe bekerült kurzusokat tartalmazza.
Hat attribútuma közül négy alkotja az elsődleges kulcsot. Az `o_teremkod`, az `o_nap`, az `o_oratol` és az `o_oraig` egyértelmű azonosítást tesz lehetővé.
Az `o_neptun_kod` és az `o_kurzuskod` együtt kulcsjelölt, de a kereső algoritmus működésének megkönnyítése érdekében célszerűbb, ha az elsődleges kulcs-integritási megszorítás betartatásával igyekszünk az óraütközéseket elkerülni.

Ez azonban nem elegendő. Például, ha már bent van a táblában egy olyan kurzus, amelynek terme és időpontja: *213-as, hétfő 12 órától 14 óráig*, és behelyezésre vár egy kurzus a következő paraméterekkel: *213-as terem, hétfő 12 órától 13 óráig*.

Ekkor az elsődleges kulcs-integritási megszorítás nem sérül, hiszen a négy érték közül három egyezik, de egy eltér. Viszont behelyezni sem lenne jó, mivel mégiscsak ütközésről van szó: átfedésről. Ezért szükség van egy trigerre is, amely az `orarend` táblába történő beszúráskor és módosításkor „tüzel”.

A ***2.3 ábrán*** látható a trigger forráskódja, amely ezeket az átfedéseket figyeli, és ha olyan kurzus várakozik beszúrásra, amely ütközne egy, már bent lévővel, a trigger kiváltja ugyanazt a kivételt, amely az elsődleges kulcs-integritási megszorítás megsértésekor váltódik ki. Így az óra nem kerülhet be az adott paraméterekkel.

```
create or replace trigger tr_utkozesvizsgalo
  before insert or update on orarend
  for each row
declare
  e exception;
  pragma EXCEPTION_INIT(e, -1);
  v_otol  orarend.o_oratol%type;
  v_oig   orarend.o_oraig%type;
  v_nk    orarend.o_neptun_kod%type;
  cursor c_orak is
    select o_oratol, o_oraig from orarend
      where o_teremkod = :new.o_teremkod
            and o_nap = :new.o_nap;
  cursor c_ipsz is
    select o_neptun_kod from orarend
      where o_nap = :new.o_nap
            and o_oratol = :new.o_oratol
            and o_oraig = :new.o_oraig;
begin
  open c_orak;
  loop
    fetch c_orak into v_otol, v_oig;
    exit when c_orak%notfound;
    if(v_otol < :new.o_oratol and v_oig > :new.o_oratol)
        or (v_otol < :new.o_oraig and v_oig > :new.o_oraig)
        or (:new.o_oratol < v_otol and :new.o_oraig > v_otol)
        or (:new.o_oratol < v_oig and :new.o_oraig > v_oig)
        then
      raise e;
    end if;
  end loop;
  close c_orak;
  open c_ipsz;
  loop
    fetch c_ipsz into v_nk;
    exit when c_ipsz%notfound;
    if(v_nk = :new.o_neptun_kod) then
```

```
      raise e;
    end if;
  end loop;
  close c_ipsz;
end;
/
```

***2.3 ábra**: Az órák időpontjainak átfedéseit ellenőrző trigger forráskódja*

3. Az alkalmazás programozói szemszögből

3.1 A program működése

A program négy Java osztályból, egy PL/SQL eljárásból, és egy triggerből áll, amely szintén PL/SQL nyelven van implementálva.

Az egyszerűség kedvéért a négy Java osztály „egybe van csomagolva”, vagyis az egyik osztályon belül helyezkedik el a másik három. Ezek beágyazott osztályokká váltak, de mivel statikus tagok, külső szintű osztályoknak minősülnek. Nem kell léteznie a befoglaló osztály egyetlen példányának sem ahhoz, hogy példányosíthassuk őket.
Mivel szerver oldalon fut majd a program, nem kell, hogy létezzen az osztálynak `main` metódusa, e nélkül is futtatható. Azt a metódust viszont, amely közvetlen vagy közvetett módon meghívja az összes többit, publikálnunk kell. Ennek osztályszintű metódusnak kell lennie, tehát el kell látnunk a `static` módosítóval.

A memóriában felépített visszalépéses kereső algoritmusok állapotterekkel, operátorokkal, feltételekkel dolgoznak. Az operátorokat sorba rakják a könnyebb kezelhetőség és a csoportosíthatóság céljából. Az utóbbi során különítődnek el a már felhasznált operátorok a még nem használtaktól. Egy állapotra addig alkalmazzuk a sorba rendezett operátorokat, amíg csak el nem fogynak.
Ha egyik operátor sem alkalmazható, vissza kell eggyel lépni az aktuális állapotot tartalmazó gráfbeli csúcs szülő csúcsába.
A szülőben tovább alkalmazzuk az operátorokat. Ha itt is elfogynak, ismét visszalépünk.
Ha a feladatnak van megoldása, egy idő után biztosan megtalálja az algoritmus, és a veremben ott lesz az útvonal, amely a kezdő állapotot tartalmazó csúcsból egy célállapotot tartalmazó csúcsig vezet.
Célszerű a csúcsok tárolására vermet használni, így amikor visszalépünk, egyszerűen csak leemeljük a legutolsó csúcsot a verem tetejéről.

A visszalépéses kereső algoritmus adatbázisban történő megvalósítása némiképpen eltér ettől, de alapjában véve ugyanarról van szó.

Nem a memóriában és gráfbeli csúcsokban tároljuk az előálló újabb állapotokat, hanem magában az adatbázisban, rekordok formájában.
Az operátorok a kurzusok lehetséges időpontjai lesznek. Nem a csúcsot reprezentáló osztály adattagjaival valósítjuk meg a már használt operátorokat tartalmazó adatszerkezetet, és a még nem használtakat tartalmazót, hanem a tábla oszlopainak segítségével.
A feltételeket szintén táblákban tároljuk. Mindig tudjuk, hogy melyik kurzushoz tartoznak, hiszen ezek a többi attribútum-értékkel együtt alkotnak egy rekordot.

A visszalépést úgy biztosíthatjuk, hogy visszagörgetjük a tranzakciót egy mentési pontig. Az algoritmus teljes futása egyetlen nagy tranzakció köré épül, amelyet ez alatt egyszer sem véglegesítünk, csak ha már előállt egy megoldás.
A tranzakció megszakadásának, azaz sikertelen befejezésének esélye nagyon kicsi. Nem áll a program és az adatbázis között hálózati réteg, így gyakorlatilag garantált a sikeres tranzakció-végrehajtás.

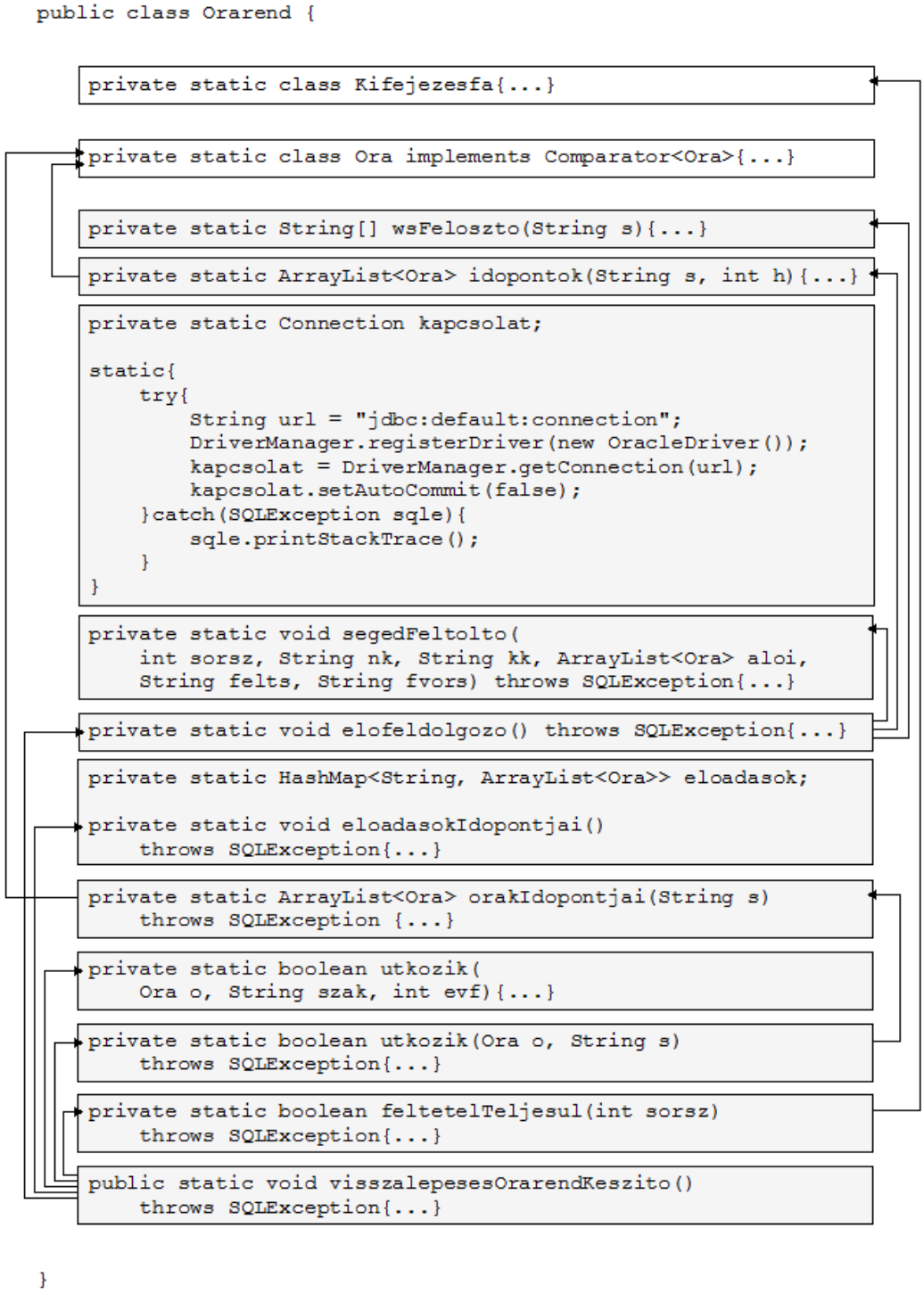
Minden sikeres órabehelyezés után létrehozunk egy mentési pontot. Ha a következő kurzus behelyezésénél nem alkalmazható egyik operátor sem – egyik időpont sem megfelelő az aktuális állapotban –, vissza kell lépnünk. Ez mindössze egy visszagörgetést igényel a legutóbbi mentési pontig.

A mentési pontokat veremben érdemes tárolni. A tranzakció visszagörgetése után a legutóbbi mentési pontot kivesszük a verem tetejéről. Ezután egy új kerül majd a helyére. Így a mentési pontokon visszafelé lépkedve megkaphatjuk a kezdő állapot és az egyik lehetséges végállapot közötti utat. Az órarend-készítés feladatánál nem ennek az útnak a „feltérképezése” a célunk, hanem egy tetszőleges célállapot elérése, és ezzel az órarend előállítása.

3.2 Az osztályok és a metódusok

A négy osztály közül a legkülső szinten lévő, amely magában foglalja a többit, az `Orarend` osztály. Ennek metódusai a visszalépéses keresés, valamint az ehhez szükséges segédmetódusok.
Ebben az osztályban helyezkedik el két statikus tagosztály, a `Kifejezesfa` és az `Ora`.

A ***3.1 ábra*** a hívási láncok megjelölésével nagy vonalakban áttekintést nyújt az `Orarend` osztály tagjainak működéséről.



