ORACLE®

Az ORACLE és a web

Haladó ORACLE 9*i* ismeretek

Java az Oracle-ben, Java tárolt alprogramok

XML használatának lehetőségei

Oracle Portal használata weboldalak kialakításánál

Oracle Internet File System

GÁBOR ANDRÁS
GUNDA LÉNÁRD
JUHÁSZ ISTVÁN
KOLLÁR LAJOS
MOHAI GÁBOR
VÁGNER ANIKÓ

ORACLE 9*i* verzióhoz

Gábor András, Gunda Lénárd, Juhász István,
Kollár Lajos, Mohai Gábor, Vágner Anikó

Az Oracle és a web

Haladó Oracle9*i* ismeretek

PANEM

ISBN 963 545 385 X

Ez a könyv az Oktatási Minisztérium támogatásával, a Felsőoktatási Pályázatok Irodája által lebonyolított felsőoktatási tankönyvtámogatási program keretében jelent meg.

A kiadásért felel a Panem Kft. ügyvezetője, Budapest, 2003

Lektorálta: Fazekas Gábor
Tipográfia: Beck Zsuzsa
Borítóterv: Mitró Ákos
Tördelte: SEBE's Bt.

panem@panem.hu
www.panem.hu

Tartalomjegyzék

Előszó

2001-ben megjelent az Oracle adatbázis-kezelő rendszer legújabb verziója, az Oracle9*i*. A könyv megírása idején a verzión belül már a második kiadás is napvilágot látott, azonban a könyv az Oracle9*i* Release 1 alapján készült. A könyvben tárgyalt eszközök használata csak minimálisan tér el a két kiadásban.

Ez a könyv elsősorban *tankönyv*, amely egy gyakorlatorientált felsőoktatási kurzus anyagát tartalmazza. A megértést sok példa szolgálja. A könyv *haladó* ismereteket tartalmaz, amelyek elsajátításához az alábbi informatikai alapismeretek szükségesek:

- a relációs adatmodell fogalmai;
- a relációs adatbázis-kezelő rendszerek alapeszközei;
- a szabvány SQL:1999 nyelv ismerete;
- az Oracle SQL, PL/SQL és SQL*Plus ismerete;
- az eljárásorientált és objektumorientált programozási nyelvek alapeszközeinek fogalmi szintű ismerete;
- programozási gyakorlat a Java nyelven;
- a HTML és XML ismerete;
- a webes technológia ismerete.

Ezek az alapismeretek jórészt megtalálhatók az [1], [2], [4], [5], [6], [7], [8], [9], [10], [25], [29], [31], [36] művekben.

A könyv tankönyvként használható a felsőoktatási intézmények informatika szakjainak speciális kurzusain. A fejezetek egy-egy témát receptszerűen, lépésről lépésre tárgyalnak, sok apró példát, mintát adva az egyes eszközök használatához. A könyv alkalmas önálló feldolgozásra is.

Gyakorló szakemberek az Oracle új verziójának egyes lehetőségeivel ismerkedhetnek meg belőle.

Debrecen, 2002. november

a szerzők

1. FEJEZET

Bevezetés

Az elmúlt években az internetes, intranetes alkalmazások nagyon elterjedtek a világon és egyre inkább az informatikai fejlesztések középpontjába kerültek. Ez új igények megjelenését, új típusú kérdések felmerülését hozta magával. Az adott kihívások megválaszolására dobta piacra 2001-ben adatbázis-kezelőjének új verzióját az Oracle.

Az Oracle9*i* Release 1 és Release 2 tulajdonképpen három termékcsaládot takar. Az egyik a Database Server, ez egy adatbázis-kezelőt és a hozzá tartozó kiszolgáló eszközegyüttest tartalmazza. A másik az Application Server. Ez tulajdonképpen a középső réteget jelentő alkalmazásszerver. A harmadik család a Developer Suite, az a fejlesztőkészlet, amelynek segítségével az adatbázis és alkalmazásszerver alkalmazásai kifejleszthetők.

Könyvünk a Release 1 néhány eszközét ismerteti. A tárgyalt eszközök kiválasztásánál elsődleges szempont a webes alkalmazásokban szükséges ismeretek egy részének csokorba gyűjtése volt.

A könyv a Bevezetésen túl további négy fejezetet tartalmaz. Ezek a fejezetek egymástól függetlenül feldolgozhatók, önállóan is megállják a helyüket. Ez, és a tárgyalt eszközök használatának különbözősége indokolja azt, hogy a fejezetek stílusa, szerkesztése bizonyos pontokan különbözik egymástól.

Könyvünkben igyekeztünk magyar terminológiát használni, de néhány esetben maradtunk az angol szónál (például *cookie*, *XML tag*, *socket*), mert ezekre egyáltalán nincs elfogadott magyar változat. A terület újdonsága miatt lehet, hogy az általunk választott magyar kifejezések egy része sem fog meghonosodni. Sőt még az is előfordul bizonyára, hogy a könyvben inkonzisztens módon használunk bizonyos fogalmakat. Ezért a Tisztelt Olvasó elnézését kérjük és szívesen fogadunk minden megjegyzést, javítást, bírálatot.

Ezek után lássuk az egyes fejezetek rövid áttekintését. A második fejezetben az Oracle JVM legalapvetőbb jellemzőit, lehetőségeit tárgyaljuk. Szó esik a Java forrás-, osztály- és erőforrás-állományok adatbázisba való betöltéséről, metódusok publikálásáról, az operációs rendszer erőforrásainak kezeléséről és a jogosultságokról. A fejezet második részében a Java tárolt alprogramok használatát tárgyaljuk.

A harmadik fejezetben bemutatjuk, hogyan támogatja az Oracle9*i* az XML nyelvet. A fejezet első szakaszában tárgyaljuk az XML-dokumentumok adatbázisban való

tárolására szolgáló XMLType típust, amelynek műveleteivel a dokumentumban tárolt információk kinyerhetők. A DBMS_XMLGEN csomag bemutatásán keresztül megismertetjük azon alprogramokat, amelyek egy tetszőleges lekérdezés eredményhalmazának XML formátumba alakítására szolgálnak. Külön kitérünk arra, hogy miképpen állíthatunk elő lapozható eredményeket az adatbázistáblákban, illetve nézetekben tárolt adatok alapján. A SYS_XMLGEN soronkénti SQL-függvény ismertetésével bemutatjuk, hogy mit kell tenni abban az esetben, ha nem a teljes eredményhalmazból, hanem csupán egyetlen eredménysorból kell XML formátumú kimenetet előállítani. A SYS_XMLAGG oszlopfüggvénnyel a csoportosítást tartalmazó lekérdezések XML-be alakítása végezhető. Az alkalmazások migrációja során fontos feladat, hogy ne csak az adatbázisban tárolt adatokat tudjuk XML-be alakítani, de az XML formátumú adatokból tudjunk (objektum)relációs táblákba és nézetekbe is beszúrni. Erre szolgálnak az első szakasz végén bemutatott TABLE függvények. A fejezet további részében egy, a webes tartalomszolgáltatást megkönnyítő eszközről, az XSQL keretrendszerről írunk, amely parametrizált SQL-lekérdezések eredményeit XML-ben megkapva (azt esetleg transzformációkkal tovább alakítva), azokat azonnal egy webszerveren publikálja.

Az Oracle Portal az Oracle9*i* Application Server része, mellyel könnyen, gyorsan építhetjük fel vállalati portáljainkat. Egyaránt alkalmas intranetes és internetes alkalmazások fejlesztésére. A negyedik fejezetben megismerjük a Portal-rendszer építőelemeit. Példákon keresztül mutatjuk be, hogyan használhatjuk a Portal oldalait összetett weboldalak készítésére, sokrétű információ megjelenítésére. Megismerjük az adatterületeteket, amik összetartozó adatok rendszerezett tárolására és elérésére szolgálnak. Egy példaalkalmazáson bemutatjuk, hogyan készíthetünk varázslók segítségével komponenseket az adatbázisban tárolt dinamikus adatok megjelenítésére. Foglalkozunk a Portal felhasználókezelésével és demonstráljuk, hogyan egészíthető ki, fejleszthető tovább a Portal eszközkészlete saját alkalmazásainkkal a PDK keretrendszer felhasználásával.

Az ötödik fejezet az Oracle *i*FS-ről szól. Az Oracle Internet File System az adatbázisok állománykezelő rendszere. Az állományokat egy relációs adatbázisban, nem pedig egy helyi merevlemezes tárolón tárolja, ezért számos olyan funkció áll rendelkezésünkre, amelyek a szabványos állománykezelő rendszerekkel nem valósíthatók meg. Az Oracle *i*FS lehetővé teszi, hogy vagy egy Windowsos vagy egy webes felületen dolgozzunk, amelyek kinézete bár eltérő, mégis ugyanazon alapműveletek végrehajtására alkalmasak. A Windowsos kezelőfelületen keresztül az Oracle *i*FS állományokat és mappákat a Microsoft Windows 95, 98, 2000 és NT operációs rendszereknél megszokott módon láthatjuk. Az Oracle *i*FS webes interfészen keresztüli eléréséhez nincs szükségünk semmilyen szoftverre vagy segédeszközre. Mindössze a Netscape Navigator 4.7 vagy az Internet Explorer 5 (vagy ezek későbbi verziószámú) böngészőkbe be kell gépelni az *i*FS kiszolgáló címét. Az Oracle *i*FS átfogó és rugalmas biztonsági modellt biztosít a jogosulatlan hozzáférések megakadályozására. Egységes hozzáférési mechanizmust alkalmaz, függetlenül a felhasználó személyétől, illetve attól, hogy milyen protokollt és eszközt használ. Az Oracle *i*FS támogatja XML-dokumentumok tárolását – elemző és leképező eszközeivel a felhasználók XML-állo-

mányaikat strukturált vagy strukturálatlan adatokként tárolhatják az adatbázisban. Az Oracle az első olyan adatbázis-kezelő, amely XML-dokumentumok tárolásához alapvető eszközrendszerrel rendelkezik, képes automatikusan feltölteni, generálni és transzformálni XML-állományokat magában az adatbázis motorjában.

A könyvben alkalmazott jelölések, konvenciók

Az utasítások és eszközök használatához általában formálisan megadjuk azok szintaxisát. A formális leírásnál az alábbi jelöléseket használjuk:

- A *terminálisok* Courier betűtípussal jelennek meg (például `loadjava`).
- A *nemterminálisok* dőlt kisbetűs formában szerepelnek (például *típusnév*). Ha a nemterminális megnevezése több szóból áll, a szavak közé aláhúzás (_) kerül (például *metódus_teljes_neve*).
- Az *egyéb szimbólumok* az írásképükkel jelennek meg (például :=).
- Az *alternatívákat* függőleges vonal (|) választja el (például `MAP|ORDER`).
- Az *opcionális elemek* szögletes zárójelben ([ ]) állnak (például [*@ab_url*]).
- A *kötelezően megadandó alternatívákat* kapcsos zárójelek ({}) zárják közre
- (például {`-user|-u`}).
- Az *iteráció* jelölésére a három pont (...) szolgál (például *oszlop*[,*oszlop*]...).

2. FEJEZET

Az Oracle9*i* és a Java

Az Oracle9*i* egy száz százalékig kompatibilis Java 2 eszközrendszert tartalmaz, amely a JVM megvalósításán túl a következő programozási környezetet nyújtja:

1. Java tárolt alprogramok hozhatók létre, a PL/SQL tárolt alprogramokhoz teljesen hasonló módon, a PL/SQL-hez integráltan. Egy Java tárolt alprogram hívható PL/SQL-csomagból; PL/SQL-alprogramok hívhatók Java tárolt alprogramokból. Egy Java tárolt alprogram hívható közvetlenül az SQL*Plusból vagy közvetett módon egy trigger törzséből.
2. Az SQL-adatok az SQLJ és JDBC interfészeken keresztül érhetők el. Mindkét API elérhető a kliens és szerver oldalon egyaránt. A JDBC segítségével kétrétegű kliens-szerver alkalmazások fejleszthetők. Az SQLJ tulajdonképpen a beágyazott statikus SQL Java esetén.
   A JDBC egy adatbázis-elérési protokoll, amely segítségével kapcsolódni lehet az adatbázishoz, majd SQL-utasításokat lehet előkészíteni és végrehajtani. Az SQLJ egy szabványos programozási interfész, amely egyszerűbb, de erőteljesebb mint a JDBC. Segítségével Java-programokba SQL-utasítások ágyazhatók, amelyeket egy előfordító dolgoz fel és alakít át olyan standard Java-kódokká, amelyek JDBC-hívásokat tartalmaznak. Alkalmazható kliens oldali és középrétegbeli alkalmazások fejlesztésénél. Használható tárolt alprogramokban, triggerekben, metódusokban, továbbá CORBA és EJB környezetben is.
3. A Java-szervletek és a JavaServer Pages (JSP) segítségével lehetőséget nyújt dinamikus HTML-oldalak létrehozására.
   A szervlet alkalmas a HTML- (XML) és Java-alkalmazások közötti kérések kezelésére web környezetben.
   A JSP a webszerver által futtatott webalkalmazás outputjaként előálló oldalak tartalmának dinamikus generálását teszi lehetővé. Egy JSP-oldal futtatás előtt Java-szervletté alakítódik át, amely HTTP-kéréseket dolgoz fel és válaszokat generál. A JSP-oldalak és a szervletek között teljes körű az együttműködés: egy JSP-lap fogadni tudja egy szervlet outputját és továbbítani tudja azt egy másik szervletnek, és megfordítva.
4. Az Oracle9*i* tartalmaz egy beépített CORBA 2.0 ORB-t és támogatja az Enterprise Java Beanst (EJB). Ezáltal lehetővé válik elosztott alkalmazások fejlesztése. A

CORBA és az EJB lehetővé teszi Java-komponensek és az alkalmazói logika szétosztását, a kliens, a középréteg és a szerver között. Az EJB nagy méretű komponenskönyvtárat bocsát a fejlesztők rendelkezésére, amely segítségével az alkalmazások fejlesztése egyszerűbbé, gyorsabbá és biztonságosabbá válik.

5. Az Oracle9*i* több fejlesztőeszközt és segédprogramot tartalmaz, amelyek a nyílt internet- és Java-szabványokkal, illetve protokollokkal teljesen kompatibilis fejlesztést segítik. Ezek mindegyike Javában íródott. A JDeveloper olyan integrált fejlesztői környezet, amely elsősorban a tárolt alprogramok írását és az EJB használatát segíti.