***3.1 ábra**: Az `Orarend` osztály metódusai és a közöttük lévő hívási láncok*

A kifejezésfára a feltétel kiértékelése miatt van szükség. A feltételt az eredeti formájában kell megőriznünk, hogy később el tudjuk végezni a konkrét attribútum-értékeket behelyettesítését. Ezután a feltételt ki kell értékelni. Sztringként került tárolásra, és a behelyettesítés után is megőrzi típusát. A kiértékelés eredményeképpen egy logikai értéket kapunk vissza.
A kiértékelés folyamatát kifejezésfa építésével és bejárásával oldottam meg.

A `Kifejezesfa` osztály tartalmaz egy statikus tagosztályt, a `Csomopont`ot. Ez szintén külső szintű osztály, de logikailag a kifejezésfához tartozik: annak csúcsait reprezentálja.

Az `Ora` osztály egy példánya tartalmazza az óra prioritását, a napot, a kezdő és a befejező időpontot. Ez az osztály felel meg az adatbázisban tárolt objektumtípusnak, amely szintén hasonló mezőkkel rendelkezik. A keresés során ezekből az objektumokból álló kollekció elemeit kell mozgatnunk, így felesleges át-, majd visszaalakítani őket Java objektumokká. Az áthelyező metódus törzse ezért Java helyett PL/SQL nyelven íródott.

3.2.1 A `Kifejezesfa` osztály felépítése és működése

Ebben az osztályban helyezkedik el a `Csomopont` statikus tagosztály, amely a kifejezésfa egy csúcsát ábrázolja.
A csomópont osztálydefiníciója három adattagot tartalmaz, amelyek közül kettő a fa csúcsának gyermekeire, a bal és a jobb oldali részfára mutató referencia. Ezek típusa szintén `Csomopont`.
A harmadik adattag az értéket képviseli. Ennek típusa `java.lang.Object`.
A fában egész számok, sztringként ábrázolt operátorok és logikai értékek szerepelnek majd. Az operátorok logikai és aritmetikai műveleteket szimbolizálnak.
Az osztály mindhárom adattagjához tartoznak lekérdező és beállító metódusok.
A kifejezésfa kiértékelését végző metódus a `Csomopont` osztályban helyezkedik el.
A kiértékelés során az egész értékeket tartalmazó csúcsok továbbra is egészeket fognak tartalmazni, viszont az operátorokat tartalmazók logikai értéket kapnak majd. A fa gyökere is ilyen típusú érték lesz a visszatéréskor.

3.2.1.1 A `kiertekel` metódus

A kifejezésfa felépítése olyan, hogy egy csúcsnak minden esetben két rákövetkezője van, egyetlen kivétellel: ez a tagadás operátora. Ennek csak jobb oldali gyermeke van.
Ha kétoperandusú operátorról van szó, a jobb oldali részfa értéke mellett a bal oldali részfa értékét képviselő változót is el kell látnunk kezdőértékkel. Ezután már csak olyan csomópontokkal dolgozunk majd, amelyeknek két rákövetkezőjük van.
A bal-, illetve jobbérték meghatározására rekurzívan hívjuk meg a `kiertekel` metódust a bal, illetve a jobb oldali részfára.

3.2.1.2 A `felepit` metódus

Feladata egy sztringből kifejezésfát építeni. A műveletek végrehajtási sorrendjét – a fa felépítését – a zárójelek határozzák meg.
A metódus nem tartalmaz validálást, feltételezzük, hogy a sztring egy szintaktikailag helyes kifejezés. Az ellenőrzést egy korábbi időpontban kell elvégeznünk, mielőtt a karakterlánc az adatbázisban rögzítésre kerül. Ha szintaktikai hibát tartalmaz, erről azonnal tájékoztatnunk kell a felhasználót.
A metódus egy `Csomopont` példánnyal tér vissza, amely vagy egy önmagában álló operandust tartalmaz, vagy egy operátort, amelynek bal oldali rákövetkezője az egyik operandust, míg a jobb oldali a másik operandusát tartalmazza.

3.2.1.3 A `feltetelKiertekelo` metódus

Az osztály többi metódusát felhasználva eldönti egy „sztringbe csomagolt” feltételről, hogy az igaz vagy hamis.
A kereső algoritmus ezt a metódust használja majd az oktatók által megadott feltételek kiértékelésére, annak eldöntésére, hogy az órák leendő időpontjai, helyei és egyéb paraméterei megfelelő értékeket kaptak-e.

3.2.2 Az `Ora` osztály

Ennek az osztálynak a példányai alkotják a lehetséges óraidőpontokat, amelyek a `seged` táblában tárolt kollekciók elemeinek, az `idopont` típusnak a megfelelői.
A négy adattag az időpont prioritását, a napot, a kezdő és a befejező időpontot reprezentálja. Mind a négy adattag privát láthatóságú, ezért publikus lekérdező és beállító metódusok is tartoznak hozzájuk.

3.2.2.1 A két `equals` metódus

A `java.lang.Object` típusú paraméterrel rendelkező `equals` a `java.lang.Object` osztályban lévő metódust definiálja felül. A négy adattag egyidejű egyezősége esetén igaz értékkel tér vissza, egyébként hamissal.
A metódus neve túl van terhelve: a másik `equals` egy `Ora` típusú objektumot kap paraméterül, ám ennek és az aktuális példánynak csak három adattagját hasonlítja, a prioritás kimarad.

3.2.2.2 Az `egybeEsik` metódus

Ez szintén `boolean` visszatérési típussal rendelkezik, mint a két `equals`.
Abban az esetben ad vissza igaz értéket, ha két óra egybeesik, tehát az időintervallumuknak van közös része.
A metódus példány szintű, ezért elegendő egy paraméterének lennie, amely egy `Ora` típusú objektum. A másik objektum az aktuális példány.

A tagosztályok leírása után következzen a befoglaló osztály ismertetése.

3.2.3 Az `Orarend` osztály

Ez az osztály a fő egysége a teljes programnak, amely a kereső algoritmus implementációját tartalmazza. Az előzőekben bemutatott osztályok példányaival ennek az osztálynak a metódusai dolgoznak, és az adatbázis-táblákat is ezek kezelik. Az adatbázis-kapcsolatért szintén ez az osztály a felelős.

Mivel az osztály maga is az adatbázisban van tárolva, nincs szükség a kapcsolat URL-jének specifikálására: elegendő megadnunk egy belső kapcsolatot. Ekkor a korábbi adatbázis-csatlakozáshoz megadott paraméterek kerülnek ismét használatba.
Az osztály egy kapcsolattal dolgozik, mivel a végrehajtás ennél a kereső algoritmusnál nem párhuzamosítható, a szekvenciális végrehajtás miatt elegendő egy kapcsolatot fenntartani. Az osztály minden metódusa ezt használja, de nem egy időben.
Ezen okok miatt a legcélszerűbb a kapcsolat létrehozását tartalmazó kódrészletet egy statikus inicializáló blokkban elhelyezni. Ha nem csak az órarend-készítő alkalmazás magját valósítjuk meg, hanem a webes interfészeket kezelő osztályokat is, akkor kényelmesebb lehet egy jellemzőként (property) definiált kapcsolatot kezelni, amelyhez beállító és lekérdező metódusok egyaránt tartoznak. Az alkalmazás motorját képező megoldáskereső algoritmus önmagában történő implementációjához viszont elegendő a statikus blokk, amely az osztály betöltődésekor már ki is építi az adatbázis-kapcsolatot.
A visszalépéses megoldáskeresés szempontjából döntő fontosságú, hogy a kapcsolatnál kiiktassuk az automatikus tranzakció-véglegesítést. A Java alapbeállításai szerint ez engedélyezve van, ami azt jelenti, hogy az adatbázison végzett minden egyes DML-művelet automatikusan véglegesítésre kerül, tehát nem lesz visszavonható. Az algoritmus éppen azt fogja kihasználni, hogy ha ez a paraméter ki van kapcsolva, akkor az adatbázis előző állapota visszaállítható az előző véglegesítésig, vagy egy meghatározott mentési pontig. Ezért az inicializáló blokkban a kapcsolatnál be kell állítanunk az automatikus tranzakció-véglegesítés letiltását.
Ahhoz, hogy ezek után a metódusok el tudják érni az adatbázis-kapcsolatot, szükség van egy `java.sql.Connection` típusú statikus adattagra. Az inicializáló blokk ennek az objektumnak specifikálja a paramétereit.

Az osztály majdnem minden adattagja reguláris kifejezéseket tartalmazó, `java.lang.String` típusú objektum, amely nem köthető példányokhoz.
Az `Orarend` osztály egyetlen példányára sincs szükségünk a feladat megvalósításához, így annak minden adattagja és metódusa statikus, vagyis osztály szintű.
Annak a metódusnak, amelyik a többi közvetlen vagy közvetett hívója, mindenképpen statikusnak kell lennie, mivel az adatbázisban a csak ezzel a minősítéssel rendelkező Java metódusok publikálhatók függvényként vagy eljárásként.

A reguláris kifejezések elengedhetetlenek a sztringek feldarabolásához, és a megoldáskeresés előkészítő fázisához.

3.2.3.1 A `feloszt` metódus

A `feloszt` metódus a weben keresztül érkező sztringet szedi darabokra.
Az összes információ egyetlen sztringbe „csomagolva” érkezik be az adatbázisba. Ezt három részre kell osztanunk, hogy a táblák megfelelő attribútum-értékeiként elhelyezhessük őket.
Az első rész az időpontokat tartalmazza, amelyet a következő szakaszban bemutatásra kerülő metódussal még tovább finomítunk.
A második rész a feltétel, amelynek konkrét értékeiből a `Kifejezésfa` osztály metódusainak segítségével állapítjuk meg, hogy teljesül-e vagy sem.
A sztring harmadik része egy szak-évfolyam pár, azt az órarendet adja meg, amelynek óráival nem szabad ütköznie az aktuális órának. Azért van szükség ilyen megadási módra, mert a megjelölt órarend óráinak helyei is a keresés során dőlnek majd el.
Az adattagként tárolt reguláris kifejezések tartalmazzák azt a mintát, amely alapján tokenizáljuk a karakterláncot, valamint a következő metódus tevékenységénél is ezeket használjuk az információ kinyerésére.

3.2.3.2 Az `idopontok` metódus

Ez egy segédmetódus, amely könnyen feldolgozható formára hozza az időpontokat.
A webről érkező időpont-sztring az azonos prioritású, összefüggő intervallumot képező (közvetlenül egymás utáni) időpontokat rövidítve „kézbesíti”. Az adatbázisban ezt vissza kell alakítanunk.
A metódus paraméterként kapja az időpontokat tartalmazó karakterláncot és egy egész típusú értéket, amely az adott kurzus óráinak a hossza. Ezt az értéket a hívás helyén kapjuk meg a `targyak` táblából. Ezek az órák általában 2 órás időtartamúak, de olykor vannak kivételek. Az is releváns, hogy előadásról vagy gyakorlatról van-e szó. Létezik olyan kurzus, amelynél az előadás 3 órás, míg a gyakorlat eggyel kevesebb. Ezért a `targyak` táblában kurzusonként az előadás és a gyakorlat paraméterei is tárolva vannak. Az intervallumként érkező időpontok miatt az óra hossza különösen fontos.

Például, ha egy 1 órás időtartamú előadás vagy gyakorlat időpontjait dolgozzuk fel, és nem ismerjük a hosszt, nem tudhatjuk, hogy a *10-től 12-ig* egy időpontot vagy egy időintervallumot jelent-e.
Az intervallumot a metódus a hossz alapján osztja fel a megfelelő részekre. Itt már különálló időpontokról van szó, amelyeket egyesével veszünk majd sorra. A metódus a belőlük képzett `Ora` objektumok kollekciójával tér vissza, amelyet egy `java.util.ArrayList` objektum implementál.