Könyvünk jelen fejezetében az általános áttekintésen túlmenően a Java tárolt alprogramok használatára helyezzük a hangsúlyt. A Javával kapcsolatos további ismereteket a következő Oracle dokumentációk tartalmazzák:

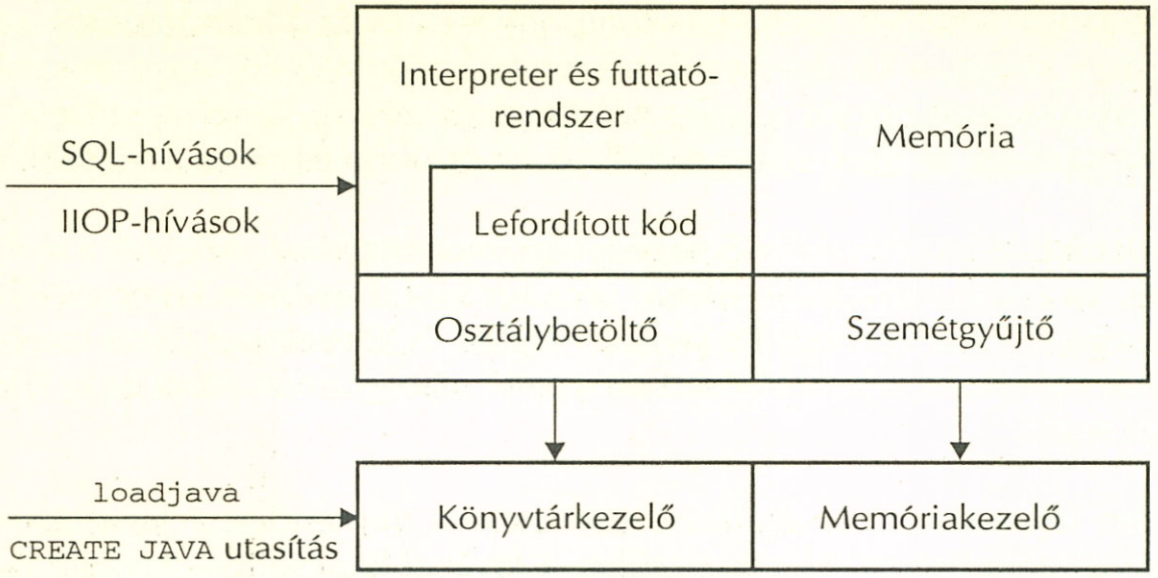
- JDBC – Oracle9*i JDBC Developer's Guide and Reference*
- SQLJ – Oracle9*i SQLJ Developer's Guide and Reference*
- szervletek – Oracle9*i Servlet Engine Developer's Guide*
- JSP – Oracle9*i JSP Support for JavaServer Pages Developer's Guide and Reference*
- CORBA – Oracle9*i CORBA Developer's Guide and Reference*
- EJB – Oracle9*i Enterprise JavaBeans Developer's Guide and Reference*

2.1. Java-alkalmazások készítése

Az Oracle9*i* szabványos Java-alkalmazások futtatását teszi lehetővé. Ugyanakkor viszont, az adatbázis-szerverrel való integráltság miatt, az Oracle környezet több ponton eltér a szokásos Java-környezetektől. Ebben a fejezetben nagy vonalakban áttekintjük azt, hogyan lehet az Oracle9*i*-ben Java-alkalmazásokat írni, telepíteni, futtatni.

2.1.1. Munkamenetek és Java-alkalmazások

Az Oracle9*i*-be ágyazott JVM segítségével a Java-alkalmazások *betölthetők* az adatbázisba, *publikálhatók* az interfészeik és aztán az alkalmazások egy adatbázis-munkameneten belül *lefuttathatók*. Tehát egy Java-alkalmazás mindig egy adatbázis-munkamenethez kötődik. A munkameneten belül minden kliens saját Java-környezettel rendelkezik, ez különbözik minden más kliens környezetétől. A kliens úgy látja, hogy saját, egyedi JVM-el rendelkezik, és a szemétgyűjtögetés, memóriakezelés is külön-külön valósul meg. Az Oracle JVM fő komponenseit mutatja a 2.1. ábra.



2.1. ábra. *Az Oracle JVM fő komponensei*

Egy adatbázis munkamenete befejeződik ha,

- az `oracle.aurora.mts.session.Session.THIS_SESSION().endSession()` metódus meghívásra kerül egy Java-alkalmazásban,
- CORBA vagy EJB esetén a munkamenet ideje lejár,
- a kliens más módon befejezteti azt.

A szabványos Java-környezetekben egy alkalmazást úgy tudunk elindítani, hogy megadunk egy olyan osztálynevet, amely tartalmazza a `main()` metódust. A Java-interpreter ekkor betölti az osztályt és átadja a vezérlést a `main()` metódusnak. Adatbázis-környezetben azonban egy Java-alkalmazás belépési pontja nem kötelezően a `main()` metódus, hanem lehet bármelyik statikus metódusa a betöltött osztálynak. A belépési pontot explicit módon meg lehet adni (l. 2.2.4. alfejezet).

2.1.2. Java az adatbázisban

Az Oracle a Java-forrásállományt, a lefordított bináris állományt és az alkalmazások futási idejében használt adatállományokat sémaobjektumokként (mint *forrás-*, *osztály-* és *erőforrás*-objektumok) kezeli és tárolja őket az adatbázisban. Az CLASSPATH helyett pedig egy ún. *feloldót* használ, amely segítségével meg lehet adni azokat a sémákat, ahol a Java-sémaobjektumokat keresni kell. Egy Java-forrásállományt az alábbi módokon fordíthatunk le:

- explicit módon meghívunk egy Java-fordítót (például `javac`),
- a fordítást az adatbázis végzi betöltés közben a `loadjava` (l. 2.1.3. alfejezet) segítségével,
- a fordítás dinamikusan, futási időben történik.

A Java-forráskódot hagyományos Java-fordítóval lefordítva, a bináris állományt szokás betölteni az adatbázisba. Tesszük ezt általában azért, mert a forráskód kényelmesebben manipulálható a megszokott Java-környezetben, mint az adatbázisban.

Ha a fordítást a `loadjava` segítségével végezzük, akkor a következő történik:

- a forrásállomány forrásobjektumként betöltődik,
- a forrásállomány lefordításra kerül,
- a forrásállományban található minden osztályhoz létrejön egy osztály sémaobjektum.

Betölthető a forrásállomány az adatbázisba anélkül is, hogy lefordításra kerülne. Ekkor csak a forrásobjektum jön létre. Az Oracle9*i* ekkor a forrásállományt automatikusan fordítja futási időben, amikor egy osztályra hivatkozás történik.

A második és harmadik esetben a fordítási hibák a USER_ERRORS nézetben kerülnek letárolásra.

A fordítási opciókat vagy a `loadjava` opcióiként, vagy pedig a JAVA$OPTIONS táblában adhatjuk meg. A táblában megadott opciókat a `loadjava` opciói felülértelmezik. A tábla minden sora tartalmazza azon sémaobjektum nevét, amelyre a beállítást megadjuk, magát a fordítási opció megnevezését és az opció értékét.

A tábla sorait a következő alprogramokkal tudjuk kezelni, ezek a DBMS_JAVA csomagban találhatók:

- `PROCEDURE set_compiler_option (name VARCHAR2, option VARCHAR2, value VARCHAR2)` – Bevisz egy új sort a táblába és létrehozza azt, ha még nem létezik.
- `FUNCTION get_compiler_option (name VRACHAR2, option VARCHAR2) RETURNS VARCHAR2` – Lekéri egy adott sor adott opciójának értékét.
- `PROCEDURE reset_compiler_option (name VARCHAR2, option VARCHAR2)` – Törli az adott sor opciójának beállítását.

A `name` paraméter értéke egy Java-csomag neve, egy teljesen minősített osztály neve vagy egy üres sztring lehet. Amikor a fordító végignézi a táblát az adott sémaobjektum fordítási opcióját keresve, azt a sort fogja választani, amelyik a leginkább illeszkedik a sémaobjektum teljes minősített nevéhez. Az üres sztring minden sémaobjektum nevéhez illeszkedik.

Az `option` paraméter `on-line`, `encoding`, `debug` lehet.

Az Oracle9*i* a Java esetén is automatikusan kezeli a függőségeket, vagyis egy forrásprogramot (amennyiben a forrás a szerveren van) újrafordít, ha azon forrásobjektumok vagy osztályobjektumok megváltoznak, amelyektől a forrásprogram függ.

A hagyományos Java virtuális gépek az egy adott osztály által hivatkozott más osztályokat, ZIP- és JAR-állományokat a CLASSPATH-ban megadott könyvtárakban keresik. Ezzel szemben az Oracle9*i* JVM, mint már említettük, egy feloldót használ. Minden osztály az adatbázis egy sémaobjektumaként jelenik meg, ezért a feloldó számára meg kell adnunk azokat a sémákat, ahol a hivatkozott osztályokat keresni

kell. A sémák ezen listáját *feloldó specifikációnak* nevezzük. A klasszikus Java CLASSPATH globális minden osztály számára. A feloldó specifikáció viszont osztályonként létezik az Oracle9*i*-ben. Az Oracle9*i* JVM osztályai pedig PUBLIC minősítésűek.

A feloldó az osztályok közötti függőségek alapján minden osztályt érvényesnek vagy érvénytelennek jelöl meg működése során. Ha egy adatbázisba betöltött osztály egy olyan osztályra hivatkozik, amely nem található a megadott sémák egyikében sem, akkor érvénytelen lesz. Az érvénytelen osztály használata futási időben a CLASS NOT FOUND kivételt váltja ki.

Az érvénytelen sémaobjektumtól függő osztályok maguk is érvénytelenek lesznek.

Az Oracle9*i* JVM biztosít egy alapértelmezett feloldót, amely először a létrehozó sémájában, majd a PUBLIC osztályok között keres. Ha a más sémákban való keresést is lehetővé akarjuk tenni, akkor saját feloldó specifikációt kell létrehozni. Részleteket lásd a 2.1.3. alfejezetben.

Ha egy osztályt használni akarunk, akkor függetlenül fordítási módjától, azt és a szükséges hivatkozott osztályokat és erőforrásokat be kell tölteni az adatbázisba. Ehhez is a `loadjava` használható, amely különböző típusú állományokkal kapcsolatosan a következő tevékenységeket hajtja végre:

`java` forrásállomány	1. Létrehoz egy forrás sémaobjektumot a megadott vagy a saját sémában. 2. Betölti a forrásállomány tartalmát a sémába. 3. Létrehoz a forrásállomány minden egyes osztályához egy osztály sémaobjektumot. 4. Ha meg van adva a `-resolve` opció, akkor: *a)* Lefordítja a forrásobjektumot. *b)* Feloldja az osztályhivatkozásokat. *c)* A lefordított osztályokat tárolja az osztály sémaobjektumokban.
`sqlj` forrásállomány	1. Létrehoz egy forrás sémaobjektumot a megadott vagy a saját sémában. 2. Betölti a forrásállomány tartalmát a sémába. 3. A forrásállomány minden osztályához és erőforrásához létrehoz egy osztály sémaobjektumot. 4. Ha meg van adva a `-resolve` opció, akkor: *a)* Lefordítja a forrásobjektumot. *b)* A lefordított osztályokat tárolja az osztály sémaobjektumokban. *c)* Tárolja a profilt egy `.ser` erőforrás sémaobjektumban.
`class` állomány	1. Létrehoz egy osztály sémaobjektumot a megadott vagy a saját sémában. 2. Betölti az osztályállományt a sémaobjektumba. 3. Ha meg van adva a `-resolve` opció, akkor feloldja az osztályhivatkozásokat és ellenőrzi a függőségeket.

`properties` erőforrás-állomány	1. Létrehoz egy erőforrás sémaobjektumot a megadott vagy a saját sémában. 2. Betölti az erőforrás-állományt a sémaobjektumba.
`ser` SQLJ profil	1. Létrehoz egy erőforrás sémaobjektumot a megadott vagy a saját sémában. 2. Betölti az erőforrás-állományt a sémaobjektumba.

2.1.3. A `loadjava` eszköz

A `loadjava` forrás, osztály és erőforrás sémaobjektumok létrehozására és betöltésére való. használatához a saját sémába való osztálybetöltéshez a CREATE PROCEDURE és CREATE TABLE, a más sémába való osztálybetöltéshez a CREATE ANY PROCEDURE és CREATE ANY TABLE privilégiumok szükségesek.

A `loadjava` meghívható vagy a parancssorból (a továbbiakban csak ezzel foglalkozunk) vagy a DBMS_JAVA csomag `loadjava` eljárásaként.

A `loadjava` tud fogadni olyan JAR- vagy ZIP-állományokat is, amelyek forrás- és erőforrás- vagy osztály- és erőforrás-állományokat tartalmaznak. Egyszerre forrás- és osztályállomány nem tölthető be az adatbázisba. JAR- és ZIP-sémaobjektum nem létezik. A `loadjava` megnyitja a JAR- vagy ZIP-állományt és tagjait egyenként tölti be. Ha valamely állomány tartalma a legutolsó betöltés óta nem változott, akkor az nem kerül újra betöltésre.

A nem változott állományok felismerésére a `loadjava` minden sémában felépít egy ún. *kivonat* táblát. Ez a felhasználók számára láthatatlan és minden állományhoz kapcsolódóan tartalmazza az állomány tartalmának egy hash alapú rövid reprezentációját. Ha egy állományt be kellene tölteni, de a kivonat tábla tartalmaz rá vonatkozó kivonatot és az megegyezik az aktuális állomány kivonatával, akkor az állomány az utolsó betöltés óta nem változott, nem kell újra betölteni.

Egy forrásállomány betöltése létrehoz vagy felülír egy forrás sémaobjektumot és érvényteleníti az összes olyan osztály sémaobjektumot, amely a forrásból származott. Ha az osztály sémaobjektum eddig nem létezett, létrehozza azt.

A parancssoros szintaxis a következő:

```
loadjava [options]
    {állománynév.java |
     állománynév.class |
     állománynév.jar |
     állománynév.zip |
     állománynév.sqlj |
     erőforrás_állománynév}...
    [-andresolve]
    [-debug]
    [-d|-definer]
```

```
[{-e|-encoding} séma]
[-fileout állománynév]
[-f|-force]
[{-g|-grant} felhasználó[, felhasználó]...]
[-help]
[-jarasresource]
[-noserverside]
[-nousage]
[-noverify]
[-o|-oci|-oci8]
[-r|-resolve]
[{-R|-resolver} "feloldó_specifikáció"]
[{-S|-schema} séma]
[-stdout]
[-s|-synonym]
[-tableschema séma]
[-t|-thin]
[-time]
[-unresolvedok]
{-user|-u} felhasználó/jelszó[@ab]
[-v|-verbose]
```

Az opciók sorrendje tetszőleges, mi abc sorrendben adtuk meg őket. Mint látjuk egyes opciók rövidíthetők.