3.2.3.3 Az `elofeldolgozo` metódus

Ennek a metódusnak a feladata a webről érkező sztringet még eredeti állapotában tároló, `oktatoi_igenyek` tábla feldolgozása, amely során megtörténik a sztring tokenizálása, a daraboknak a `seged` tábla megfelelő oszlopaiban való elhelyezése, és annak biztosítása, hogy a keresés a megfelelő sorrendben dolgozza fel a `seged` tábla rekordjait. A metódusnak a kurzusokhoz rendelt prioritás alapján csökkenő sorrendben kell behelyeznie a rekordokat, hogy ezt biztosítani tudja.
A `seged` táblában szereplő kollekció típusú attribútumok miatt egy segédmetódust veszünk igénybe azok feltöltésekor.

Az `elofeldolgozo` metódus „lelke” az a `for`-ciklus, amelynek segítségével az összes prioritást a megfelelő (csökkenő) sorrendben tudjuk érinteni. Egy olyan lekérdezést kell a ciklusmagban végrehajtanunk, amely visszaadja azokat a sorokat az `oktatoi_igenyek` táblából, amelyek megfelelő prioritással rendelkeznek. Ez az `oktatok_kurzusai` táblából nyerhető ki. Meg kell még ismernünk a kurzus hosszát, amelyhez a `targyak` táblát vesszük igénybe.
Az `idopontok` metódus hívásával, a hossz ismeretében előállíthatóvá válik az időpontokból álló kollekció, amelyet a `segedFeltolto` metódus helyez a `seged` táblába.
Az `elofeldolgozo` metódus ezzel elvégezte tevékenységét.

Az `Orarend` osztály további metódusainak feladata az operátor-alkalmazási előfeltételek teljesülésének ellenőrzése, és az órarend elkészítése visszalépéses megoldáskeresővel.
Az előbbi funkciót öt metódus látja el.

Az `eloadasokIdopontjai` és az `orakIdopontjai` egy-egy `java.util.ArrayList` típusú kollekciót töltenek fel azon órák adataival, amelyek relevánsak egy aktuális óra behelyezésénél. Az algoritmus kihasználja azt a tényt, hogy mire a gyakorlati kurzusok feldolgozásához ér, addigra már ismert az összes előadás időpontja. Ezt a kurzusokhoz rendelt prioritás biztosítja, amely szerint az órák beillesztése zajlik. Az előadások egymás után kerülnek feldolgozásra. Az első gyakorlat elérésével az előadások feldolgozása befejeződik.

Az `eloadasokIdopontjai` metódus bemásolja minden évfolyam előadásainak időpontját egy olyan `java.util.HashMap` típusú objektumba, amelynek kulcsa az évfolyam, míg értéke egy `java.util.ArrayList` típusú kollekció, amely `Ora` objektumokat tartalmaz. Ebben az esetben már nincs szükség az órák prioritására, ezért ennek helyét az évfolyam veszi át.

Amikor egy gyakorlatra alkalmazható operátorok, időpontok alkalmazásának előfeltételét ellenőrzi az algoritmus, első lépésként ez imént említett metódust hívja meg, amelynek implementálásához egy statikus, osztályszintű adattag felvétele szükséges. Az erőforrások fölösleges igénybevételét azzal küszöböljük ki, hogy az előadások időpontjainak tárolására csak abban az esetben kerül sor, amikor elérjük az első gyakorlatot.

Az adattag feltöltése után a további gyakorlatoknál nem szükséges ezen előadás-időpontokat ismételten lekérdeznünk, csupán a megfelelő évfolyam és szak óráinak paramétereire vagyunk kíváncsiak. Egy adott évfolyam-szak páros előadásainak időpontjaihoz az `orarend` és a `targyak` tábla segítségével férhetünk hozzá.

A következő előfeltétel-ellenőrzéshez az `orakIdopontjai` metódus nyújt segítséget, amely a webről érkező karakterlánc harmadik részében található másik órarend óráival való ütközések elkerülésére készült.

Míg az `eloadasokIdopontjai void` visszatérési típusú, és üres paraméterlistával rendelkezik, addig az `orakIdopontjai` esetében a másik órarend megjelölését, mint sztringet vesszük át, és az időpontokat reprezentáló `Ora` objektumok `java.util.ArrayList` típusú kollekciójával térünk vissza.

Az itt használt lekérdezés hasonló az előbbi metódusban használthoz. Míg a másik metódus minden szak minden évfolyamát végigvizsgálja a kívánt eredményért, addig az

`orakIdopontjai` mindössze a paraméterként átvett sztringben szereplő szak vagy szakok és évfolyam órarendjét keresi meg.
A megfelelő szakok megtalálásánál nem szabad arról elfeledkeznünk, hogy a szakkódok sztringként jelennek meg az adatbázis `targyak` táblájában, és ha egy tárgy – bár két kód megjelölésével, de mégis – két vagy több szak számára kerül meghirdetésre, a sztringben is több karakter jelenik meg. A vizsgálat során tekintettel kell lennünk arra, hogy egy karakter nem biztos, hogy teljesen kimeríti az adott szakhoz tartozó tárgyak listáját.

Az imént ismertetett két metódust közvetlenül másik két metódus használja, amelyeket a kereső algoritmust tartalmazó kódrészlet hív meg.
Mindkét metódus neve `utkozik`, és logikai visszatérési típussal rendelkeznek.
Amelyikükre az `eloadasokIdopontja` hívása után kerül a vezérlés, három paraméterrel rendelkezik: a kereső által behelyezni kívánt soron következő `Ora` objektum mellett átadjuk neki a szakokat tartalmazó karakterláncot és az évfolyamot, hogy megtaláljuk az összes előadást tartalmazó kollekcióból az aktuális órát érintő időpontokat.
A másik hasonló nevű metódus eggyel kevesebb paraméterrel rendelkezik. Ebben is szükségünk van az `Ora` objektumra, de a szakot és az évfolyamot „egybe csomagolva” kapja.
Az ezen információkat tartalmazó sztringet továbbadjuk az `orakIdopontjai` metódusnak, amely nem adattagot használ az időpontok gyűjteményének tárolására, hanem egy lokális változót, amelynek értékével a működése befejeztével visszatér. Ebben az esetben minden gyakorlathoz egyedi igények tartozhatnak, és a kapott kollekciót más programegység nem használja, ezért nem indokolt a globális változó használata.
Az `eloadasokIdopontjai` metódussal együttműködő `utkozik` nem közvetlenül hívja azt. Erről a kereső algoritmus gondoskodik, mielőtt az `utkozik`et felhasználná.

Mindkét ütközésvizsgálat esetében a kollekció elemein és a paraméterként kapott objektumon operáló, az `Ora` osztályban lévő `egybeEsik` metódust használjuk annak eldöntésére, hogy azok nem fedik-e egymást.
A metódusok hamis értékkel térnek vissza, ha a vizsgálat során egyetlen óraütközés sem fordult elő, és igaz értékkel, ha akár csak egy vizsgált órapár időpontja is fedi egymást.

Az utolsó operátor-alkalmazási előfeltétel vizsgálata nem teljesen „szabályos módon” történik. Nem az operátor alkalmazása előtt, hanem utána állapítjuk meg, hogy teljesült a feltétel vagy sem.
Amennyiben ezt a megállapítást mégis hamarabb szeretnénk megtenni, ez elvégezhető, de bonyolultabb módon. Az általam választott megoldás nem kíván kétféle kezelési módot attól függően, hogy a kurzus már az órarendben van, vagy csak most készülünk betenni oda. Ha ugyanis már betettük, akkor ugyanaz a lekérdezés használható mindegyik kurzus esetében (de különböző módon paraméterezve), ha viszont az egyik óra a helyén van, de a következő még nincs, akkor csak az első adataihoz tudunk lekérdezéssel hozzáférni, a másik tulajdonságait tisztán Java nyelvi eszközökkel deríthetjük fel.
A rövidebb utat választva, a lekérdezések eredményét egy olyan `java.util.HashMap` típusú objektumban tároljuk, amelyben a kulcs a kurzuskódokat tartalmazó karakterlánc, az érték egy újabb `java.util.HashMap` típusú kollekció, amelynek kulcsa az `orarend` tábla négy, az elsődleges kulcsot alkotó attribútumának felel meg (`o_teremkod`, `o_nap`, `o_oratol` és `o_oraig`), az értéke a kurzuskódhoz tartozó rekord adott attribútumának értéke.

A formulát tartalmazó sztringben a kurzuskódok egy érvényes `o_kurzuskod` attribútum-értéket jelölnek, amelyeknek a végén ott áll az attribútumot azonosító karakter.
A sztring tokenizálásával a megfelelő helyekre helyettesíthetjük be a kurzuskódhoz tartozó attribútum-értékeket, majd a sztringet a darabjaiból újra összefűzve elvégezhető a feltétel kiértékelése. Ehhez a `Kifejezesfa` osztály `feltetelKiertekelo` metódusát vesszük igénybe. Ha igaz értéket kapunk vissza, az órát a helyén hagyhatjuk, de ha hamisat, akkor vissza kell vonnunk az iménti órabeszúrás hatásait.

3.2.3.4 A `visszalepesesOrarendKeszito` metódus

Ez a fő metódus, a visszalépéses órarend-készítés algoritmusának implementációja.
(Az Oracle rendszerben futtatott Java osztályoknak nem szükséges `main` metódussal rendelkezniük.)
Ez a metódus hívja közvetlenül vagy közvetett módon a többi Java metódust, és az egyetlen PL/SQL nyelven írt eljárást is.

Az előfeldolgozás során az `igenyek` táblában tárolt, weben keresztül beérkezett információt kell feldolgozásra előkészíteni, ezért a fő metódusban először az `elofeldolg` hívódik meg.
Ha a `seged` táblát feltöltöttük a megfelelő adatokkal, létrehozzuk a metódus lokális változóit, többek között egy mentési pont (`java.sql.Savepoint`) példányokból álló verem (`java.util.Stack`) objektumot.
Miután egy lekérdezés segítségével megállapítjuk, hogy milyen nagy a `seged` tábla, azaz hány sor található benne, egy, sorokat számláló változó növelésével minden sorhoz hozzáférhetünk az egyedi `s_sorszam` attribútumon keresztül. Az előfeldolgozás során a táblában kurzus-prioritás szerinti csökkenő sorrendben történt a rekordok beszúrása. A rendezéssel tudjuk az előadások elsőbbségét biztosítani az órarendben történő elhelyezés során.
Az aktuális sor lehetséges időpontokat tartalmazó kollekciója szolgáltatja az operátorokat, pontosabban, a behelyező operátor paramétereit. A kollekció elemei az oktató által megadott prioritások szerinti csökkenő sorrendben kerülnek feldolgozásra.
A `seged` tábla `s_prior_idopont1` oszlopában lévő gyűjtemény a felhasznált operátorok függvényében egyre fogy, míg párja, az `s_prior_idopont2` attribútum kollekciója egyre nő. A feldolgozás során kipróbált operátorokat az első oszlopból egyesével áthelyezzük a másodikba. Amikor az első kollekció kiürül, de további operátorokra lenne szükségünk, mert nem sikerült behelyezni az órát, akkor kell visszalépnünk. Ilyen esetben már elfogytak az alkalmazható operátorok, így visszagörgetjük a tranzakciót az utolsó mentési pontig.

Előfordulhat, hogy az algoritmus olyan megoldást talál, ahol egyenlőtlen az időpontok prioritása szerinti eloszlás, tehát néhány oktató óráinak a leginkább preferált helyet sikerül megtalálni, míg mások órái a kevésbé előnyben részesített – de nem tiltott – időpontokra kerülnek. Az algoritmus futását lassítja az erre a problémára nyújtható megoldás.
Ez abban különbözik az előbb leírtaktól, hogy szükségünk lesz legalább még egy attribútumra. Így oldható meg a kívánt időpontok prioritás szerinti kettéválasztása (többfelé is oszthatjuk, de akkor további oszlopokra lesz szükségünk a táblában). Ekkor csak a magasabb prioritás-értékekkel rendelkező kollekcióra hajtjuk végre a keresést. Így azonban előfordulhat, hogy nem találunk megoldást. Ha ez bekövetkezik, meg kell ismételnünk a keresést, most már a teljes időpont-kollekcióval. Ha viszont találunk megoldást az első oszlop kollekciója

alapján, az egy minden oktató számára ideálisnak mondható eredmény lesz. Egyelőre ennek a funkciónak nem készült el az implementációja.

Annak a ciklusnak a törzsében, amely a lehetséges időpontokat veszi sorra, meg kell határoznunk a kurzushoz tartozó termek listáját. Az adatbázisból megismerhetjük a kurzusok teremigényét: az előadások általában nagy termekbe kerülnek, de ha kis létszámú a hallgatóságuk, elegendő lehet egy kisebb is.
A terem minősítésére is szükségünk lesz ahhoz, hogy a géptermet igénylő órák ne hagyományos tanterembe kerüljenek.
A termek kollekcióját egy beágyazott ciklussal járjuk végig. Ekkor már adottak a paraméterek (időpont és helyszín), elvégezhetjük a beszúrást az `orarend` táblába. A tábla elsődleges kulcsának megválasztása, valamint a beszúráskor és módosításkor aktiválódó `tr_utkozesvizsgalo` trigger biztosítja az ütközések elkerülését, tehát egyazon időpontban ugyanabba a terembe nem kerülhet két különböző kurzus. A trigger gondoskodik az órák kezdete és vége közötti időintervallumok esetleges átfedésének kizárásáról egy kivétel kiváltásával, amely megegyezik az elsődleges kulcs-integritási megszorítás megsértése esetén kiváltódó kivételes eseménnyel.

Az operátor-alkalmazási előfeltételek ellenőrzésére is ezen a ponton kerül sor. Amint elérjük a `seged` táblában az első gyakorlatot, lekérdezhetjük az előadások időpontjait, amelyek ekkorra már ismertté válnak. Majd megvizsgáljuk, hogy nem áll-e fenn ütközés az éppen behelyezésre váró óra és a szakján és évfolyamán lévő előadások bármelyik órájával. Ha nem teljesül a feltétel, ki kell lépnünk a belső ciklusból, hiszen a terem megváltoztatása nincs hatással az időpontra, amelyre vonatkozólag az ütközéseket figyeljük.
Ha teljesül a feltétel, következhet egy újabb ellenőrzés. Erre csak abban az esetben kerül sor, ha a webes felületről beérkezett sztring harmadik része nem üres. Ekkor meg kell győződnünk arról, hogy a megadott órarend foglalt időpontjaira nem tesszük a soron következő órát.
Ha ezen a feltétel-vizsgálaton is sikeresen túljutottunk, a beszúrás következik. Ezután kerül sor az `s_feltetel` oszlopban megadott feltétel teljesülésének ellenőrzésére. Ha ez a feltétel az adatbázisbeli értékek behelyettesítése után nem teljesül, vissza kell vonnunk az utolsó beszúrás hatását.

Amikor minden megadott feltételnek eleget tesz az óra időpontja és terme, a beszúrást egy `java.sql.Savepoint` objektum létrehozásával, és a mentési pontokat tartalmazó verem tetejére helyezésével „nyugtázzuk”. Akár ezt tesszük, akár visszagörgetjük a tranzakciót egy előző mentési pontig, a már „alkalmazott operátort” a `seged` táblában lévő első kollekcióból át kell helyeznünk a másodikba. Visszalépéskor nincs szükség a második kollekció elemeinek visszahelyezésére az „érvényes” időpontok gyűjteményébe, mert a *`rollback`* utasítás kiadásával az áthelyezések is semmissé válnak, így azonnal a helyes kollekció-állapotokat kapjuk. Visszagörgetés esetén a fő ciklus számlálóját csökkentjük, mivel mindig az aktuális rekordot eggyel megelőző sorra kell visszalépnünk.

Ha minden kurzust, amely a `seged` táblában szerepel, sikeresen elhelyeztünk az `orarend` táblában, elkészült a féléves órarend!

4. Az alkalmazás fejlesztésére vonatkozó tervek

A felhasználókat szerepkörük szerint három csoportra oszthatjuk: tanulmányi adminisztrátorok, oktatók és hallgatók (a korábban említett rendszer-adminisztrátori tevékenységeket a tanulmányi adminisztrátor is elláthatja). Minden szerepkör más-más felhasználói felülettel rendelkezik, amelyek a rendszerbe történő bejelentkezés után érhetőek el.

4.1 Bejelentkezési felület

Bejelentkezéskor a felhasználónak lehetősége van a Neptun-kódja vagy a felhasználói neve (hálózati azonosítója) közül a már megszokottat kiválasztani.
A szerepkörhöz szintén egy lenyíló menü van rendelve, amelyből a felhasználó kiválasztja azt a szerepkörét, amellyel éppen be akar jelentkezni. Egy személyhez több szerepkör is tartozhat: valaki lehet egyszerre adminisztrátor és oktató is. A demonstrátorok oktatóként és hallgatóként is bejelentkezhetnek.
Új felhasználó felvételére jelenleg még nincs lehetőség. Egyelőre kérdéses, hogy a későbbiekben erre felmerülhet-e igény, mivel a tanulmányi rendszer minden lehetséges felhasználó adatait tartalmazza, legyen szó bármelyik szerepkörről. Ha az órarend-készítő alkalmazás kapcsolatban áll a tanulmányi rendszerrel, abból a változásokat is képes nyomon követni, így egy új kollégát elegendő egy helyen, a tanulmányi rendszerben felvenni, aki ezután automatikusan be tud jelentkezni az órarend-készítő rendszerbe is.
A bejelentkezési felület terve a ***4.1 ábrán*** látható. A legördülő menük a piros színnel keretezett részben láthatók lenyitott állapotban.

Belépés után a szerepkörtől függően más-más felületek állnak a felhasználók rendelkezésre.
Az alábbiakban az interfészek által biztosított funkciók bemutatása következik.

4.2 Oktatói felület

Minden szerepkörhöz tartozó felület közös jellemzője a böngésző ablakának bal oldalán megjelenő lista, amely a kiválasztható tevékenységeket tartalmazza. A kattintás után az ablak

jobb oldali részében megjelenik a tétel kifejtése. Ezek mindegyike egy-egy űrlap, amely a bekért információt hivatott a szerver felé továbbítani és feldolgozni.

Az oktatók által, az aktuális félévben tartandó kurzusok kódjai külön lapokon jelennek meg. Az első lap az oktató összes kurzusának egyben történő kezelésére szolgál. Azt az információt, hogy melyik oktató milyen órákat tart a félévben, az adminisztrátor által felvitt adatokból nyeri az alkalmazás, amelyek az `oktatok_kurzusai` nevű táblában tárolódnak. A kurzusok a kódjaikkal jelennek meg a lapfüleken, de ha az adatbázisban ezek össze vannak rendelve a tárgy rövidített nevével, ez képviseli a kurzusokat a weboldalon; sokkal könnyebb ezeket megjegyezni, mint a kurzusok kódjait.

Minden lap esetében megjelenik egy órarend, amelyben az oktató a lehetséges időpontokat adhatja meg a következő módon: minden időponthoz tartozik egy legördülő lista, amelyből egy tételt lehet kiválasztani. Színek és számok egyaránt segítik a tájékozódást, az öt zöld mező mindegyikéhez más-más szám tartozik 1-től 5-ig. Ezek az oktató által preferált időpontok megjelölésére szolgálnak. Belépéskor alaphelyzetben minden időpont szürke színnel és nullás értékkel jelenik meg. Azok az időpontok, amelyeket az oktató nem tesz tiltólistára, de nem is választja ki őket, a lehetséges időpontok közé fognak tartozni 0 prioritással. A piros színhez rendelt -1 érték kiválasztásával tilthatják le azon időbeli helyeket, amelyekbe semmilyen körülmények között nem kerülhet az óra.

Az összes kurzusfülön van lehetőség egyéb igények megadására is. Itt az oktató minden kurzusához beállíthat, illetve megadhat különböző feltételeket. A leggyakoribbakat igyekeztem előre elkészített beviteli mezőkkel és nyomógombokkal ellátni, de lehetőség van további feltételek megadására is. Ekkor egy új ablak nyílik, amelyben a felhasználható operátorok nyomógombok formájában állnak rendelkezésre. Fontos megjegyezni, hogy ezek az adatbázisban történő rögzítésük előtt validálásra kerülnek, mert a későbbiekben erre már nincs lehetőségünk. A beírt formulát szintaktikai ellenőrzésnek vetjük alá, és hiba esetén tájékoztatjuk a felhasználót.

Az oktatói felület és a további feltételek megadására szolgáló űrlap a ***4.2*** és ***4.3 ábrán*** láthatók.

4.3 Adminisztrátori felület

A tanulmányi adminisztrátorok feladata a kurzusok oktatókhoz rendelésétől a termek kezelésén át az órák manipulálásáig nagyon sokrétű. Ennek megfelelően a felhasználói felületük is sokkal több lehetőséget nyújt, mint az oktatói vagy a hallgatói interfész.
A tárgyak, a termek és az órák kezelése további alpontokat tartalmaz, amelyek mindegyikéhez saját űrlap tartozik.
Új tárgy felvitele esetén az adatok egyesével történő megadása mellett kínálkozik egy másik lehetőség is: egy, a megfelelő szintaktikai felépítéssel rendelkező szöveges állomány feltöltése, amely tartalmazhatja mindegyik tárgy jellemzőit. Erre akkor lehet szükség, ha fennakadás történik a hálózati forgalomban, de az adminisztrátor folytatni szeretné a tevékenységét. Amikor ismét helyreáll a hálózat elérhetősége, az időközben felvitt adatok egyszerre történő rögzítésére használható a fájl feltöltésének modulja. (***4.4***, ***4.5*** és ***4.6 ábra***)
Az órák kezelésére szolgáló űrlapokon az adatbázis-táblákon végrehajtandó lekérdezéseket, törléseket és módosításokat eszközölhet az adminisztrátor.
Az ablak bal oldalán lévő *Órák* tétel *Módosítás* alpontját kiválasztva egy olyan felületre juthat a felhasználó, amelyről kényelmesen kezelhető az órák cseréje. A kész órarend utólagos módosításai közül ez fordul elő leggyakrabban. A kettes és hármas cserét előre definiált beviteli mezők és nyomógombok segítik. A további, speciálisabb változtatásokra a lekérdezéseknél van lehetőség, mivel itt módosító SQL-műveletek is használhatók. (Nem csupán lekérdezések.) (***4.7*** és ***4.8 ábra***)
Az adminisztrátorok elvégzendő feladatai csökkennek az alkalmazás folyamatos használata esetén a következő félévek kezdetén, hiszen a tárgyak listája nem változik meg teljesen félévenként, és az egyszer már rögzített adatokat az adatbázis megőrzi, így csak a változásokat kell feldolgozniuk.

4.4 Hallgatói felület

Ez az interfész nem nyújt olyan széles választékot, mint az előzőekben bemutatottak.
A hallgató olyan feltételeket adhat itt meg, amelyek a félévben teljesítendő óráinak időpontjait behatárolják. Így ki tudja választani a felajánlott órarendek közül a legmegfelelőbbet. (***4.9*** és ***4.10 ábra***)

Ennek a modulnak abban rejlik az erőssége, hogy a tanulmányi rendszer és az órarend-készítő alkalmazás összekapcsolásával a tárgyfelvételi időszakban a hallgatónak az általa elfogadott órarend kurzusaira nem szükséges külön jelentkeznie, ez automatizálható a rendszerek kooperációja révén.