Tetszőleges számú és sorrendű `.java`, `.class`, `.jar`, `.zip`, `.sqlj` és erőforrás-állomány neve adható meg a `loadjava` számára. Ezek feldolgozása a korábban leírtaknak megfelelően történik. A sémaobjektumok nevének képzése a felhasználó számára nem látható módon történik. A forrás- és osztályobjektumok esetén a sémaobjektum neve az állományban elsőnek definiált osztály teljes minősített neve lesz. Az erőforrás sémaobjektum neve viszont az *erőforrás-állománynév*ből képződik, általában megegyezik vele.

Az `-andresolve` opció azt írja elő, hogy az állomány fordítása és a hivatkozások feloldása egy menetben, a betöltés alatt történjen. Ekkor a függő osztályok nem érvénytelenítődnek. Az opció a `-resolve` helyett használatos és alkalmazása akkor javallott, ha egy korábban betöltött osztályt akarunk felülírni. Például egy osztály valamely metódusát akarjuk módosítani.

A `-debug` opció bekapcsolja az SQL naplózását, egyenértékű a `javac -g` megadásával.

Alapértelmezés szerint egy sémaobjektum futásánál a hívó privilégiumai az érvényesek. Ezt tudjuk megváltoztatni a `-definer` megadásával. Ekkor a létrehozó (a loadjava meghívását végző felhasználó) privilégiumai lépnek érvénybe.

Az `-encoding` megadása csak forrásállomány betöltésénél értelmes. Megfelel a `javac -encoding` megadásának. Felülírja a JAVA$OPTIONS táblában esetleg megadott előírásokat. Ha sem a JAVA$OPTIONS táblában, sem a parancssorban

nem adunk meg fordítási opciót, akkor az alapértelmezett érték `"System.getProperty("file.encoding");"` lesz.

A `-fileout` megadása esetén minden üzenet a megadott nevű állományba íródik ki.

A `-force` kikényszeríti az állomány betöltését akkor is, ha a kivonat tábla alapján ezt nem kellene megtenni.

A `-grant` opció az EXECUTE privilégiumot adományozza a felsorolt felhasználóknak. Ez szükséges ahhoz, hogy a felhasználó egy adott osztály metódusát meghívhassa. Szerepkörnek a privilégium nem adható.

A `-help` kiírja a `loadjava` felhasználására és az opciókra vonatkozó információkat.

A `-jarasresource` megadása azt eredményezi, hogy a teljes `.jar` állomány mint egyetlen erőforrás kerül be a sémába.

A `-noserverside` megadása esetén a szerver oldali `loadjava` egy JDBC-meghajtót fog használni az objektumok eléréséhez.

A `-nousage` letiltja a használatra vonatkozó üzenetek kiíratását `-help` esetén.

Az Oracle9*i* a bájtkód ellenőrzését a hivatkozások feloldásával egy időben végzi.

A `-noverify` opció ezt az ellenőrzést tiltja le. Csak a `-resolve` opcióval együtt értelmes.

Az `-oci` arra utasítja a `loadjava`-t, hogy az adatbázissal az OCI JDBC meghajtón keresztül kommunikáljon. Az `-oci` és a `-thin` kizárják egymást. Ha egyik sem szerepel, az `-oci` az alapértelmezett.

A `-resolve` opció fordítást és a külső hivatkozások feloldását írja elő az alapértelmezett feloldóval. Ha hiányzik, akkor az állományok betöltésre kerülnek, de nincs fordítás és feloldás.

A `-resolver` megad egy explicit feloldó specifikációt, melynek alakja:

"((*név_spec séma _spec*) [(*név_spec séma_spec*)]...)"

A *név_spec* hasonló a Java import utasításában megadott névhez. Lehet egy csomag minősített neve vagy egy Java-osztály teljes minősített neve. Az utolsó elem itt is lehet *. A minősítésben a . helyett / szerepel. A sé*ma_spec* egy séma neve vagy pedig a – karakter. A – karakter megadása esetén egy fel nem oldható külső hivatkozás nem teszi érvénytelenné az osztályt. Szerepeltetése esetén a fel nem oldott hivatkozások törlődnek. A feloldási művelet abban a sorrendben keresi végig a sémákat, ahogy azok fel vannak sorolva. Ha talál egy olyan sémaobjektumot, amelynek neve illeszkedik a névspecifikációhoz, a feloldást azzal végzi el. Például a

```
-resolver "((* HALLGATO)(* PUBLIC))"
```

esetén a keresés először a HALLGATO, majd a PUBLIC sémában történik. Ha a feloldás eredménytelen, akkor a hivatkozó osztály érvénytelen lesz és az Oracle9*i* hibaüzenetet küld (figyelmeztet egy hiányzó osztályra). A

```
-resolver "((* HALLGATO)(* PUBLIC)(hallgato/ptm/* -))"
```

esetén a keresés először a HALLGATO, majd a PUBLIC sémában történik. Ha nem sikerül a hivatkozás feloldása de a `hallgato.ptm` csomagban van hozzá osztály, akkor a hivatkozó osztályt érvényesnek tekinti. Ha viszont nincs a `hallgato.ptm` csomagban hozzá osztály, akkor a hivatkozó osztályt érvénytelennek tekinti és hiba keletkezik.

A `-schema` megadja azt a sémát, amelyben a sémaobjektum létrejön. Hiányában az alapértelmezett séma a bejelentkezési séma.

A `-stdout` átirányítja a kimenetet a `stderr`-ről a `stdout`-ra.

A `-synonym` létrehoz egy nyilvános szinonimát a betöltött osztály számára, lehetővé téve más sémából való elérését. Ehhez az opcióhoz a CREATE PUBLIC SYNONYM privilégium szükséges.

A `-tableschema` megadja azt a sémát, ahol a `loadjava` belső táblái elhelyezkednek. Hiányában ezek a táblák is a célsémában kerülnek elhelyezésre.

A `-thin` megadása esetén a `loadjava` a **thin** JDBC-meghajtót használja a kommunikációnál.

A `-time` minden üzenet mellé elhelyez egy időbélyeget.

Az `-unresolvedok`, a `-resolve` mellett megadva ignorálja a feloldási hibákat.

A `-user` megadja a bejelentkezési sémát a jelszóval együtt. A sémaobjektumok ebben jönnek létre. A *@ab* alakja az `-oci` és `-thin` esetén más-más. `-oci` esetén megadása opcionális. Hiánya esetén a `loadjava` a felhasználó alapértelmezett adatbázisát használja. Megadása esetén az *ab* egy TNS-név vagy egy Oracle Net Services név–érték lista lehet. `-thin` esetén megadása kötelező. Ilyenkor formája: *host:port:SID*, ahol a *host* az adatbázist futtató gép neve; a *port* azon port, amelyet az Oracle Net Services kapcsolódások követésére konfiguráltak (alapértelmezés szerint 5521); a *SID* az adatbázispéldány azonosítója (alapértelmezés szerint ORCL).

A `-verbose` megadása mellett a `loadjava` kiírja a futás közbeni részletes státusinformációkat. Ezekből lehet például megtudni, hogy a kivonat tábla információi alapján nem került betöltésre az állomány.

2.1.4. A `dropjava` eszköz

A `dropjava` a `loadjava` párja, a fordított irányú tevékenységek végrehajtására: törli a megadott Java-állományokhoz tartozó sémaobjektumokat. Mindig csak ezt használjuk egy `loadjava` által létrehozott sémaobjektum megszüntetésére.

A `dropjava` használható parancssorban és a DBMS_JAVA csomag eszközeként is.

Egy osztály törlése érvénytelenné teszi az összes tőle függő osztályt. Egy forrás törlése törli az összes belőle származó osztályt is.

Egy sémaobjektum csak pontosan ugyanúgy törölhető, ahogy létrehoztuk. Tehát, ha egy kliensen közvetlenül osztályokat és erőforrásokat töltöttünk be, akkor a `dropjava` csak ugyanazon osztály- és erőforrás-állományokkal futtatható.

A parancssoros szintaxis a következő:

```
dropjava [options]
    {állománynév.java |
     állománynév.class |
     állománynév.sqlj |
     állománynév.jar |
     állománynév.zip |
     erőforrás_állománynév}...
    {-u | -user} felhasználó/jelszó [@ab]
    [-jarasresource]
    [-noserverside]
    [-o | -oci | -oci8]
    [{-S | -schema} séma]
    [-stdout]
    [-s | -synonym]
    [-t | -thin]
    [-time]
    [-v | -verbose]
```

Az opciók és paraméterek jelentése ugyanaz mint a `loadjava` esetén.

Ha egy *állománynév* után nincs kiterjesztés, akkor azt a `dropjava` sémanévként értelmezi és minden olyan forrás, osztály és erőforrás sémaobjektum törlésre kerül, amelynek neve illeszkedik hozzá.

Ha a `dropjava` olyan állománynevet talál, amelyhez nem illeszkedik egyetlen sémaobjektum neve sem, üzenetet küld és folytatja a feldolgozást a következő állománynévvel.

2.1.5. Publikálás

Az Oracle9*i* csak úgy engedi meg adatbázisba betöltött Java-metódusok hívását kliensek és az SQL számára, hogy azokat előbb publikálni kell. Publikálható maga az objektum vagy az egyedi metódusok. A publikálás attól függ, hogy milyen típusú a Java-alkalmazás.

Egy Java tárolt alprogram esetén a metódusok egyedileg publikálhatók az osztályokban úgy, hogy hívási specifikációt adunk hozzájuk. Egy Java-alkalmazás sok osztályból és sok metódusból áll, mégis tipikusan csak néhány statikus metódust publikálunk, azaz látunk el hívási specifikációval. Részletesen lásd a 2.2.4. alfejezetben.

A szervletek és a JSP esetén a hozzájuk tartozó URL-t publikáljuk a JNDI névtéren belül.

CORBA és EJB esetén az objektumreferenciát publikáljuk.

2.1.6. Az operációs rendszer erőforrásainak kezelése

A Java-osztályok metódusai az operációs rendszer erőforrásait a JVM-en keresztül érik el. Alapértelmezés szerint egy Java-felhasználó nem férhet hozzá közvetlenül ezekhez az erőforrásokhoz. A rendszeradminisztrátor engedélyezheti a hozzáférést, módosítva a JVM biztonsági megszorításait. Az Oracle9*i* JVM biztonsági modellje megfelel a Java 2 biztonsági modelljének (l. 2.1.9. alfejezet).

Az operációs rendszer erőforrásai a következők: állományok, a memória, sockettek, szálak.

A Java tartalmaz olyan osztályokat, amelyek az állományokat reprezentálják. Ezek példányai állománykezelőként működnek. Az állományműveleteknél a relatív elérési út megadása mindig a $ORACLE_HOME-hoz képest relatív.

Az Oracle9*i* a memóriát saját hatáskörben kezeli, biztosítva egy újonnan létrehozott objektum elhelyezését és felszabadítva a memóriát, ha már nincs szükség az objektumra. A Java nem támogatja a közvetlen tárfoglalást és felszabadítást. Az automatikus tárkezelést egy szemétgyűjtögető valósítja meg.

A memória két részre oszlik: a Java-objektumok memóriájára és az operációs rendszer memóriájára. A Java-objektumok memóriájában vannak elhelyezve az objektumok és változók. Az operációs rendszer által használt memóriában helyezkednek el azok az erőforrások, amelyeket az operációs rendszer helyez el, ha egy objektum azt kéri. Ilyenek például az állományok, sockettek stb.

A szemétgyűjtögető csak a Java-objektumok memóriáját kezeli, az operációs rendszer memóriájában elhelyezett konstrukciók kezeléséről a programnak kell gondoskodnia. Például, amikor egy objektum megnyit egy állományt, az operációs rendszer egy állománykezelőt hoz létre és azt adja át az objektumnak. Ha az állományt nem zárjuk le, akkor az nyitva marad az adatbázishívás végéig, vagy amíg a JVM létezik. Minden operációs rendszerben véges számú állománykezelő lehet. Tehát ha nem zárjuk le a már nem használt állományokat, akkor lehet, hogy nem tudunk újat megnyitni.

A sockettek a kliens és a szerver közötti kapcsolat kezelésében játszanak szerepet. Egy alkalmazás nem állíthatja be magát a kapcsolatot, azt a JVM konfigurálása eldönti, hogy melyik hálózati protokoll kerül felhasználásra (Oracle Net's TTC vagy IIOP). A socket használható viszont például egy adott URL-hez való kapcsolódásra, ahonnan egy osztályt be akarunk tölteni az adatbázisba.

A kliens és a szerver oldalon egyaránt létrehozhatunk socket példányokat a `java.net.Socket()`, illetve a `java.net.ServerSocket()` konstruktorok segítségével.

A socket a kapcsolat végéig él.

Az Oracle9*i* JVM egy nem preemptív szálmodellt valósít meg. Ebben a JVM minden Java-szálat egy egyszerű operációs rendszer folyamatként futtat. Az ütemezés *round-robin* elvű, az egyik szálról a másikra az átkapcsolás akkor történik, mikor az blokkolt állapotba kerül (például meghívtunk egy socket metódust vagy a `yield()`

módszert) vagy véget ér. Ha egy szál soha nem kerül blokkolt állapotba, akkor mindaddig fut, amíg be nem fejeződik.

Az Oracle9*i* azért választja ezt a nagyon egyszerű modellt, mert a JVM be van ágyazva az adatbázisba és az a konkurenciának sokkal magasabb fokát biztosítja bármely hagyományos JVM-mel szemben.

Az Oracle9*i*-ben egy Java-szál véget ér, ha

- visszaadja a vezérlést a hívónak,
- nem kezelt kivétel váltódik ki,
- meghívásra kerül a `System.exit()` vagy az `oracle.aurora.vm.OracleRunTime.exitCall()` vagy az `oracle.aurora.mts.session.Session.THIS_SESSION().endSession()` metódus.

Ha egy szál további szálakat indít el, akkor a befejeződésre a következő szabályok vonatkoznak:

- A fő szál véget ér, ha visszaadja a vezérlést a hívónak vagy egy nem kezelt kivétel váltódik ki és minden más nem démon szál befejeződik.
- A fenti három metódus valamelyikét meghívja valamelyik szál.

Ha egy szál visszaadja a vezérlést vagy nem kezelt kivétellel ér véget, akkor az Oracle9*i* JVM minden démon szálban kiváltja a `ThreadDeathException` kivételt. A befejeztető metódusok meghívása nem váltja ki ezt a kivételt.

2.1.7. A Java memóriahasználata

Egy jól konfigurált adatbázis alkalmas tetszőleges Java-alkalmazás futtatására, azonban elképzelhető, hogy az erőforrások felhasználását az alkalmazáshoz kell igazítani. A következőkben áttekintjük, hogyan lehet konfigurálni a memóriát.

Az alábbi adatbázis inicializációs paraméterek módosításával tudjuk az alkalmazásunkhoz igazítani a memória használatát:

SHARED_POOL_SIZE

A megosztott memóriát az osztálybetöltő használja, átlagban 8 KB-ot lefoglalva egy-egy osztály számára. A használatára az osztályok betöltésénél és a feloldásánál van szükség, továbbá akkor, ha erőforrás objektum kezelésére kerül sor.

JAVA_POOL_SIZE

A Java-terület tartalmazza a Java-osztályok és metódusok megosztott módon használható reprezentációját, továbbá azokat a Java-objektumokat, amelyek a hívás végén a munkamenet területére kerülnek át. Alapértelmezett értéke 20 MB.

JAVA_SOFT_SESSIONSPACE_LIMIT

Ez a paraméter egy munkamenet Java-területének méretét szabályozza. Minden tárfoglalásnál ellenőrzésre kerül. Ha a méretet meghaladó tárfoglalás történik, akkor a nyomkövető állományba egy figyelmeztető üzenet kerül. Az alkalmazás menetét ez nem befolyásolja, de később elemezni lehet a működést.

JAVA_MAX_SESSIONSPACE_SIZE

A Java-terület maximális mérete (alapértelmezett értéke 4 GB). Ha a tárfoglalás meghaladná ezt, akkor az alkalmazás félbeszakad OUT OF MEMORY hibával.

A JAVA_POOL_SIZE és a SHARED_POOL_SIZE értékét a Database Configuration Assistant Memory menüpontjában állíthatjuk be.

Ha a Java-terület egy dedikált szerveren van, akkor a Java-osztályok megosztott része, amely csak olvasható memóriát jelent (idetartoznak például a metódusok) összességében átlagban 4–8 KB területet igényel osztályonként. Az egyéb információk tárolása a Program Globális Területen történik. Dedikált szerver esetén, ha az alkalmazás csak tárolt Java-alprogramokat tartalmaz, a teljes Java-terület nem nagyobb 10 MB-nál.

Megosztott szerver esetén az egyéb információk a Rendszer Globális Területen jelennek meg. CORBA és EJB alkalmazások esetén a szükséges Java-terület akár 1 GB is lehet.

Az aktuálisan használt Java-területre és megosztott területekre vonatkozó információkat a V$SGASTAT tábla lekérdezésével érhetjük el.

Ha a fordítás közben az

```
A SQL exception occurred while compiling: ORA-04031: unable to allocate
bytes of shared memory...
```

hibaüzenetet kapjuk, akkor kevés a memória. Ekkor le kell állítani az adatbázist, meg kell növelni a JAVA_POOL_SIZE értékét és újra kell indítani a szervert. A hibaüzenetben szereplő *shared memory* megtévesztő, nem a megosztott terület fogy el a fordításnál, hanem a Java-terület.

Ha egy osztály betöltésénél fogy el a memória, akkor ez nem fog kiderülni közvetlenül, viszont az osztály érvénytelen lesz. Később, ha hivatkozunk erre az osztályra, akkor a `ClassNotFoundException` vagy a `NoClassDefFoundException` kivétel váltódik ki futási időben. Ugyanezek a kivételek jelzik azt a szituációt is, amikor megsérült osztályállományt akarunk betölteni. Ekkor a következőket tehetjük:

- Ellenőrizzük, hogy az adott osztályt egyáltalán elhelyeztük-e a betöltendő osztályok között.
- Használjuk a `loadjava -force` opcióját, hogy a már az adatbázisban levő osztályt mindenképpen felülírja az újjal.
- Használjuk a `loadjava -resolve` opcióját, hogy a feloldás rögtön a betöltéskor (és ne futási időben) megtörténjen.

- Ha a `loadjava` memóriaproblémát jelez, növeljük meg a SHARED_POOL_SIZE és a JAVA_POOL_SIZE értékét és próbáljuk újra.
- Ellenőrizzük le a USER_OBJECTS táblában az új osztály STATUS mezőjét, ebben VALID-nak kell szerepelnie.

Az Oracle9*i* megőrzi egy Java-program állapotát két hívás között. Ezt úgy valósítja meg, hogy a statikus változók tartalmát a hívás végén elmenti a munkamenet területére. A munkamenet terület a kliens munkamenetének teljes időtartama alatt létezik és a statikus változókat és azon objektumokat tárolja, amelyek élnek két hívás között is. Minden hívás végén a migráció automatikusan bekövetkezik.

Az egy adott pillanatban egy adott Java-programot futtató felhasználók számának maximalizálása érdekében az egy munkamenet által lefoglalt tárterületet minimalizálni kell. Az inaktív munkamenet területét olyan kicsivé kell tenni, amilyenné csak lehet. Ennek nagyon egyszerű módja az, hogy a hívás végén a nagy méretű adatszerkezeteket megszüntetjük és ha a későbbiekben ismét szükség lenne rájuk, akkor újra létrehozzuk őket. Ilyen adatszerkezetek tipikusan például a pufferek, gyorsítótárak, statikus tömbök.

Az alapprobléma persze a szokásos, a tárfoglalás és a futási idő egymás ellen dolgozik. A megszüntetett adatszerkezetek létrehozása időbe kerül, a megőrzöttek azonnal használhatók, de foglalják a tárat.

Az Oracle9*i* egy olyan mechanizmust bocsát rendelkezésünkre, amely lehetővé teszi statikus változók törlését a hívás végén és azok újralétrehozását, amikor egy következő híváskor szükségesek. Az ilyen változókat regisztrálni kell. Erre szolgál az `oracle.aurora.memoryManager.EndOfCallRegistry` osztály.

A Java nyelv szemantikája szerint a statikus változók léteznek két hívás között is és megtartják állapotukat. Az Oracle9*i* JVM minden hívás végén a statikus változókat a hívási memóriából átviszi a munkamenet memóriába.

Az Oracle9*i* rendelkezésre bocsát egy `MemStat` nevű segédprogramot, amely a statikus változók által lefoglalt memória nyomonkövetésére alkalmas. Ennek segítségével behatárolhatók és eltávolíthatók azok a statikus adatok, amelyeket nem szükséges megőrizni és azáltal növelni tudjuk a Java-programunk hatékonyságát.

A meghívás módjától függően a `MemStat` vagy egyetlen osztályt vagy az aktuális munkamenet összes osztályát elemzi. Minden osztályban a statikus változókat veszi sorra (ezeket *gyökérnek* hívja) és azok szerkezetétől függően a következőket adja meg:

- Ha a változó egy primitív objektum típusú (például Integer), akkor megadja az osztályát és a méretét.
- Ha a változó nem primitív objektum típusú, akkor rekurzívan felsorolja az összes hivatkozott objektumot.
- Ha a változó tömb típusú, akkor felsorolja az elemeit.

A fenti folyamat rekurzívan addig folytatódik, amíg az összes, a statikus változók által elérhető objektumot nem érinti. Ezáltal tetszőleges objektumgráfok jelenhetnek meg, amelyek tartalmazhatnak ciklusokat is.

A MemStat az elemzés eredményét HTML-állományba írja, amely egy általunk megadott könyvtárban keletkezik. Az elemzés végén bármely webböngészővel megtekinthetjük az állományt.

Az elemzést a program bármely pontján kérhetjük.

A MemStat maga az oracle.aurora.memstat.Memstat osztály három statikus metódusaként van implementálva.

Ha egyetlen osztály statikus változóit akarjuk elemezni, akkor a

```
writeDump(Class MyClass, String outputPath, String filePrefix)
```

metódust, ha a munkamenet minden betöltött osztályának statikus változóira vagyunk kíváncsiak, akkor a

```
writeDump(String outputPath, String filePrefix)
```

metódust, és ha a munkamenet összes osztályának statikus változóira a hívás végén vagyunk kíváncsiak, akkor a

```
writeDumpAtEOC(String outputPath, String filePrefix)
```

metódust alkalmazzuk.

Az outputPath paraméter azt a könyvtárat adja meg, ahol a MemStat az eredményt elhelyezi. A sztring alakja platformfüggő. Például Windows alatt C:\\base\\memstat alakú lehet.

A filePrefix adja a generált HTML-állományok alapnevét. Mivel a MemStat eredménye igen nagy is lehet, és egyes webböngészők korlátozzák az állományok méretét, a MemStat bizonyos méreten felül több output állományt hoz létre. Ezek neve az alap állománynév után írt sorszámokban különbözik. Tehát ha az elemzes alapnevet adtuk meg, akkor a generált állománynevek elemzes.htm, elemzes1.htm, elemzes2.htm, ... lesznek.

Ha egy híváson belül a MemStat-ot többször meghívjuk ugyanazon paraméterekkel, akkor a generált output állományok felülírják a korábbiakat.

A MyClass paraméter az elemzendő egyetlen osztály nevét határozza meg.

A MemStat az Oracle szerverfolyamaton belül fut, ezért a következő jogosultságokat biztosítani kell annak a felhasználónak vagy szervernek, aki meghívta:

- Az objektumgráf objektumainak privát változóit el kell tudni érni. Ha ez a jogosultság nincs meg, akkor a MemStat működik, de nem lesz feljegyzés azokról az objektumokról, amelyek privát változókra hivatkoznak.
- El kell tudni érni a Java futtatórendszerét ahhoz, hogy meghatározhatók legyenek az aktuális munkamenetbe betöltött osztályok. Ha ez a jogosultság hiányzik, a MemStat nulla darab betöltött osztályt érzékel és az output üres lesz.
- El kell tudni érni a szerver állományrendszerét, ahol az output elhelyezésre kerül. Ezen jogosultság hiányában a MemStat az output generálásának kezdetén egy kivételt vált ki.

A következő SQL-utasítások garantálják ezeket a jogokat (részletesen lásd a 2.1.9. alfejezetben) a KISS felhasználónak:

```
CALL DBMS_JAVA.GRANT_PERMISSION ('KISS',
'SYS:java.lang.reflect.ReflectPermission',
'suppressAccessChecks',null);
CALL DBMS_JAVA.GRANT_PERMISSION ('KISS',
'SYS:oracle.aurorasecurity.JServerPermission`,
'JRIExtensions',null);
CALL DBMS_JAVA.GRANT_PERMISSION ('KISS',
'SYS:java.io.FilePermission', 'C:\\base\\memstat',
'read,write');                    // Windows
```

Az utolsó példa alapján itt jegyezzük meg, hogy a Windows-platformon a `MemStat` használatánál a Java-sztringen belül a visszaperjeleket kettőzni kell, SQL-ben viszont nem.

2.1.8. Fordítás natív kódra

A Java nyelv tervezési szempontjai között első helyen állt a platformfüggetlenség és a biztonságos kódolás szempontja. Ezek érdekében viszont a futtatható kód hatékonysága szenvedett csorbát.

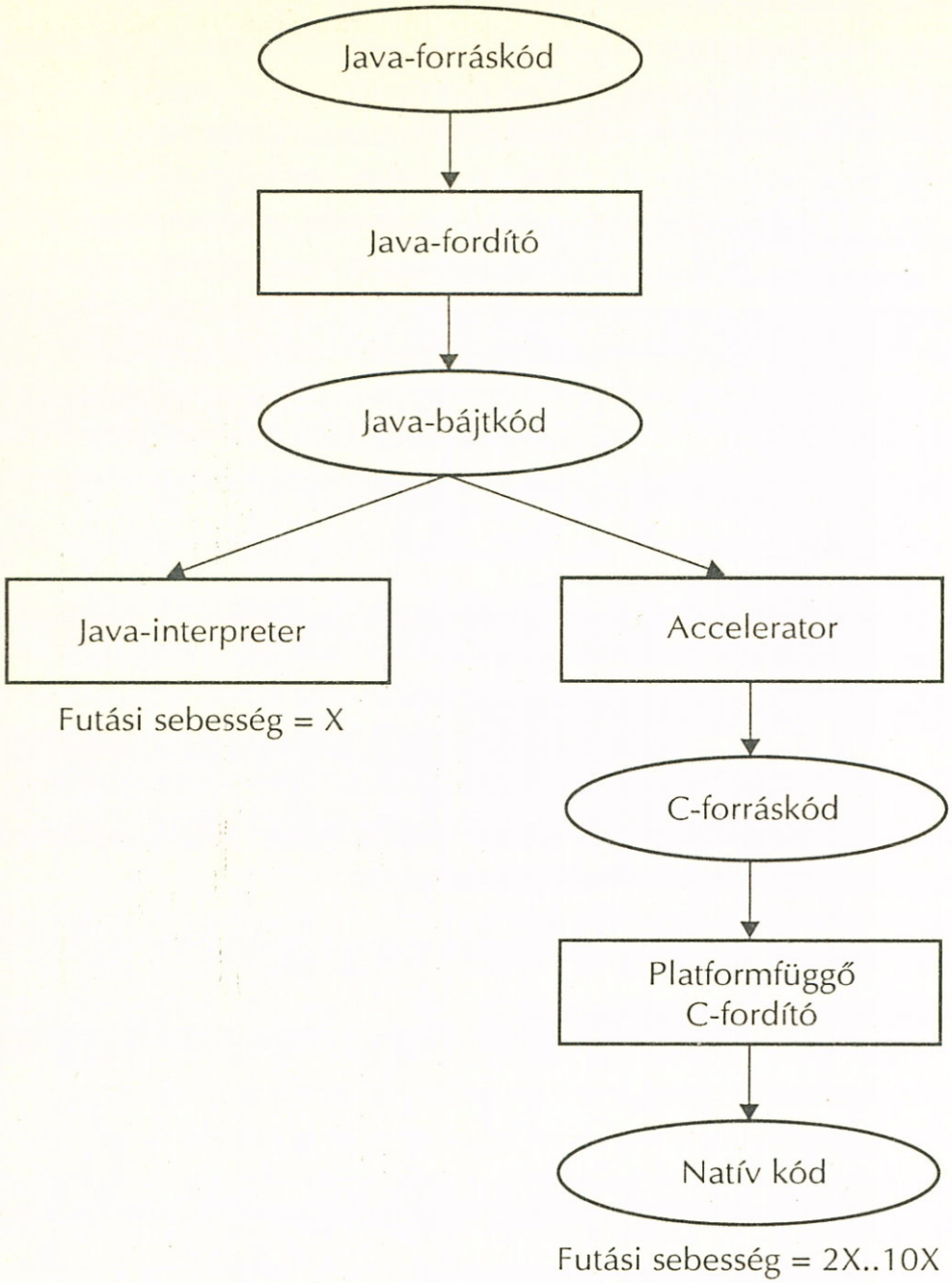
A natív kódra való fordítás a futási időt általában közel egy nagyságrenddel csökkenti, de legrosszabb esetben is felezi.