Összegzés

Az órarendkészítés sok felsőoktatási intézményben a mai napig kézzel készül. Ebben segítséget nyújthat a tanulmányi rendszerek órarend-szerkesztő modulja, amely tárolja és ellenőrzi a felvitt adatokat. Ezek az eszközök nem biztosítanak hatékony órarend-készítést, csupán a szerkesztésben lehetnek az adminisztrátor segítségére. Diplomamunkámban bemutatott órarend-készítő alkalmazás használatával leegyszerűsödhet az adminisztrátorokra háruló feladat, amelyet a tennivalóktól zsúfolt félévkezdésekkor a kar vagy intézet óráinak összehangolása jelent.

Az alkalmazás két fő részből áll. Az információk összegyűjtéséhez használt vékony kliens, böngészőben megjelenő webes felület képezi a felhasználói interfészt, míg a háttérben adatbázis-műveletek zajlanak. Itt nem csupán a táblákba történő beszúró, módosító és lekérdező operációkról van szó, hanem az órarend összeállítását végző algoritmusról is.
A kész órarendet általában szerver oldalon, adatbázisbeli táblákban tároljuk. A kliensen futó órarendkészítő program működésének befejezésével biztosítani kell az eredmény adatbázisba kerülését. Ha módosítás szükséges, a program nagy valószínűséggel újra kezdi az összeállítást.
Mivel az adatbázis-kezelő rendszerek a deklaratív eszközök (SQL nyelv) mellett lehetőséget biztosítanak a procedurális paradigma használatára, ezért lehetőségünk van az órarend elkészítését magában az adatbázisban végrehajtani. Ennek előnyei közé sorolható a végrehajtási idő rövidülése, és a hálózati adatforgalom kiküszöbölése. A módosítás is egyszerűbb helyben, mint a kliens oldalról.

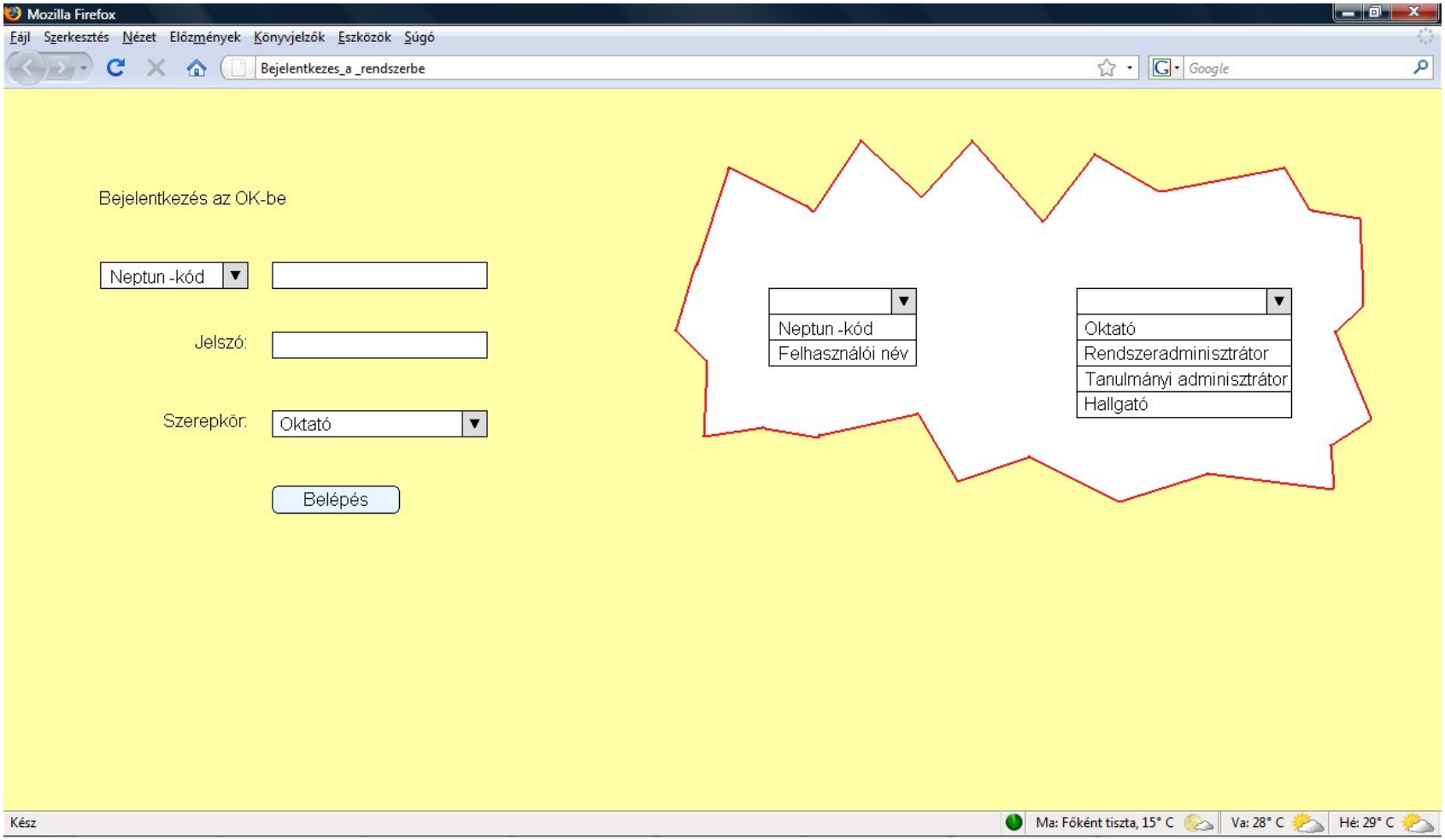
A feladat elvégzéséhez kiválasztott adatbázis-kezelő rendszer az Oracle Database 11g adatbázis-szerver, amely az Oracle cég honlapjáról ingyenesen letölthető. Ez a legújabb változat, amely a procedurális programozást biztosító PL/SQL nyelv mellett saját Java Virtuális Gépet is tartalmaz, ezáltal lehetővé teszi az adatbázisban tárolt eljárások Java nyelvű (1.5-ös verziójú) implementációját.
A feladat megoldásához mind a deklaratív, mind a procedurális eszközöket igénybe vettem.

Az alkalmazás adatbázisban tárolt része a táblák definícióját, egy triggert, egy PL/SQL nyelven írt eljárást, és négy Java osztályt tartalmaz.
Az órarend elkészítéséhez használt visszalépéses megoldáskereső algoritmust az egyik Java nyelvű metódus implementálja. A forráskód további részei ezt segítik az előkészületi munkálatok elvégzésével.

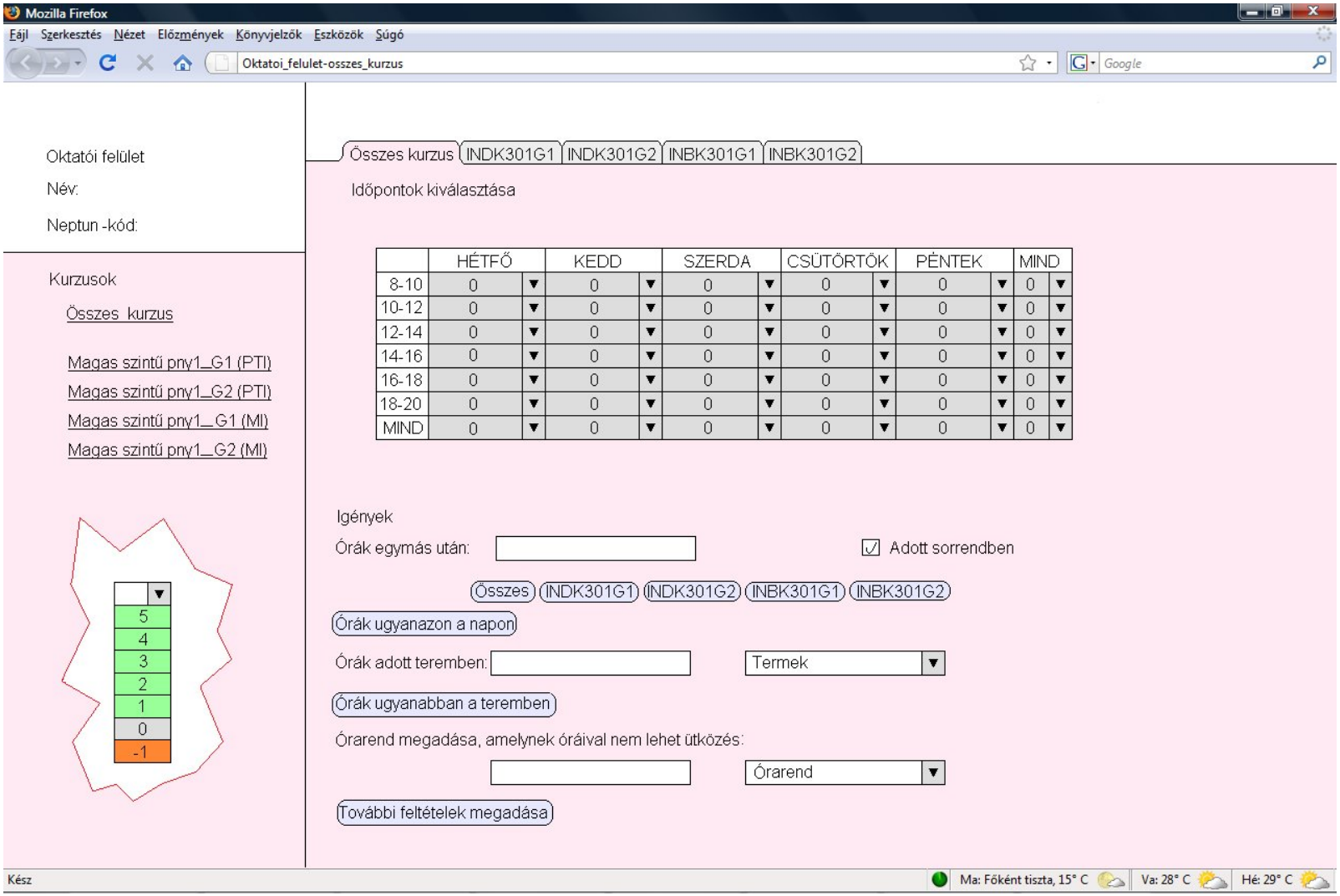
A keresés során nem a memóriában építünk fel egy gráfot, hanem a csúcspontokat reprezentáló rekordok azonnal az `orarend` táblába kerülnek.
A keresési folyamat során egyetlen tranzakció hajtódik végre. A visszalépést ez biztosítja mentési pontok létrehozásával, amelyeket egy verem adatszerkezetben tárolunk. Ha visszalépésre kerül sor, a legutolsó mentési pontig görgetjük vissza a tranzakciót, amelyet a verem tetejéről leemelt `java.sql.Savepoint` objektum reprezentál.
Az operátor a kurzusok órarendbe helyezéséért felelős, paraméterei a lehetséges időpontok. Ezen időbeli és az oktatók által megadott egyéb feltételek szintén táblákban helyezkednek el. A megadott időpontok, az óraütközések elkerülése és a feltétel-vizsgálatok leszűkítik az alkalmazható operátorok számát.

Ha teljes körű a weben keresztül kapott információ, akkor a feladatnak van megoldása, azaz előállítható az órarend. A későbbi módosítások adminisztratív úton kézzel, de ezek gyakori előfordulása esetén automatizáltan is megoldhatók.