A natív kódra való fordításnak kétféle megvalósítása van a JVM-implementációkban. A Just-In-Time (JIT) fordítók (dinamikus fordítók) a Java-bájtkódot híváskor fordítják le natív gépi kódra. Az Ahead-Of-Time (AOT) fordítók (statikus fordítók) a fordítást a futtatás előtt végzik.

Az Oracle9*i* a statikus fordítást alkalmazza, ennek eszköze a Accelerator. Az Accelerator a platformfüggetlen bájtkódot platformfüggetlen C-kódra fordítja le, emelyet aztán egy C-fordító platformfüggő gépi kóddá alakít át, amely aztán megosztott könyvtárakban vagy DLL-ekben kerül elhelyezésre. Ezt szemlélteti a 2.2. ábra. Ezután az Oracle9*i* biztosítja a kódok szétosztását a felhasználók, folyamatok, munkamenetek között.

Ha adva van egy JAR-állomány, akkor az Accelerator a következőket hajtja végre:

1. Ellenőrzi az adatbázisba töltendő osztályokat.
2. Visszakeresi az adatbázisból az osztályok bájtkódját és egy saját könyvtárba tölti azokat.
3. A bájtkódot átalakítja C-kóddá.
4. Lefordítja és szerkeszti a C-kódot egy natív C-fordító segítségével.
5. Eredményül a `$ORACLE_HOME/javavm/admin` könyvtárban létrejön a megosztott könyvtár.



2.2. ábra. *A Java-interpreter és az Accelerator*

A megosztott könyvtárak csak az Oracle9*i* azon kiadásában használhatók, melyekben létrehoztuk őket. Tehát ha át akarjuk vinni az alkalmazást egy másik verzió alá, akkor újra kell fordítani azt.

Minden megosztott könyvtár egy Java-csomaghoz tartozik. A csomagrendszer és az elnevezések platformfüggők. Például Windows NT alatt a `java.lang` osztályai az `orajox8java_lang.dll`-ben találhatók.

A natív kódokat tartalmazó Java-állományokat a felhasználó számára nem látható módon kezeli az Oracle9*i*.

Az Accelerator az `ncomp` eszközben van megvalósítva. Az `ncomp` natív módon lefordít minden listában, JAR- vagy ZIP-állományban megadott osztályt és elhelyezi őket egy megosztott könyvtárban. Ezeket az osztályokat először be kell tölteni az adatbázisba.

Az ncomp szintaxisa a következő:

```
ncomp [options] állomány
    {-user | -u} felhasználó/jelszó[@ab_url]
    [-load]
    [{-d | -projectDir} könyvtár]
    [-force]
    [-lightweightDeployment]
    [-noDeploy]
    [{-o | -output JarFile} jar_állomány]
    [-thin]
    [-oci | -oci8]
    [-update]
    [-verbose]
```

Az *állomány* lehet *név*.jar, *név*.zip vagy *név*.classes alakú, ahol a *név*.jar egy olyan JAR-állomány, a *név*.zip pedig egy olyan ZIP-állomány, amely a natív módon fordítandó osztályokat tartalmazza. A *név* egy teljes állománynév, amely az útvonalat is tartalmazza. Ha abban a könyvtárban akarjuk lefuttatni az ncomp-ot, amely a JAR- vagy a ZIP- vagy a CLASSES-állományt tartalmazza és nem adtuk meg a -projectDir opciót, akkor csak az állomány nevét kell megadni.

A *név*.classes azon állomány teljes neve, amely a natív módon fordítandó osztályok listáját tartalmazza.

A -user megadja a felhasználó nevét, jelszavát és az adatbázis kapcsolósztringjét. Az állományok az így meghatározott adatbázispéldányba töltődnek be. Az URL megadása alapértelmezésben az OCI szintaxisát követi.

A -load opció lefuttatja a loadjava-t. Nem használható *név*.classes esetén.

A -projectDir megadja a projekt könyvtár teljes nevét. Ha hiányzik, az Accelerator azt a könyvtárat tekinti projekt könyvtárnak, amelyből az ncomp-ot meghívtuk. Ha a megadott könyvtár nem létezik, akkor az aktuálisat fogja használni.

A -force az összes osztály (a már lefordítottakat is beleértve) fordítását írja elő.

A -lighweightDeployment azt jelzi, hogy a megosztott könyvtárak és a natív fordítás információi elkülönítetten jelennek meg. Az Accelerator alapértelmezésben a fordítási információkat és a lefordított megosztott könyvtárakat egyetlen JAR-állományba helyezi, a megosztott könyvtárakat felmásolja a szerverre, a $ORACLE_HOME/javavm/admin könyvtárba és a fordítási információkat is elhelyezi a szerveren. Ha magunk akarjuk a szerverre felvinni a megosztott könyvtárakat, akkor használjuk a -lightweightDeployment opciót. Ekkor két lépésben történhet meg a telepítés:

1. A -lightweightDeployment opció mellett használjuk a -noDeploy opciót is. Ekkor létrejön egy telepítő JAR-állomány, amely csak az ncomp információit tartalmazza, a megosztott könyvtárak nem kerülnek bele.

2. Másoljuk át a natív fordítási projekt könyvtár `lib` alkönyvtárából a megosztott könyvtárakat a `$ORACLE_HOME/javavm/admin` könyvtárba. Ehhez a `FilePermission` jogosultság szükséges. A DBA és JAVASYSPRIV szerepkörök rendelkeznek ezzel a jogosultsággal. Ezután használjuk a `deploync` eszközt a telepítő JAR-állomány szerverre telepítésében.

Az `-outputJarFile` megadja azon JAR-állomány nevét és könyvtárát, ahová a natív módon lefordított állományok kerülnek. Ha nem adjuk meg, akkor az `ncomp` az *állomány*`_depl.jar` nevet fogja adni neki. Ha nem adunk meg könyvtárat, akkor az output JAR-állomány a projekt könyvtárba kerül.

A `-thin` és `-oci` opciók jelentését lásd a `loadjava` leírásánál.

Az `-update` opció esetén, ha az *állomány* már lefordított osztályokat is tartalmaz, akkor az Accelerator a telepítő JAR-állományhoz hozzáfűzi az újonnan lefordított osztályokat.

A `-verbose` esetén az output részletes fordítási információkat is tartalmaz.

A natív fordítás alatti hibák a képernyőn jelennek meg. A megosztott könyvtárak szerverre telepítése, vagy a futási időben bekövetkező hibák viszont a `statusnc` eszközzel tekinthetők meg. Ez az eszköz a JACCELERATOR$DLL_ERRORS táblát hivatkozza. A natív fordítás közben bekövetkező hiba esetén az aktuális csomag feldolgozása félbeszakad és a fordítás a következő csomaggal folytatódik.

Tipikus hibák az alábbiak:

- A Java-osztály nem létezik az adatbázisban. Ha az osztály nincs betöltve, akkor az Accelerator nem illeszti azt be a megosztott könyvtárba.
- A Java-osztály érvénytelen, vagyis valamely eszköz, amelyre hivatkozik, nem található vagy maga is érvénytelen.

Ha a hibát felderítettük, két lehetőség közül választhatunk:

- újrafordítjuk a megosztott könyvtárat,
- betöltjük a Java-osztályt az adatbázisba.

Ha csak a betöltést választjuk újrafordítás nélkül, akkor a betöltött osztály és minden általa hivatkozott osztály végrehajtása interpretált módon fog történni. Az ilyen osztályok esetén a `statusnc` a NEED_NCOMPING üzenetet adja.

A továbbiakban áttekintjük azokat a lehetőségeket, kombinációkat, ahogyan az `ncomp` az opciók különböző módon történő megválasztásával használható.

Már betöltött osztályok natív fordítása

A már betöltött (korábban letesztelt) osztályok natív fordításához meg kell adni a JAR- vagy ZIP-állományt vagy az osztályok listáját tartalmazó CLASSES-állományt. Az Accelerator sorra veszi az összes osztályt, lefordítja, majd elhelyezi őket a megosztott könyvtárakba – minden könyvtár az osztályok egy-egy csomagját tartalmazza.

Például, ha a `berszamfejtes` JAR-állomány tartalmazza a fordítandó már betöltött osztályokat, akkor a hívás alakja:

```
ncomp -user KISS/K000 berszamfejtes.jar
```

Ha a natív fordítás után valamely osztályok módosultak és újrafordítást kérünk, akkor az Accelerator csak azokat a csomagokat fordítja újra, amelyek tartalmazzák a módosult osztályokat.

Még be nem töltött osztályok natív fordítása

Ilyen szituáció áll elő például, ha a fejlesztés és tesztelés más gépen történik, mint ahol a natív fordítást végre akarjuk hajtani. Ekkor át kell hozni az osztályokat, be kell tölteni őket, majd ezután tudunk fordítani. A betöltés a `loadjava` segítségével, a `-load` opció megadásával történik. A parancs:

```
ncomp -user KISS/K000@abtavoli:5521:orc1 -thin
      -load berszamfejtes.jar
```

Teljes újrafordítás és telepítő állomány generálása

A következő parancs:

```
ncomp -user KISS/K000 -force
      -outputJarFile telepito.jar berszamfejtes.jar
```

parancs a `berszamfejtes.jar` állomány minden osztályát lefordítja (függetlenül attól, hogy azok korábban le voltak-e már fordítva) és az eredményt egy telepítő állományba (`telepito.jar`) helyezi, amely bármely adatbázisba telepíthető. A telepítő állomány tartalmazza a megosztott könyvtárakat és azok installációs osztályait, de nem tartalmazza az eredeti Java-osztályokat. A telepítő állományt a `deploync` segítségével telepíthetjük, előtte azonban a `loadjava` segítségével az összes Java-állományt a cél adatbázisba kell tölteni.

Ha az `ncomp` segítségével előállítottunk egy telepítő állományt, akkor annak telepítését a `deploync` eszközzel végezhetjük. A natív fordítás és a telepítés operációs rendszerének és adatbázis-verziójának egyeznie kell. A már telepített megosztott könyvtárak listáját a JACCELERATOR$DLLS tábla tartlamazza.

A `deploync` szintaxisa a következő:

```
deploync [options] telepítő_áll_név
        {-user | -u} felhasználó/jelszó[@ab url]
        [{-projectDir | -d} könyvtár]
        [-thin]
        [-oci | -oci8]
```

A *telepítő_áll_név* adja meg a telepítő JAR-állomány teljes nevét. Az állomány tartalmát a `deploync` nem ellenőrzi.

A többi opció jelentése megegyezik az `ncomp` opcióiéval.

A natív módon lefordított Java-osztályok állapotát a `statusnc` eszközzel tudjuk ellenőrizni. Az osztályok státusát a képernyőre kérhetjük, vagy egy megadott állományba írhatjuk. Ezen túlmenően a `statusnc` az outputját mindig elhelyezi a JACCELERATOR$STATUS táblában. Ez a tábla mindig csak a `statusnc` utolsó futásának eredményét tartalmazza. Az osztályok státusa a következő lehet:

ALREADY_NCOMPED:	Az osztály már natív módon lefordításra került.
NEED_NCOMPING:	Az osztály újratöltésre került a natív fordítás után. Javallott az újrafordítása.
INVALID:	Az osztály érvénytelen, és így eltávolításra került a megosztott könyvtárból.

A `statusnc` szintaxisa:

```
statusnc [options] állomány
        {-user | -u} felhasználó/jelszó[@ab_url]
        [{-output | -o} output_állomány]
        [{-projectDir | -d} könyvtár]
        [-thin]
        [-oci | -oci8]
```

Az *állomány* azon ZIP-, JAR- vagy CLASSES-állomány, amely osztályainak státusára kíváncsiak vagyunk.

Az `-output` opcióban megadhatjuk azon állomány nevét, ahová az output kerül. Hiányában a státus a képernyőre íródik.

A többi opció jelentése megegyezik az `ncomp` opcióiéval.

2.1.9. Az Oracle9*i* Java-alkalmazások biztonsági kérdései

A Java-alkalmazások biztonsági kérdéseinek két szintjét különböztetjük meg. Az egyik a hálózati hozzáférés, az adatbázishoz való csatlakozás szintje. Ez függ attól, hogy hogyan kapcsolódunk az adatbázishoz: az Oracle Net, JDBC vagy esetleg CORBA vagy EJB segítségével. Ezen szint biztonsági problémáinak tárgyalása meghaladja jelen könyv kereteit.

Ha viszont már kapcsolódtunk az adatbázishoz, akkor egyrészt a Java-jogosultságokkal másrészt adatbázis-privilégiumokkal kell rendelkeznünk ahhoz, hogy az adatbázisban tárolt erőforrásokat elérhessük. Ezek a következők:

- az adatbázis erőforrásai (például táblák, PL/SQL-csomagok),
- az operációs rendszer erőforrásai (például állományok, sockettek),
- az Oracle9*i* JVM osztályai,
- a felhasználók által betöltött osztályok.

A jogosultságok egy része a megszokott adatbázis-privilégiumok körébe tartozik, ezek a szokásos módon biztosíthatók felhasználók vagy szerepkörök számára. A felhasználók által létrehozott osztályok elérésének biztosítására a `loadjava -grant` opciója szolgál. A jogosultságok másik része a Java-jogosultságokat foglalja magába. Ebben a fejezetben részletesebben ezekről szólunk.

Az Oracle9*i* JVM a Java 2 biztonsági modelljét implementálja `Permission` objektumok segítségével. A Java 2 biztonsági eszközök automatikusan épülnek fel és biztosítják az operációs rendszer erőforrásainak és a JVM osztályainak védelmét. A JAVA_ADMIN felhasználó az, aki ezeket a jogokat megadhatja további felhasználók számára.

Az Oracle9*i* is biztonsági osztályokkal, `Permission` objektumokkal oldja meg a jogosultságok kezelését. Ezek kezelését a DBMS_JAVA alprogramjai biztosítják. Alaphelyzetben létezik a biztonságpolitikai tábla (`PolicyTable`) és ezen tábla kezeli a `Permission` objektumokat. Normális esetben ez a tábla szerkeszthető.

A táblán két nézet létezik: USER_JAVA_POLICY és DBA_JAVA_POLICY. Az első csak az aktuális felhasználó számára releváns sorokat, az utóbbi a biztonságpolitikai tábla minden sorát tartalmazza. A biztonságpolitikai tábla és a nézetek oszlopai a következők:

`Kind:`	Értéke GRANT vagy RESTRICT. Azt mutatja meg, hogy az adott jogosultság teljes (GRANT) vagy korlátozott (RESTRICT).
`Grantee:`	A feljogosított felhasználó, szerepkör vagy séma neve.
`Permission_scheme:`	Az a séma, amelybe a `Permission` objektum betöltésre kerül.

`Permission_type:`	A `Permission` osztály teljes neve sztring formában. A lehetséges `Permission` típusok az alábbiak:
	• `java.util.PropertyPermission`
	• `java.io.SerializablePermission`
	• `java.io.FilePermission`
	• `java.net.NetPermission`
	• `java.net.SocketPermission`
	• `java.lang.RuntimePermission`
	• `java.lang.reflect.ReflectPermission`
	• `java.security.SecurityPermission`
	`oracle.aurora.rdbms.security.PolicyTablePermission`
	`oracle.aurora.security.JServerPermission`
	Az utolsó kettő kivételével minden `Permission` típus Java 2 típus. A `PolicyTablePermission` a biztonságpolitikai tábla módosítását engedélyezi. A `JServerPermission` pedig az Oracle9*i* JVM-erőforrásaihoz való hozzáférést garantálja.

`Permission_name:`	A `Permission` objektum célattribútuma.
`Permission_action:`	A `Permission` objektum akcióattribútuma.

`Status:`	Értéke ACTIVE vagy INACTIVE lehet. A `Permission` objektum engedélyezett (ACTIVE) vagy letiltott (INACTIVE) állapotát jelöli.
`Key:`	A táblabeli sor sorszáma mint kulcs. Ezt kell használni a `Permission` objektum engedélyezésénél, letiltásánál vagy törlésénél.

A jogosultságok beállítására kétféle lehetőség kínálkozik:

- A *finomszemcsés* beállítás esetén minden jogosultságot egyenként adományozunk a felhasználónak vagy a szerepkörnek.
- Az *általános jogosultság hozzárendelése szerepkörhöz* mód esetén csak szerepkörnek adunk jogosultságokat.

A finomszemcsés beállítás esetén a biztonságpolitikai tábla alábbi mezőit kell megadnunk: `Grantee`, `Permission_name`, `Permission_action`.

A jogosultság megadható SQL vagy Java segítségével. Mindkét esetben a biztonságpolitikai tábla azon sorának sorszámát (mint kulcsot) kapjuk vissza, amelybe az új bejegyzés bekerült. Ha a feljogosítás már a táblában létező sorra vonatkozik, az nem módosul. Ha a sor le volt tiltva, a feljogosítás engedélyezi azt.

A következőben megadjuk azokat az eszközöket, amelyekkel a jogosultságok kezelhetők.

Feljogosítás a DBMS_JAVA csomaggal

```
procedure grant_permission( grantee varchar2,
                            permission_type varchar2,
                            permission_name varchar2,
                            permission_action varchar2 )

procedure grant_permission( grantee varchar2,
                            permission_type varchar2,
                            permission_name varchar2,
                            permission_action varchar2,
                            key OUT number)
```

Feljogosítás Javával

```
long oracle.aurora.rdbms.security.PolicyTableManager.grant(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name,
     java.lang.String permission_action);

void oracle.aurora.rdbms.security.PolicyTableManager.grant(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name,
     java.lang.String permission_action,
     long[] key);
```

Jogok korlátozása a DBMS_JAVA csomaggal

```
procedure restrict_permission( grantee varchar2,
                               permission_type varchar2,
                               permission_name varchar2,
                               permission_action varchar2)

procedure restrict_permission( grantee varchar2,
                               permission_type varchar2,
                               permission_name varchar2,
                               permission_action varchar2,
                               key OUT number)
```

Jogok korlátozása Javával

```
long oracle.aurora.rdbms.security.PolicyTableManager.restrict(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name,
     java.lang.String permission_action);

void oracle.aurora.rdbms.security.PolicyTableManager.restrict(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name,
     java.lang.String permission_action,
     long[] key);
```

A következő kódrészlet a DBMS_JAVA csomag `grant_permission` eljárásának segítségével módosítja a biztonságpolitikai táblát. A KISS felhasználó rendelkezik a `PolicyTable` módosítási jogával és egy csomagon belül írási és olvasási jogot ad a NAGY felhasználónak a `Teszt1` állományra:

```
connect KISS/K000
call dbms_java.grant_permission('NAGY',
'java.io.FilePermission', '/teszt/Teszt1',
'read,write');
commit;
```

A jogosultságok korlátozhatók. Például egy adott könyvtárra egy felhasználó kaphat írási és olvasási jogot, de bizonyos állományok hozzáférését letilthatjuk. Ekkor úgy járhatunk el, hogy megadjuk a teljes jogot, majd korlátozzuk azt. Tehát a kivételeket soroljuk föl az általánoson belül. A következő példában a `temp` könyvtár minden állományát minden felhasználó írhatja és olvashatja, kivéve a `kodok` állományt, amelyhez ilyen joga csak a KISS felhasználónak lehet (az állomány az ő saját állománya).

```
connect KISS/K000
call dbms_java.grant_permission('PUBLIC',
     'java.io.FilePermission', '/temp/*', ',write');
```

```
call dbms_java.restrict_permission('PUBLIC',
     'java.io.FilePermission', '/temp/kodok', 'read,write');
call dbms_java.grant_permission('KISS', 'java.io.FilePermission',
     '/temp/kodok', 'read,write');
commit;
```

Az Oracle9*i* JVM első inicializálása után csak a JAVA_ADMIN szerepkör rendelkezik a `PolicyTable` tábla `PolicyTablePermission` módosítási jogával. A JAVA_ADMIN szerepkör hozzátartozik a DBA szerepkörhöz. Az adminisztrátor tudja tehát megadni és korlátozni a `PolicyTablePermission` jogokat más felhasználók esetén. Ehhez a DBMS_JAVA csomagban a következő eszközök állnak rendelkezésére:

```
procedure grant_policy_permission( grantee varchar2,
          permission_schema varchar2,
          permission_type varchar2,
          permission_name varchar2)

procedure grant_policy_permission( grantee varchar2,
          permission_schema varchar2,
          permission_type varchar2,
          permission_name varchar2,
          key OUT number)
```

Javából viszont az alábbi módszereket érhetjük el:

```
long oracle.aurora.rdbms.security.PolicyTableManager.
grantPolicyPermission(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name);

void oracle.aurora.rdbms.security.PolicyTableManager.
grantPolicyPermission(
     java.lang.String grantee,
     java.lang.String permission_type,
     java.lang.String permission_name,
     long[] key);
```

A következő példában a JAVA_ADMIN (mint SYS) a KISS felhasználót feljogosítja a biztonságpolitikai tábla módosítására. Ettől kezdve KISS továbbadhatja más felhasználóknak az írási, olvasási és törlési jogokat.

```
call dbms_java.grant_policy_permission('KISS', 'SYS',
'java.io.FilePermission', '*');
```

Saját `Permission` típus használatához a következő lépéseket kell végrehajtani:

1. Létre kell hozni és be kell tölteni a felhasználói `Permission` típust.
2. A saját `Permission` típussal fel kell jogosítani más felhasználókat.
3. A saját `Permission` típus biztonsági ellenőrzéseit implementálni kell.

Lássuk ezeket a lépéseket részleteiben, példákon keresztül.

1. *Felhasználói* `Permission` *típus létrehozása és betöltése.* Saját `Permission` típus a Java 2 Permission osztályának kiterjesztésével hozható létre. A következő kód ezt szemlélteti:

```
package proba.kiss;
import java.security.Permission;
import java.security.BasicPermission;
public class SajatPermission extends BasicPermission {
public SajatPermission(String name) {super(name);
}
public boolean implies(Permission p) {
boolean result = super.implies(p);
return result;
}
}
```

2. *Feljogosítás saját* `Permission` *típussal.* A `Permission` típust létrehozó felhasználó implicit módon rendelkezik az adott jogosultság adminisztrátori jogaival, tehát futtathatja a `grant_policy_permission` metódust és módosíthatja a biztonságpolitikai tábla megfelelő sorait. A következő kód ezt szemlélteti:

```
REM Mivel KISS a SajatPermission tulajdonosa, így
REM sajátmagát felruházhatja adminisztrátori
REM jogokkal

connect KISS/K000
call dbms_java.grant_policy_permission('KISS', 'KISS',
     'proba.kiss.SajatPermission', '*');
commit;
```

Ha egy felhasználó már rendelkezik adminisztrátori jogokkal, akkor továbbadhatja az akciós jogokat is. A következő kód alapján KISS mindent, NAGY csak a `jav` kezdetű akciókat hajthatja végre:

```
connect KISS/K000
call dbms_java.grant_permission('KISS',
'KISS:proba.kiss.SajatPermission', '*', null);
call dbms_java.grant_permission
('NAGY', 'KISS:proba.kiss.SajatPermission', 'jav.*', null);
commit;
```

3. *A biztonsági ellenőrzések implementálása.* Miután létrehoztuk, betöltöttük és kiosztottuk a saját jogosultságokat, implementálni kell a biztonsági ellenőrzést. A következő példában az alábbi három metódust látjuk: `bizalmas`, `jav`, `print`. A 2. lépés kódjának végrehajtása után a következő felhasználók a következő metódusokat futtathatják:

- KISS mindegyiket,
- NAGY csak a `jav` metódust,
- bárki más a `print` metódust.

```
package proba.kiss;
import java.security.AccessController;
import java.security.Permission;
import java.security.PrivilegedAction;
import java.sql.Connection;
import java.sql.SQLException;

public class Kiss {
private static String bizt = "Kiss titkai";
SajatPermission bizalmasPermission =
      new SajatPermission("bizalmas");

/**
* Ez egy védett művelet, csak KISS rendelkezik
* a végrehajtásához szükséges jogokkal.
*/

public void bizalmas() {
ellenorizPermission(bizalmasPermission);
print();
}

/**
* KISS és NAGY egyaránt végrehajthatja a következőket.
*/

public void jav(String uzenet) {
SajatPermission p = new SajatPermission("jav." + uzenet);
ellenorizPermission(p);
System.out.println("Kiss üzente: " + uzenet);
}

/**
* Ezt bárki végrehajthatja.
*/

private void print() {
System.out.println(bizt);
}

/**
* Ha implementáltuk a biztonsági ellenőrzést, akkor
* használjuk azt, különben pedig közvetlenül
* az AccessController-t
*/

void ellenorizPermission(Permission p) {
SecurityManager sm = System.getSecurityManager();
sm.checkPermission(p);
}
}
```

A biztonságpolitikai tábla valamely sorában elhelyezett `Permission` letiltható vagy engedélyezhető, a letiltott sor pedig törölhető.

A DBMS_JAVA csomagban a fenti tevékenységek végrehajtására a következő metódusok szolgálnak.

Letiltás

```
procedure revoke_permission(permission_schema varchar2,
     permission_type varchar2,
     permission_name varchar2,
     permission_action varchar2)
procedure disable_permission(key number)
```

Engedélyezés

```
procedure enable_permission(key number)
```

Törlés

```
procedure delete_permission(key number)
```

A `revoke_permission` végignézi a teljes biztonságpolitikai táblát és azokat a sorokat választja, amelyek illeszkednek a paraméterekhez. A `disable_permission` viszont csak egyetlen sort kezel, amelynek kulcsát meg kell adni. A kulcsot a DBA_JAVA_POLICY vagy a USER_JAVA _POLICY nézetből nyerhetjük.

Java használata esetén a megfelelő metódusok rendre a következők:

```
void revoke(String schema, String type,
   String name, String action);
void oracle.aurora.rdbms.security.PolicyTableManager.disable(
   long number);
void oracle.aurora.rdbms.security.PolicyTableManager.enable(
   long number);
void oracle.aurora.rdbms.security.PolicyTableManager.delete(
   long number);
```

Egy osztály adatbázisba való betöltéséhez a

`JServerPermission("LoadClassInPackage." +` *osztálynév*`)`

jogosultság szükséges. Az *osztálynév* a betöltendő osztály teljes minősített neve. Ez a jog nem terjed ki a rendszercsomagokba való betöltésre és a rendszerosztályok felülírására.

Az Oracle telepítése után a SYS rendelkezik ezzel a joggal, továbbá minden felhasználó jogosult a saját sémájában az osztályok betöltésére. A más sémába való betöltéshez szükséges a fenti jog.

2.2. Java tárolt alprogramok

2.2.1. Tárolt alprogramok és a valós idejű futtatási környezet

A tárolt alprogramok az SQL számára publikus Java-metódusok, amelyek egy Oracle-adatbázisban vannak tárolva. Ahhoz, hogy egy Java-metódus publikus legyen, írnunk kell egy *hívási specifikációt*, amely leképezi a metódus nevét, paramétereinek típusait, és visszatérési típusát azok SQL-megfelelőire.

A hívási specifikáció egyszerűen hivatkozik egy létező Java-metódusra. Így amikor meghívjuk a metódust (a hívási specifikáció segítségével), a futtató rendszer minimálisan terheli meg a rendszert a hívás lebonyolítása során.

A kliensalkalmazások által hívott tárolt alprogramok fogadják az argumentumokat, hivatkoznak a Java-osztályokra, és visszatérnek a Java-metódusból kapott értékekkel.

A GUI-metódusokat kivéve minden Java-metódus futtatható tárolt alprogramként. A futtatási környezet lehet:

- PL/SQL-függvény és -eljárás,
- adatbázistrigger,
- objektumtípus metódusa.

A `void` típusú Java-metódusnál eljáráshívási, a többinél függvényhívási specifikációt kell alkalmaznunk. Csak önálló vagy csomagbeli, tehát nem lokális eljárások és függvények használhatók hívási specifikációként.