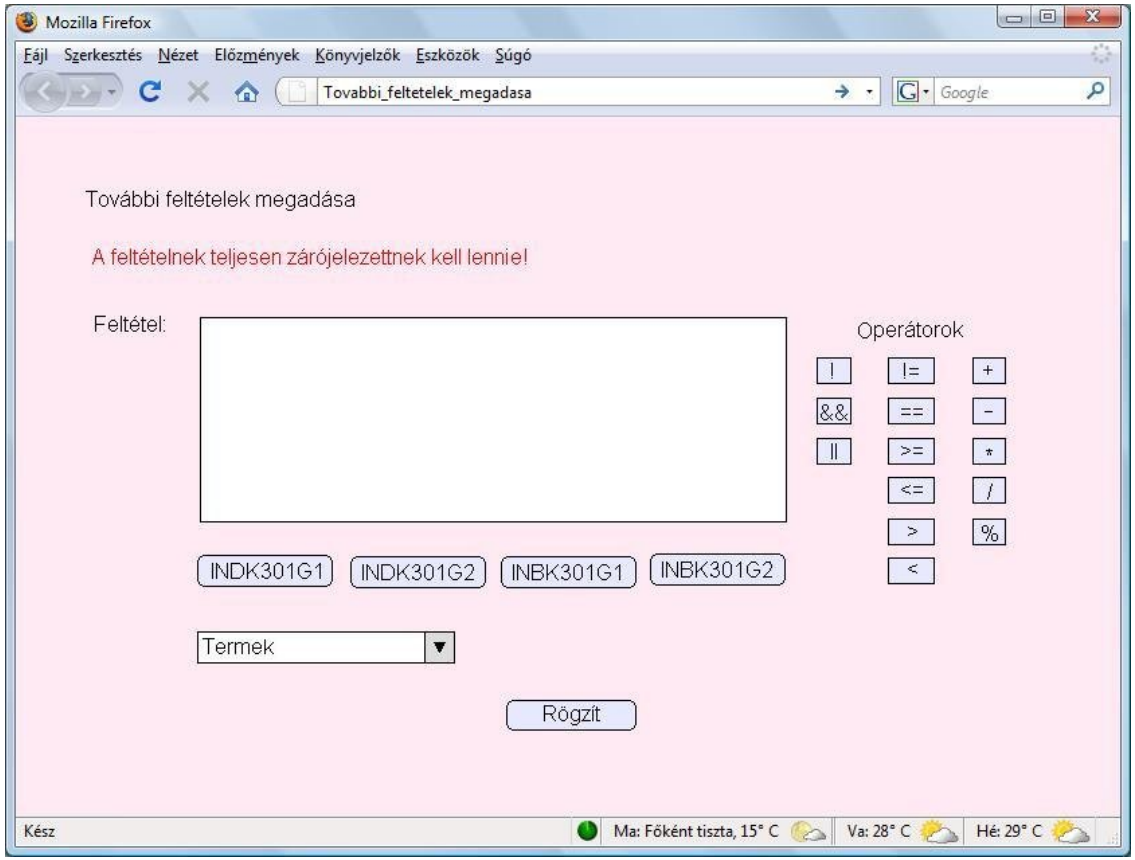
Az alkalmazás továbbfejlesztésével az automatizált órarend-készítés még kényelmesebbé és hatékonyabbá tehető, amely egyre több szolgáltatást képes a felhasználóinak nyújtani.



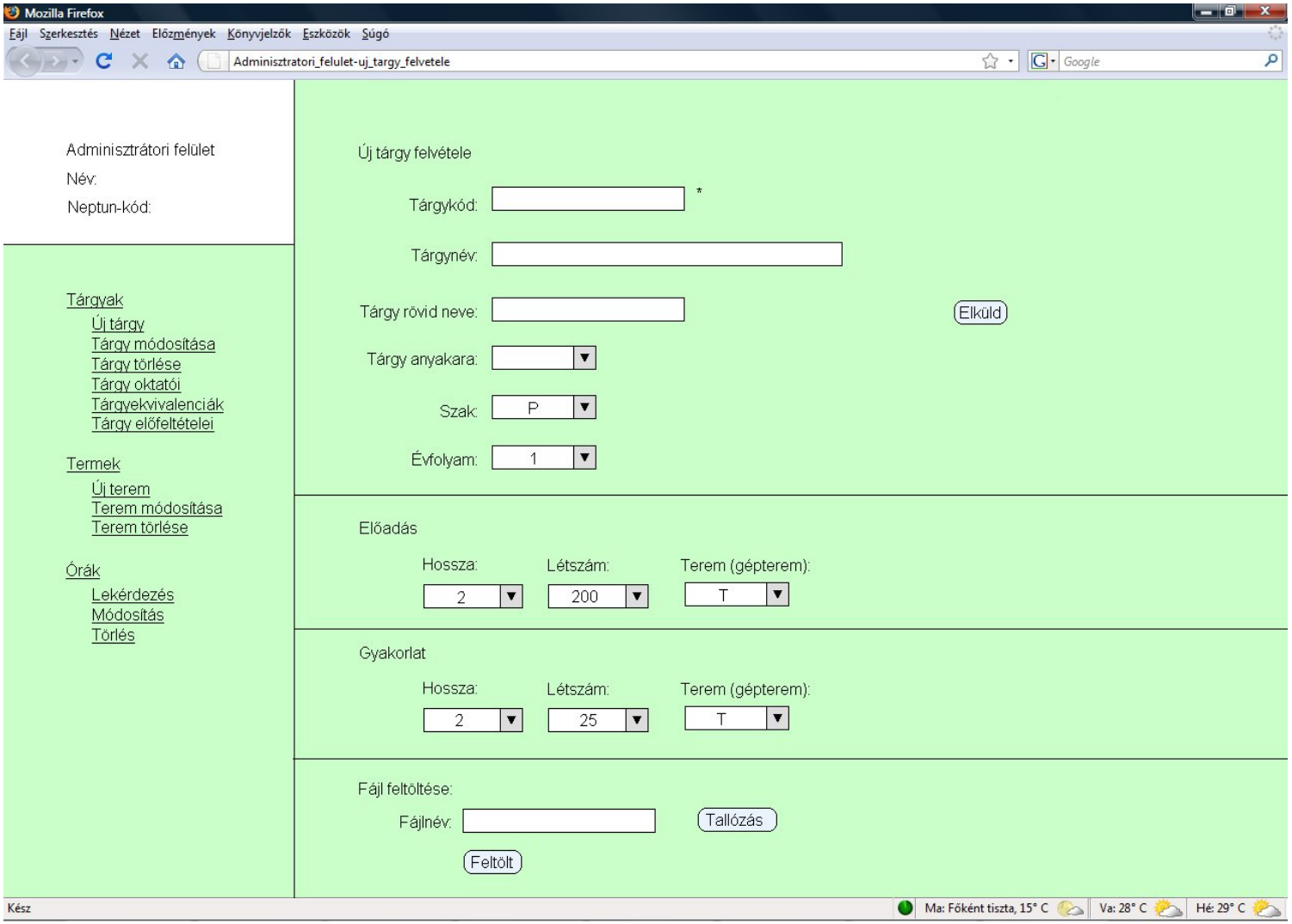
***4.1 ábra**: A bejelentkezési felület a lenyíló menük kifejtésével*



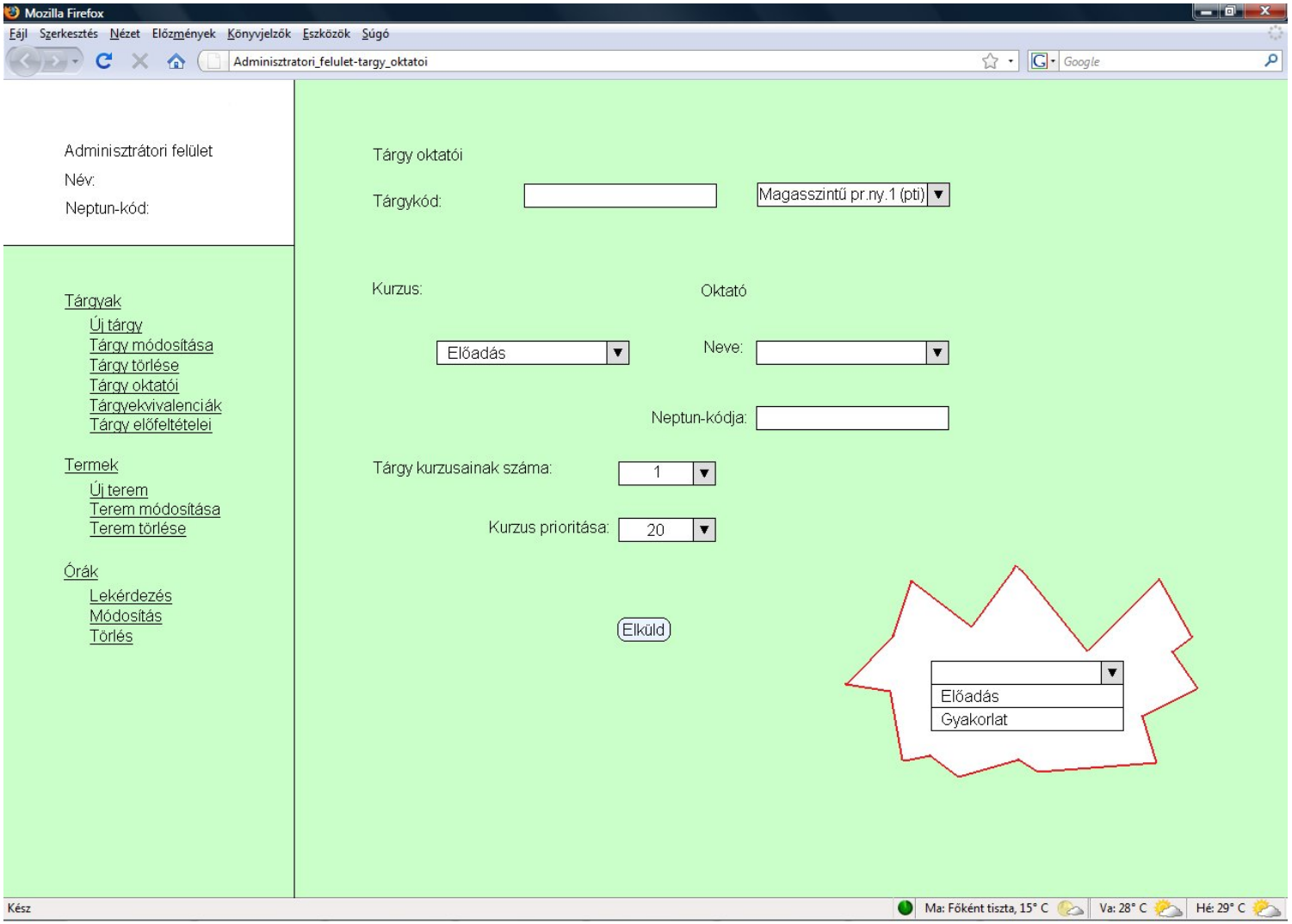
***4.2 ábra**: Az oktatói felület terve*