A publikált Java-metódusok explicit módon hívhatók a következő helyeken:

- SQL DML utasítás,
- SQL CALL utasítás,
- PL/SQL-csomag, alprogram, blokk.

A trigger törzse állhat egyetlen olyan CALL utasításból, amely egy Java-metódust hív meg.

Az objektumtípus metódusának deklarációjánál szintén megadhatunk Java-metódus hívási specifikációt.

Tárolt alprogramok futtatásához az adatbázist dedikált szerver vagy MTS módban kell konfigurálni. Részleteket az *Oracle Net Services Administrator's Guide* dokumentációban találhatunk.

2.2.2. A tárolt alprogramok előnyei

Egy nagyméretű adatbázis-alkalmazás megírásánál a tárolt alprogramok előnyei megkönnyítik a munkát. Lássuk ezeket sorban.

Teljesítmény

A tárolt alprogramokat egyszer kell megírni, és utána futtatható formában tárolni őket, így az alprogram meghívása gyors és eredményes lesz. A futtatható kód automatikusan betöltődik és megoszlik a felhasználók között. Ez kevesebb memóriát követel és kevésbé terheli a rendszert.

Az SQL-utasítások csoportosításával a tárolt alprogramok egy egyszerű hívással futtathatók. Ez minimalizálja a hálózat használatát, csökkenti a hálózati ütközéseket és csökkenti a válaszidőket. Az OLTP-alkalmazások különösen előnyösek, mert a folyamat eredménye kiküszöböli a hálózati torlódásokat.

A tárolt alprogramok lehetővé teszik továbbá a szerver erőforrásainak jobb kihasználását. Például, ha egy hálózati eljárás a szerverre másolódik, akkor ott gyorsabban fog futni. Hasonlóan, ha a szerveren futnak az SQL-utasítások által meghívott tárolt alprogramok, nő a teljesítmény.

Hatékonyság és könnyű használat

Az alkalmazások tervezésénél a tárolt alprogramok általános megadásával elkerülhetjük a redundáns kódolást és növelhetjük a hatékonyságot. Továbbá a tárolt alprogramokkal az adatbázis-kezelő rendszer sokoldalúbb lesz. Például tárolt függvény hívása SQL-utasításból növeli az SQL hatékonyságát.

A tárolt alprogramok létrehozásához kiválaszthatjuk a Java fejlesztői környezetet. A tárolt alprogramok így bármely hálózati architektúrán futtathatóak. Ráadásul a standard Java-interfészek, mint a JDBC, CORBA, EJB, valamint a programozói interfészek és fejlesztői eszközök, mint az SQLJ, OCI, Pro*C/C++ és JDeveloper is meghívhatják a tárolt alprogramokat.

A tárolt alprogramok ezen széles körű elérési lehetősége támogatja a munka szétosztását. Például egy adott probléma szerint implementált tárolt alprogramokat minden olyan kliens oldali alkalmazás meghívhat, amely részt vesz a probléma megoldásában. A szerver lehetőségeit még azzal is növelhetjük, hogy a kedvenc programozói felületünkkel dolgozunk.

Skálázhatóság

A tárolt alprogramok növelik a skálázhatóságot azzal, hogy a szerveren elkülönítik az alkalmazások futását. Tárolt alprogramok automatikus követése is a skálázható alkalmazások fejlesztését támogatja.

A többszálas szerver osztottmemória-lehetősége biztosítja, hogy az Oracle9*i* több tízezer párhuzamos felhasználót fogadjon egyetlen csomóponton.

Karbantarthatóság

Ha a tárolt alprogramot egyszer létrehoztuk, akkor bármilyen sokszor felhasználhatjuk. Ha megváltozik, csak maga az alprogram változik, de az alkalmazás, amely meghívja, nem. Ez egyszerűbb kezelést és fejlesztést eredményez. Tehát egy alprogram kezelése sokkal egyszerűbb a szerveren, mint a különböző kliensgépeken lévő másolatok változtatása.

Együttműködés

Az adatbázis-kezelő rendszerben levő Java megfelel a Java Language Specificationnak, és rendelkezik egy univerzális objektumorientált programozási nyelv minden előnyével. A PL/SQL-hez hasonlóan az Oracle-adatokhoz a Java is hozzáfér, így sok PL/SQL-ben megírt alprogramot Javában is meg lehet írni.

A PL/SQL tárolt alprogramok és a Java tárolt alprogramok jól kiegészítik egymást. Jellemzően, azok az SQL-programozók, akik procedurális eszközöket akarnak használni, a PL/SQL-t választják, míg azok a Java-programozók, akik az Oracle-adatok könnyű elérését szeretnék, a Javát választják.

Az adatbázis-kezelő a Java és a PL/SQL között magas fokú együttműködést biztosít. A Java-alkalmazások beágyazott JDBC segítségével PL/SQL tárolt alprogramokat hívhatnak. Viszont PL/SQL-alkalmazások közvetlenül hívhatnak Java tárolt alprogramokat.

Biztonság

Az Oracle-adatok elérését úgy korlátozhatjuk, hogy a felhasználókat az adatokhoz nem engedjük hozzáférni, csak az adatokat kezelő tárolt alprogramokhoz, de a felhasználó azokat is csak a megadott jogokkal futtathatja. Például hozzáférhet az eljáráshoz, amelyik megváltoztat egy adatbázistáblát, de magához a táblához nem.

2.2.3. Tárolt alprogramok fejlesztése

Egy Java tárolt alprogramot a PL/SQL tárolt alprogramokhoz hasonlóan fejleszthetünk és futtathatunk. Ehhez a következő öt lépést kell rendre végrehajtanunk:

1. Osztályok létrehozása vagy újrafelhasználása

Az osztályok írására vagy a már létező osztályok újrafelhasználására használhatjuk a kedvenc Java fejlesztői környezetünket. Az Oracle Java eszközei támogatják a különféle Java fejlesztőeszközöket és kliens oldali programozói interfészeket. Például az Oracle JVM elfogadja azokat a programokat, amelyeket a népszerű Java fejlesztői környezetekben (például Symantec Visual Cafe, Oracle JDeveloper vagy Borland JBuilder) fejlesztettek.

Tekintsük a klasszikus „Hello világ" példát. Az osztály forráskódja a következő (a `Hello.java` állomány tartalmazza):

```
public class Hello
{
  public static String vilag()
  {
  return "Hello világ";
  }
}
```

Fordítsuk le a kliensen a JDK fordítójával:

```
javac Hello.java
```

Ekkor létrejön a `Hello.class` bináris állomány.

2. Java-osztályok betöltése és feloldása

Használjuk a `loadjava`-t a parancssorban ahhoz, hogy a Java forrás-, osztály-, és erőforrás-állományt az adatbázis-kezelő rendszerbe töltsük, ahol azok sémaobjektumként tárolódnak. Példánkban adjuk ki a

```
loadjava -user KISS/K000 Hello.class
```

utasítást. Ebben az esetben az alapértelmezett feloldót fogja használni. A feloldó először végignézi a saját séma, majd a PUBLIC osztályait. Jelenleg csak a `String` osztályt kell megtalálnia. A magosztályok, köztük a `java.lang` csomag osztályai, a PUBLIC-ban találhatók. Így a feloldás sikeres lesz.

3. Java-metódusok publikálása

Minden nem Javából meghívandó metódushoz meg kell adni egy hívási specifikációt. Ha SQL-ből akarjuk meghívni a metódust, akkor egy ún. *felső szintű* hívási specifikáció szükséges. Például SQL*Plusban a példánk metódusát a következő módon láthatjuk el felső szintű hívási specifikációval:

```
SQL> CONNECT KISS/K000
connected
SQL> CREATE OR REPLACE FUNCTION HELLOVILAG RETURN VARCHAR2 AS
2 LANGUAGE JAVA NAME 'Hello.vilag() return java.lang.String';
3 /
Function created.
```

Az SQL és a PL/SQL nem tesz különbséget aközött, hogy egy tárolt alprogram PL/SQL-ben, Javában vagy más nyelven íródott-e meg. A hívási specifikáció elfedi az implementáció részleteit és egy nyelvfüggetlen hozzáférési felületet ad. Éppen ezért hívási specifikáció csak azon belépési pontokhoz szükséges, amelyeken keresztül meghívjuk az alkalmazást. Azok a Java-metódusok, amelyeket más publikált Java-metódusok meghívnak, soha nem publikálandók. Részletesen lásd a 2.2.4. alfejezetben.

4. Java tárolt alprogramok hívása

A Java tárolt alprogramokat meghívhatjuk közvetlenül SQL DML utasításból, PL/SQL-blokkból vagy alprogramból. Az SQL CALL utasításával a tárolt alprogram felső szintről (például SQL*Plus) és adatbázistriggerből is meghívható. Például

```
SQL> VARIABLE SZOVEG VARCHAR2[20];
SQL> CALL HELLOVILAG() INTO :SZOVEG;
Call completed.
SQL> PRINT SZOVEG;

SZOVEG
------------------
Hello világ

SQL>
```

Ugyancsak szabályos a

```
SELECT HELLOVILAG() FROM DUAL;
```

utasítás is. Részletesen lásd a 2.2.8. alfejezetben.

5. Java tárolt alprogramok tesztelése

A JDK belövő eszköze (a `jdb`) nem használható távoli Java-programok esetén. Az Oracle9*i* ilyen esetben a `DebugProxy` osztályt bocsátja a rendelkezésünkre. Ennek segítségével úgy kezelhetjük a távoli Java-programot, mintha az lokális lenne. A proxy továbbítja a kérést a szerverhez és közvetíti a választ a belövőhöz.

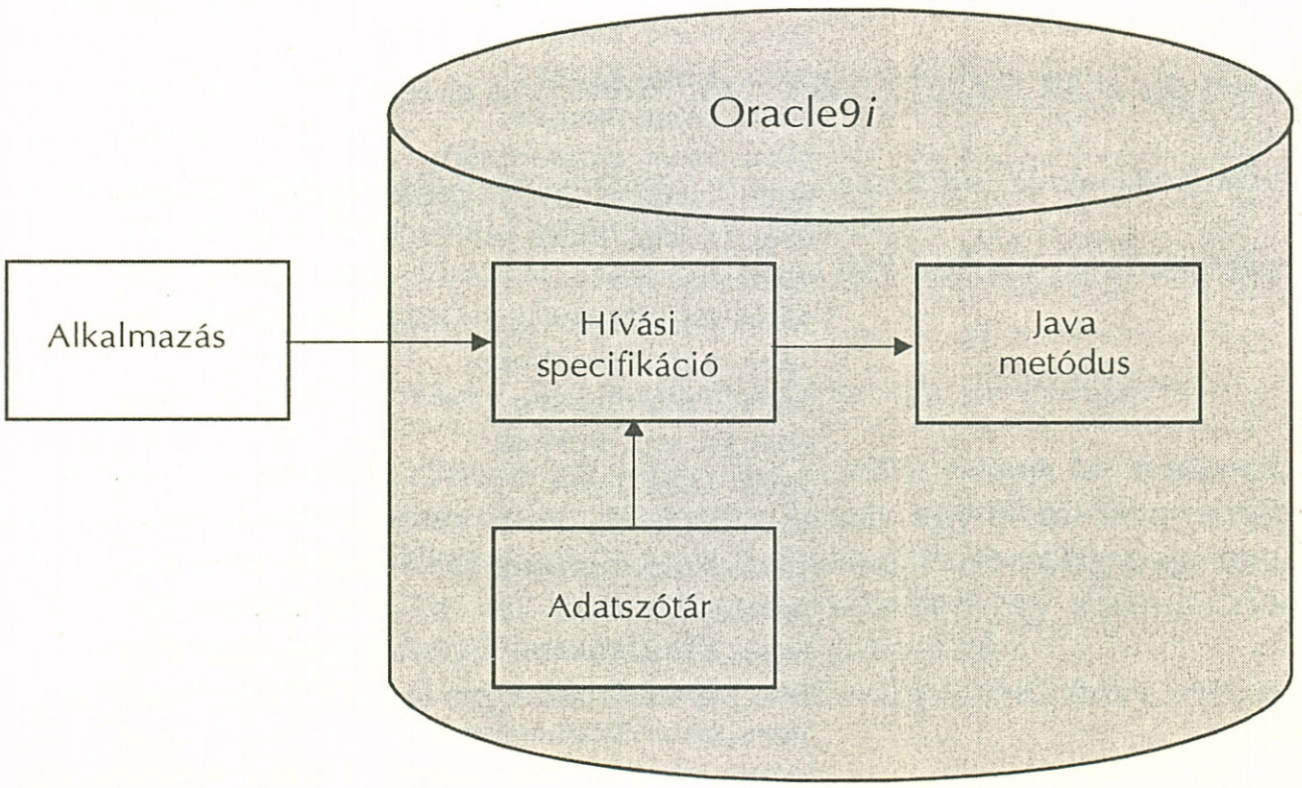
2.2.4. Tárolt alprogramok publikálása

Mielőtt meghívnánk egy Java-metódust, publikussá kell tennünk az Oracle adatszótárban. Amikor betöltünk egy Java-osztályt az adatbázis-kezelő rendszerbe, a Java-osztály metódusai nem lesznek automatikusan publikusak, mert az Oracle nem tudja, melyik metódus lehet belépési pont. Egy Java-metódus publikussá tételéhez egy hívási specifikációt kell írnunk, amely leképezi a Java-metódus nevét, paramétereinek típusait és visszatérési típusát azok SQL-megfelelőire.

Egy adott Java-metódushoz egy függvényt vagy eljárást kell deklarálnunk, amit megtehetünk az SQL CREATE FUNCTION vagy CREATE PROCEDURE utasításával. PL/SQL-csomagban vagy objektumtípusban pedig egyszerűen deklarációt alkalmazunk.

A visszatérési értékkel rendelkező Java-metódusokat függvénynek, a `void` Java-metódusokat eljárásnak publikáljuk. A függvények és eljárások törzsében a `LANGUAGE JAVA` opciónak kell szerepelnie. Ez az opció tartalmazza a Java-metódus információit: a teljes nevet, a paraméterek típusát és a visszatérési típust.

A 2.3. ábrán azt láthatjuk, hogyan hívják meg az alkalmazások a hívási specifikáción keresztül a Java-metódust. A futtató rendszer átnézi a hívási specifikáció definícióit az Oracle adatszótárában, és lefuttatja a megfelelő Java-metódust.