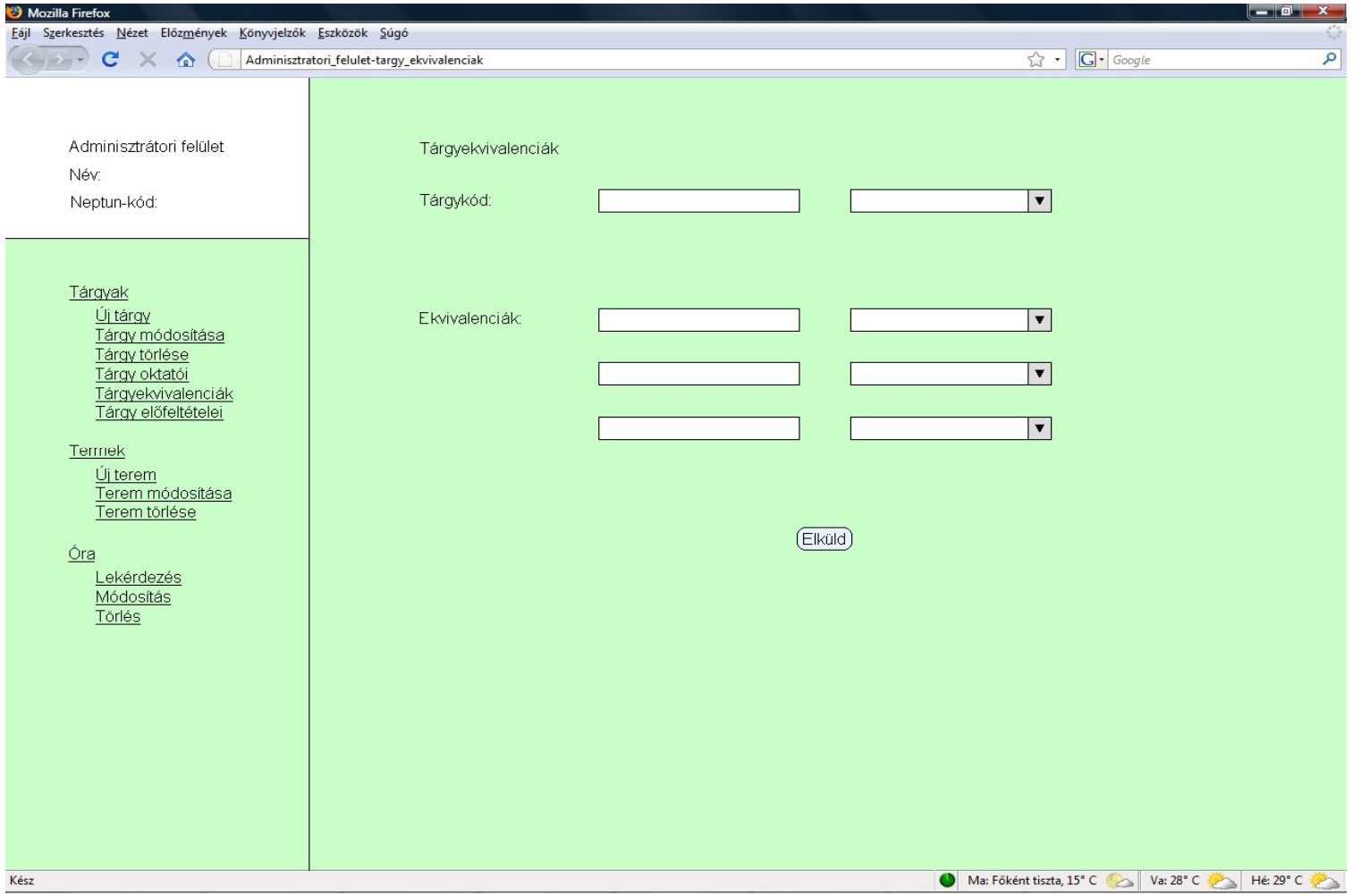
***4.3 ábra**: További feltételek megadására szolgáló interfész*



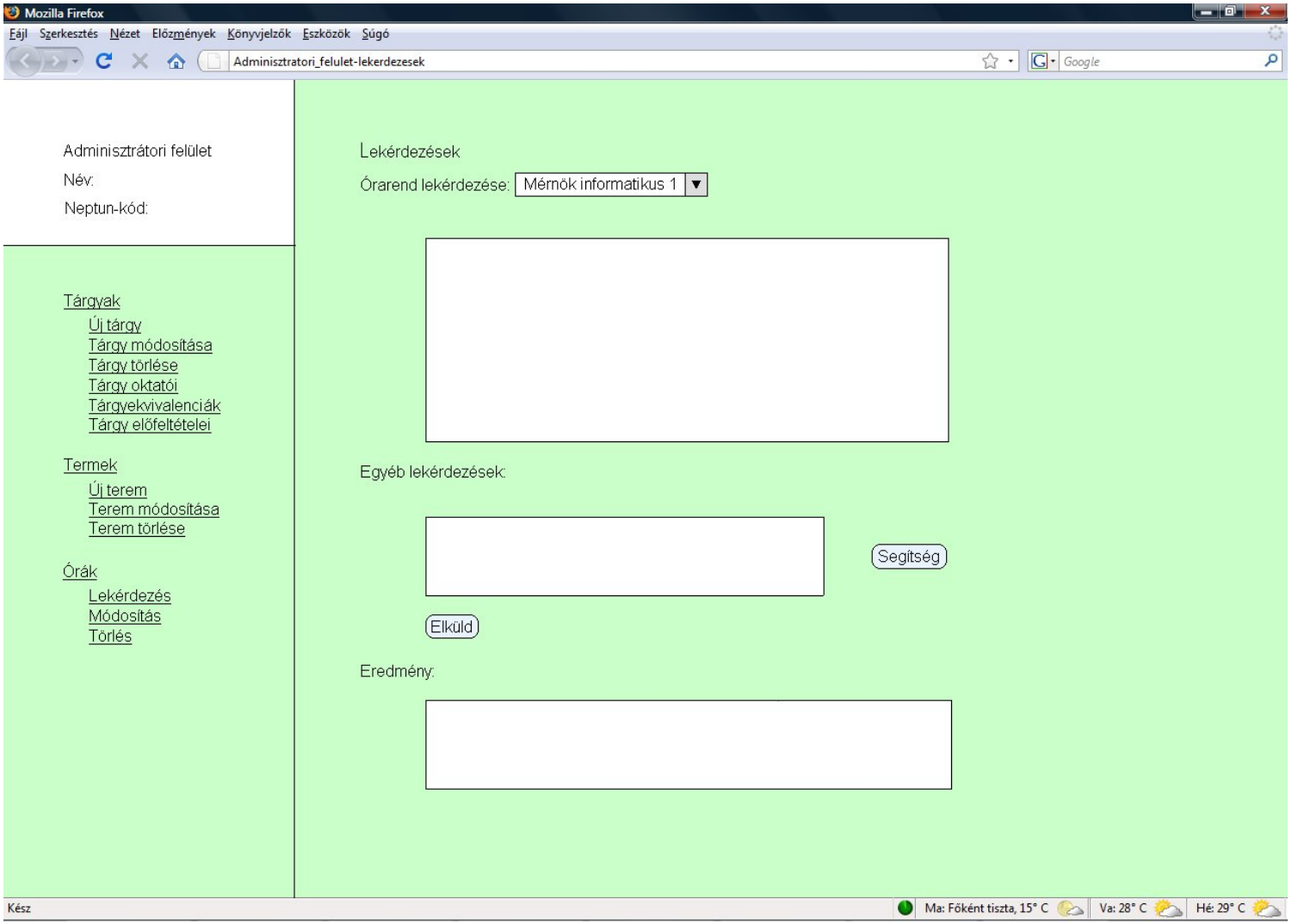
***4.4 ábra**: Új tárgy felvétele a rendszerbe az adminisztrátori felületről*



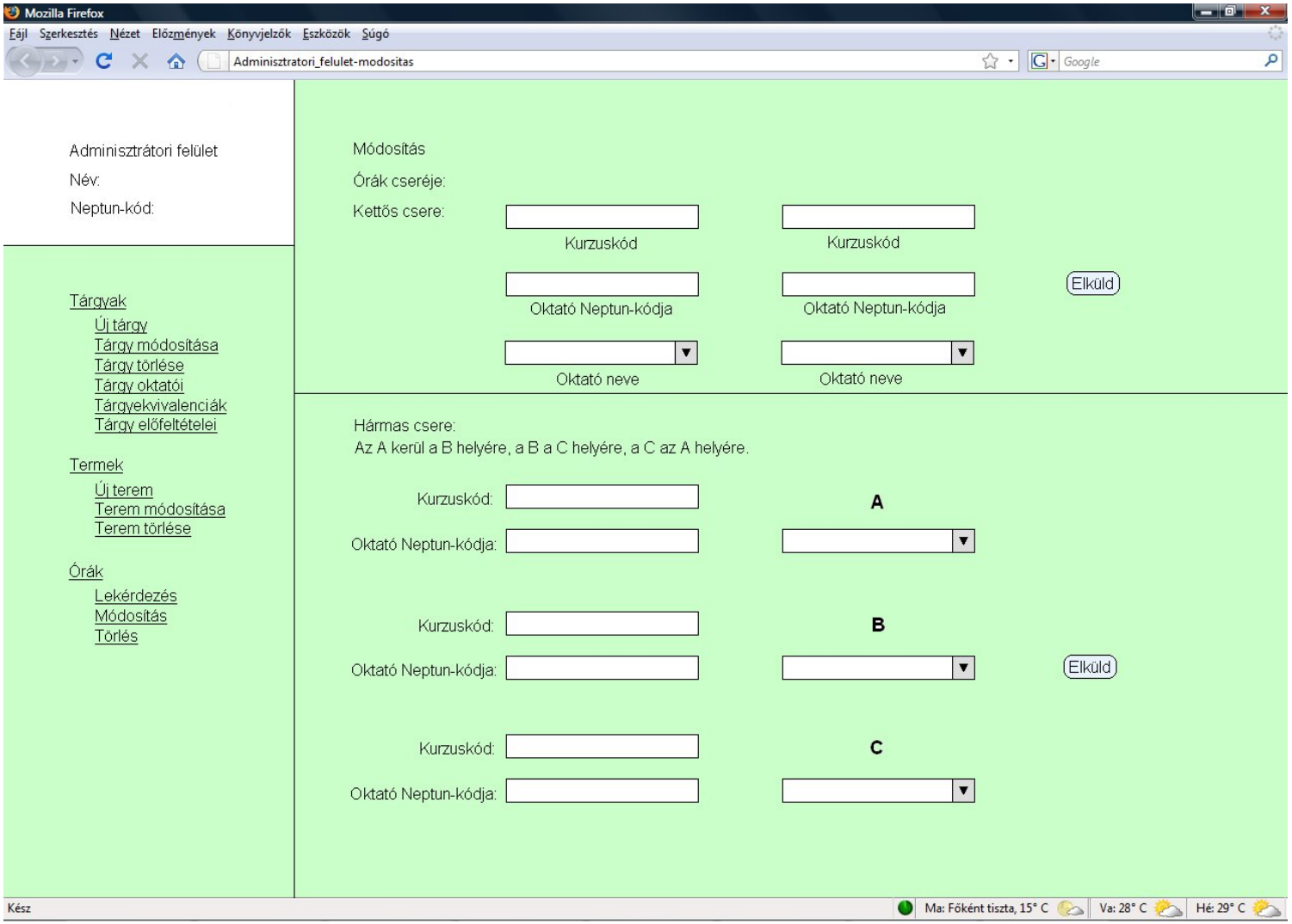
***4.5 ábra**: Oktatók és kurzusok összerendelése az adminisztrátori felületen*



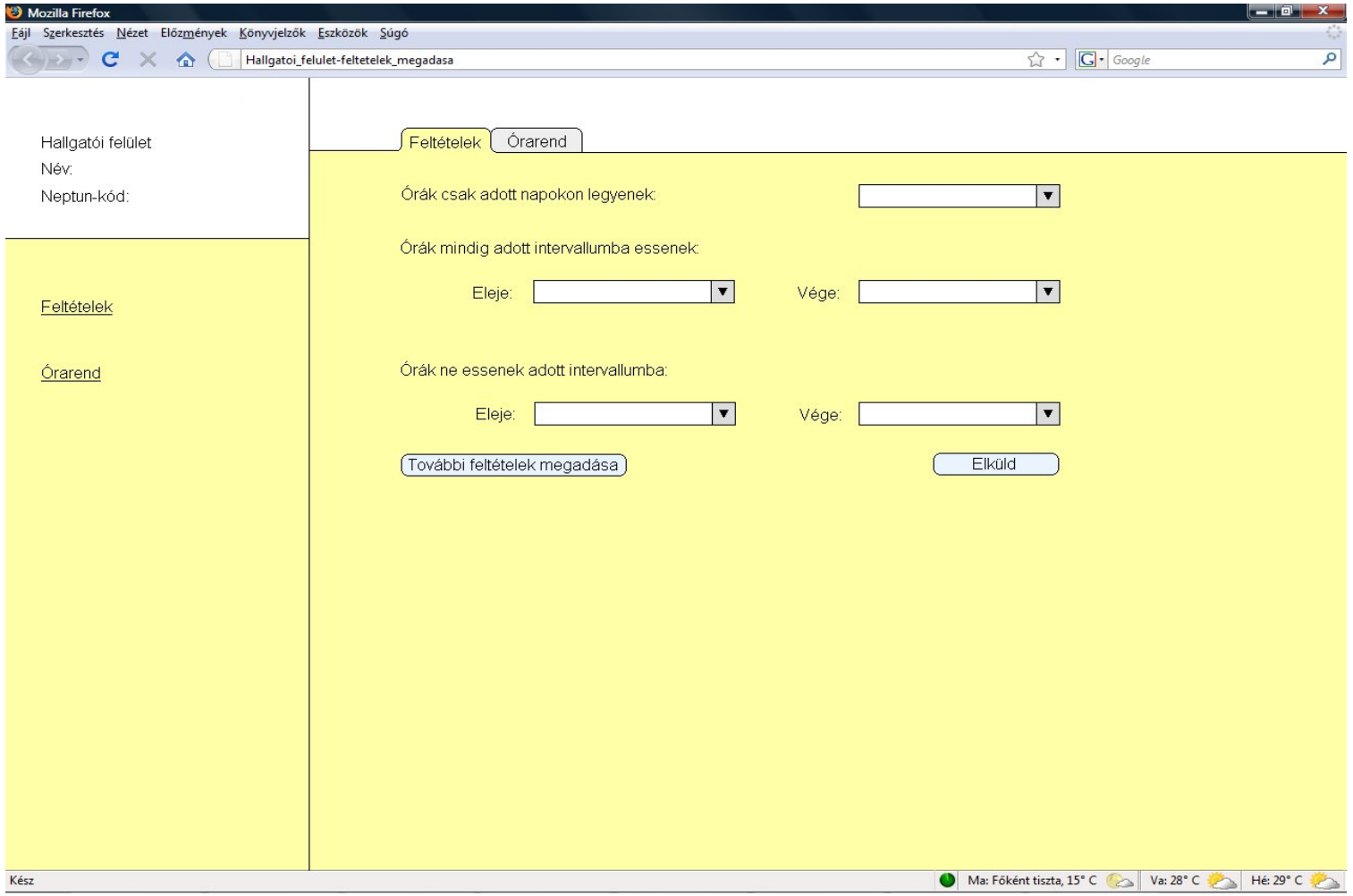
***4.6 ábra**: Tárgy-ekvivalenciák megadása*



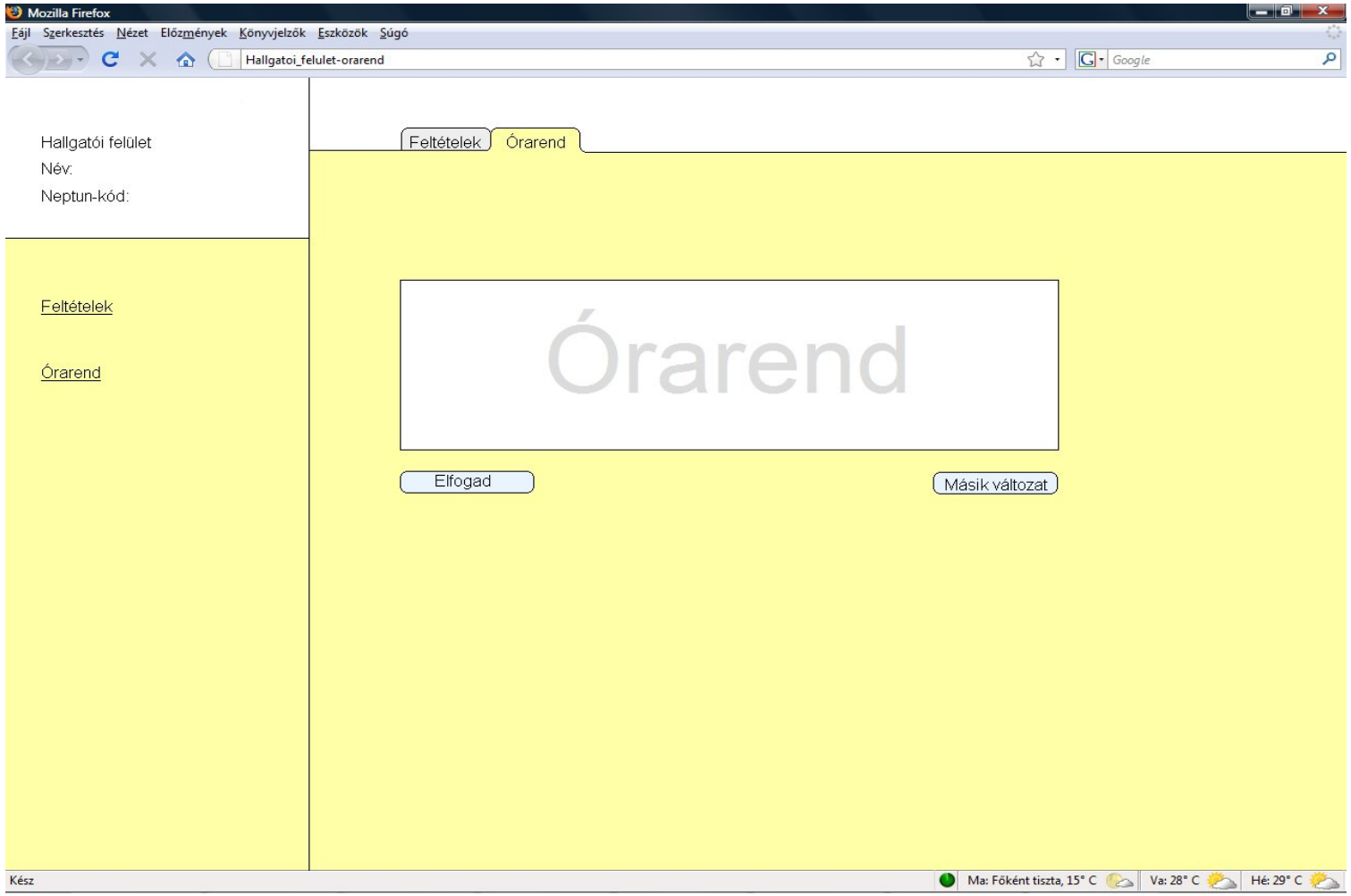
***4.7 ábra**: Az elkészült órarendek lekérdezése*



***4.8 ábra**: Az órák utólagos módosítására szolgáló adminisztrátori felület terve*



***4.9 ábra**: Hallgatói feltételek megadására szolgáló interfész*



***4.10 ábra**: A feltételeknek megfelelő órarend*

IRODALOMJEGYZÉK

[1] KENDE MÁRIA, NAGY ISTVÁN: Oracle példatár, SQL, PL/SQL *Panem*, 2005

[2] GÁBOR ANDRÁS, JUHÁSZ ISTVÁN: PL/SQL programozás, Alkalmazásfejlesztés Oracle 10g-ben *Panem*, 2007

[3] JASON PRICE: SQL, Oracle Database 11g *Oracle Press*, 2008

[4] MICHAEL McLAUGHLIN: PL/SQL programming, Oracle Database 11g *Oracle Press*, 2008

[5] http://www.niif.hu/rendezvenyek/networkshop/97/tartalom/NWS/5/15/index.htm

[6] http://www.ektf.hu/neptun/NEPTUN_ORAREND.pdf

[7] http://info.etr..elte.hu/info/pdfs/orarend_v1.0_040311.pdf

[8] http://www.inf.elte.hu/mot/oktatas/tanterv_orarend/rarendek/Forms/AllItems.aspx

[9] http://www.jomagam.hu/programok/orarend.htm

[10] http://www.idevelopment.info/data/Programming/java/jdbc/PLSQL_and_JDBC/CallPLSQLProc.java

MELLÉKLET

Az adatbázisban használt táblák definíciói

```
create table felhasznalok (
    f_neptun_kod              varchar2(6) primary key
,   f_nev                     varchar2(50) not null
,   f_felh_nev                varchar2(14)
,   f_jelszo                  varchar2(10) not null
,   f_admin                   varchar2(10)
);
create table oktatok_kurzusai (
    ok_neptun_kod             varchar2(6) references felhasznalok(f_neptun_kod)
,   ok_kurzuskod              varchar2(9)
,   ok_prioritas              integer not null
,   ok_kurzus_sorsz           integer
,   primary key (ok_neptun_kod, ok_kurzuskod)
);
create table oktatoi_igenyek (
    oi_neptun_kod             varchar2(6) references felhasznalok(f_neptun_kod)
,   oi_kurzuskod              varchar2(9)
,   oi_webes_sztring          varchar2(800) not null
,   primary key (oi_neptun_kod, oi_kurzuskod)
);
create table targyak (
    tk_targykod               varchar2(7) primary key
,   tk_anyakar                char(1)
,   tk_szak                   varchar2(10) not null
,   tk_evfolyam               integer not null
,   tk_ea_hossz               integer check (tk_ea_hossz > 0 and tk_ea_hossz <= 12)
,   tk_ea_fo                  integer
,   tk_ea_gep_terem           char(1) check (tk_ea_gep_terem in ('G', 'T'))
```

```
,   tk_gy_hossz                integer check (tk_gy_hossz > 0 and tk_gy_hossz < 10)
,   tk_gy_fo                   integer
,   tk_gy_gep_terem            char(1) check (tk_gy_gep_terem in ('G', 'T'))
);
create table targy_ekvivalenciak (
    te_targykod                varchar2(7) references targyak(tk_targykod)
,   te_ekv_targykod            varchar2(7) references targyak(tk_targykod)
,   primary key (te_targykod, te_ekv_targykod)
);
create table targy_elofeltetelei (
    tf_targykod                varchar2(7) references targyak(tk_targykod)
,   tf_elof_targykod           varchar2(7) references targyak(tk_targykod)
,   primary key (tf_targykod, tf_elof_targykod)
);
create table targy_sorszama (
    ts_targykod                varchar2(7) primary key
,   ts_sorszam                 integer not null check (ts_sorszam >= 0)
);
create table termek (
    t_teremkod                 integer primary key
,   t_teremnev                 varchar2(10)
,   t_anyakar                  char(1)
,   t_fo                       integer check (t_fo >= 0)
,   t_gep_terem                char(1) not null check (t_gep_terem in ('G', 'T'))
);

create type t_idopont as object (
    ti_prioritas       integer
,   ti_nap             integer
,   ti_oratol          integer
,   ti_oraig           integer
)
/
```

```
create type t_kollekcio is table of t_idopont
/
create table seged (
    s_sorszam                   integer unique
,   s_neptun_kod                varchar2(6) references felhasznalok(f_neptun_kod)
,   s_kurzuskod                 varchar2(9)
,   s_prior_idopont1            t_kollekcio
,   s_prior_idopont2            t_kollekcio
,   s_feltetel                  varchar2(680)
,   s_fv_orarend                varchar2(20)
,   primary key (s_neptun_kod, s_kurzuskod)
) nested table s_prior_idopont1 store as snt_pi1,
  nested table s_prior_idopont2 store as snt_pi2
/

create table orarend (
    o_teremkod                  integer references termek(t_teremkod)
,   o_nap                       integer check (o_nap >= 1 and o_nap <= 5)
,   o_oratol                    integer check (o_oratol >= 8 and o_oratol < 20)
,   o_oraig                     integer check (o_oraig > 8 and o_oraig <= 20)
,   o_neptun_kod                varchar2(6) references felhasznalok(f_neptun_kod)
,   o_kurzuskod                 varchar2(9) not null
,   primary key (o_teremkod, o_nap, o_oratol, o_oraig)
);
/
```