2.3. ábra. *Java-metódus hívása*

A Java-metódusnak és az őt publikáló hívási specifikációnak ugyanabban a sémában kell lennie. Hívási specifikációt a következőképpen adhatunk meg:

- önálló PL/SQL-függvényként vagy eljárásként (felső szintű specifikáció);

- PL/SQL-csomag függvényeként vagy eljárásaként;
- objektumtípus metódusaként.

Csak `public static` metódusokat publikálhatunk, ez alól egyetlen kivétel van: példánymetódusokat publikussá tehetünk objektumtípus tagmetódusaiként.

Egy hívási specifikációban a megfelelő SQL- és Java-paramétereknek (és függvényértékeknek) kompatibilis adattípusúaknak kell lenniük. A 2.1. táblázatban megadjuk az SQL-osztályoknak megfelelő legális Java-osztályokat. Az Oracle az SQL-típusokat és a Java-osztályokat automatikusan konvertálja.

2.1. táblázat. *SQL-típusoknak megfelelő legális Java-osztályok*

SQL-típus	*Java-osztály*
`CHAR, LONG, VARCHAR2`	`oracle.sql.CHAR`
	`java.lang.String`
	`java.sql.Date`
	`java.sql.Time`
	`java.sql.Timestamp`
	`java.lang.Byte`
	`java.lang.Short`
	`java.lang.Integer`
	`java.lang.Long`
	`java.lang.Float`
	`java.lang.Double`
	`java.math.BigDecimal`
	`byte, short, int, long, float, double`
`DATE`	`oracle.sql.DATE`
	`java.sql.Date`
	`java.sql.Time`
	`java.sql.Timestamp`
	`java.lang.String`
`NUMBER`	`oracle.sql.NUMBER`
	`java.lang.Byte`
	`java.lang.Short`
	`java.lang.Integer`
	`java.lang.Long`
	`java.lang.Float`
	`java.lang.Double`
	`java.math.BigDecimal`
	`byte, short, int, long, float, double`
`OPAQUE`	`oracle.sql.OPAQUE`
`RAW, LONG RAW`	`oracle.sql.RAW`
	`byte[]`
`ROWID`	`oracle.sql.CHAR`
	`oracle.sql.ROWID`
	`java.lang.String`

2.1. táblázat. *SQL-osztályoknak megfelelő legális Java-osztályok (folytatás)*

SQL-típus	*Java-osztály*
`BFILE`	`oracle.sql.BFILE`
`BLOB`	`oracle.sql.BLOB`
	`oracle.jdbc2.Blob`
`CLOB, NCLOB`	`oracle.sql.CLOB`
	`oracle.jdbc2.Clob`
`OBJECT`	`oracle.sql.STRUCT`
	`oracle.sql.ORAData`
	`java.sql.Struct`
	`java.sql.SqlData`
`REF`	`oracle.sql.REF`
	`oracle.sql.ORAData`
	`java.sql.Ref`
`TABLE, VARRAY`	`oracle.sql.ARRAY`
	`oracle.sql.ORAData`
	`java.sql.Array`
a fenti SQL-típusok közül bármelyik	`oracle.sql.CustomDatum`
	`oracle.sql.Datum`

Megjegyzések a táblázathoz:

1. Az UROWID típus és a NUMBER altípusok (INTEGER, REAL stb.) nem támogatottak.
2. A Java burkoló osztályai (például `java.lang.Byte`, `java.lang.Short`) alkalmasak arra, hogy NULL értékeket kezeljenek.
3. Ha `oracle.sql.CustomDatum` osztállyal deklarálunk paramétereket, akkor a következő módszert implementálni kell:

```
public static oracle.sql.CustomDatumFactory.getFactory();
```

4. Az `oracle.sql.Datum` egy absztrakt osztály. Annak az értéknek, amit az `oracle.sql.Datum` típusú paraméter kapott, egy olyan Java-osztályhoz kell tartoznia, amelyik kompatibilis az SQL-típussal. Hasonlóképp egy metódus által visszaadott `oracle.sql.Datum` típusú értéknek is egy olyan Java-osztályhoz kell tartoznia, amelyik kompatibilis az SQL-típussal.
5. Javában 32 KB-nál nagyobb méretű LONG vagy LONG RAW érték nem kezelhető.

2.2.5. Hívási specifikáció SQL*Plusban

Az SQL*Plusban felső szintű hívási specifikációt interaktívan a következő szintaktikával adhatunk meg:

```
CREATE [OR REPLACE]
{   PROCEDURE eljárás_név [(paraméter[, paraméter]...)]
 | FUNCTION függvény_név [(paraméter[, paraméter]...)] RETURN sql_típus}
[AUTHID {DEFINER | CURRENT_USER}]
[PARALLEL_ENABLE]
[DETERMINISTIC]
{IS | AS} LANGUAGE JAVA
NAME 'metódus_teljes_neve (java_típus_teljes_neve[, java_típus_teljes_neve]...)
  [RETURN java_típus_teljes_neve]';
```

ahol a *paraméter* a következő alakú:

```
név [IN | OUT | IN OUT] sql_típus
```

Az AUTHID eldönti, hogy egy tárolt alprogram a létrehozó (DEFINER) vagy a hívó (CURRENT_USER – ez az alapértelmezett) jogosultságaival futtatható-e és hogy vannak-e feltétel nélküli hivatkozások olyan sémaobjektumokra, amelyek a létrehozó vagy a hívó sémájában vannak. A `loadjava -definer` opcióját nem tudjuk a CURRENT_USER megadásával felülbírálni.

A PARALLEL_ENABLE opció megadja, hogy a tárolt függvény biztonságosan használható a párhuzamos DML-kiértékelések kiszolgáló munkamenetében. Egy fő munkamenet állapota soha nem lehet megosztva kiszolgáló munkamenettel. A kiszolgáló munkameneteknek van saját állapotuk, amely a folyamat kezdetekor inicializálódik. A függvény eredménye nem függhet a munkamenet `static` változóinak állapotától, máskülönben az eredményt módosíthatják a munkamenetek.

A DETERMINISTIC segít az optimalizálónak elkerülni a redundáns függvényhívásokat. Ha egy tárolt függvényt előzőleg ugyanazokkal az argumentumokkal hívtunk meg, akkor az optimalizáló megválaszthatja, hogy az előző eredményt használja-e vagy sem. A függvény eredménye nem függhet a munkamenet-változók vagy sémaobjektumok állapotától. Máskülönben az eredmény módosulhat a hívások által. Csak DETERMINISTIC függvényeket lehet meghívni egy függvény alapú indexből és kérdésfelülírási engedéllyel rendelkező materializált nézetből.

A NAME utasításrész adja meg a Java-metódus teljes minősített nevét, a paraméterek típusát és függvénynél a visszatérési típust. A következőkben lássunk néhány konkrét példát.

2.1. példa

Tegyük fel, hogy a következő Java-osztály be van töltve az adatbázis-kezelő rendszerbe:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;

public class AltTorol {
  public static void torold
     (String objektum_tipus, String objektum_nev)
    throws SQLException {
    // Kapcsolódás az Oracle-hoz JDBC-meghajtóval.
    // Itt az Oracle JVM által alapértelmezett
    // kapcsolat lekérdezése történik meg.
    Connection conn =
      DriverManager.getConnection("jdbc:default:connection:");
    // SQL-utasítás összeállítása
    String sql = "DROP " + objektum_tipus + " " + objektum_nev;
    try {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate(sql);
      stmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }
}
```

Az `AltTorol` osztály `torold` nevű metódusa a sémaobjektumok minden típusát eltávolítja. Például ha a `torold`-nek a „TABLE" és „dolg" aktuális paramétereket adjuk, akkor a metódus a `dolg` táblát eltávolítja a sémából. Írjunk erre a metódusra egy hívási specifikációt:

```
CREATE OR REPLACE PROCEDURE torol(
  obj_tipus VARCHAR2,
  obj_nev VARCHAR2)
AS LANGUAGE JAVA
NAME 'AltTorol.torold(java.lang.String, java.lang.String)';
```

Jegyezzük meg, hogy meg kell adnunk a `String` osztály teljes minősítését. A `java.lang` csomag ugyan automatikusan elérhető a Java-programok számára, de explicit módon meg kell nevezni a hívási specifikációban.

2.2. példa

Ebben a példában a `SorSzamol` Java-metódust tesszük publikussá. A metódus egy adott tábla sorainak számát adja vissza:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;
```

```
public class SorSzamol {
  public static int sorSzam (String tablaNev) throws SQLException {
    Connection conn =
    DriverManager.getConnection("jdbc:default:connection:");
    String sql = "SELECT COUNT(*) FROM " + tablaNev;
    int sor = 0;
    try {
      Statement stmt = conn.createStatement();
      ResultSet rset = stmt.executeQuery(sql);
      while (rset.next()) {sor = rset.getInt(1);}
      rset.close();
      stmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
    return sor;
  }
}
```

A következő hívási specifikáció visszatérési típusa NUMBER, nem pedig INTEGER, mert NUMBER altípusok a hívási specifikációban nem használhatók:

```
CREATE FUNCTION sor_szam (tabla_nev VARCHAR2) RETURN NUMBER
AS LANGUAGE JAVA
NAME 'SorSzamol.sorSzam(java.lang.String) return int';
```

2.3. példa

A következőkben a `csere` nevű Java-metódust fogjuk publikálni, amely megcseréli az argumentumainak az értékét.

```
public class Cserel {
  public static void csere (int[] x, int[] y) {
    int cs = x[0];
    x[0] = y[0];
    y[0] = cs;
  }
}
```

A hívási specifikáció publikussá teszi a `csere` Java-metódust `csere` néven. A hívási specifikáció IN OUT formális paramétereket határoz meg. A hívási specifikáció OUT és IN OUT paramétereinek Java-tömbparaméterek felelnek meg.

```
CREATE PROCEDURE csere (x IN OUT NUMBER, y IN OUT NUMBER)
AS LANGUAGE JAVA
NAME 'Cserel.csere(int[], int[])';
```

Jegyezzük meg, hogy egy Java-metódus neve és a hívási specifikációjának neve megegyezhet.

2.2.6. Hívási specifikáció csomagban

A PL/SQL-csomag egy sémaobjektum. A csomagoknak általában két része van, a specifikáció és a törzs. A specifikáció adja az interfészt az alkalmazásokhoz, deklarálja a típusokat, a nevesített konstansokat, a változókat, a kivételeket, a kurzorokat és az alprogramok specifikációját. A törzs teljesen definiálja a kurzorokat és az alprogramokat, ezáltal implementálja a specifikációt.

Az SQL*Plusban interaktívan adhatunk meg PL/SQL-csomagot a következő szintaxissal:

```
CREATE [OR REPLACE] PACKAGE csomag_név
  [AUTHID {CURRENT_USER | DEFINER}] {IS | AS}
  [típus_definició [típus_definició] ...]
  [kurzor_specifikáció [kurzor_specifikáció] ...]
  [deklaráció [deklaráció] ...]
  [{alprogram_specifikáció | hívási_specifikáció}
  [{alprogram_specifikáció | hívási_specifikáció}]...]
END [csomag_név];

[CREATE [OR REPLACE] PACKAGE BODY csomag_név {IS | AS}
  [típus_definició [típus_definíció] ...]
  [kurzor_definíció [kurzor_definíció] ...]
  [deklaráció [deklaráció] ...]
  [{alprogram_deklaráció | hívási_spec} [{alprogram_deklaráció | hívási_spec}]...]
[BEGIN
 utasítások_sorozata]
END [csomag_név];]
```

A specifikáció az alkalmazások számára látható, publikus deklarációkat tartalmaz. A törzs implementációs részleteket és privát deklarációkat tartalmaz, amelyek az alkalmazások számára rejtve maradnak. A BEGIN alapszó utáni opcionális részben általában a csomag változóinak inicializációjára vonatkozó utasítások vannak. Ez a rész egyszer fut le, amikor először hivatkozunk a csomagra. Részletesen lásd [4].

A csomagban elhelyezett hívási specifikáció neve és paraméterlistája nem egyezhet meg a csomagban lévő egyéb alprogramok nevével és paraméterlistájával, tehát egy alprogram neve nem terhelhető túl hívási specifikációval.

2.4. példa

Tekintsük a `ReszlegKezelo` Java-osztályt, amelynek a metódusai új részleg megadására, részleg törlésére vagy részleg telephelyének megváltoztatására szolgálnak. Jegyezzük meg, hogy az `ujReszleg` egy szekvenciát használ, amely a következő új részleg számát adja meg. A három metódus logikailag összekapcsolódik, ezért a hívási specifikációjuk egy PL/SQL-csomagba kerülhet.

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;

public class ReszlegKezelo {
  public static void ujReszl (String reszlNev, String reszlCim)
  throws SQLException {
    Connection conn =
     DriverManager.getConnection("jdbc:default:connection:");
    String sql = "SELECT reszlsorsz.NEXTVAL FROM dual";
    String sql2 = "INSERT INTO reszl VALUES (?, ?, ?)";
    int reszlAZ = 0;
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      ResultSet rset = pstmt.executeQuery();
      while (rset.next()) {reszlAZ = rset.getInt(1);}
      pstmt = conn.prepareStatement(sql2);
      pstmt.setInt(1, reszlAZ);
      pstmt.setString(2, reszlNev);
      pstmt.setString(3, reszlCim);
      pstmt.executeUpdate();
      rset.close();
      pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }

  public static void torolReszl (int reszlAZ) throws SQLException {
    Connection conn =
     DriverManager.getConnection("jdbc:default:connection:");
    String sql = "DELETE FROM reszl WHERE reszlsorsz = ?";
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setInt(1, reszlAZ);
      pstmt.executeUpdate();
      pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }

  public static void cimValt (int reszlAZ, String ujCim)
  throws SQLException {
    Connection conn =
     DriverManager.getConnection("jdbc:default:connection:");
    String sql = "UPDATE reszl SET cim = ? WHERE reszlsorsz = ?";
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setString(1, ujCim);
      pstmt.setInt(2, reszlAZ);
      pstmt.executeUpdate();
      pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }
}
```

Először hozzuk létre a csomag specifikációját:

```
CREATE OR REPLACE PACKAGE reszl_kez AS
  PROCEDURE uj_reszl (reszl_nev VARCHAR2, reszl_cim VARCHAR2);
  PROCEDURE torol_reszl (reszl_az NUMBER);
  PROCEDURE cim_valt (reszl_az NUMBER, uj_cim VARCHAR2);
END reszl_kez;
```

Ezután a csomag törzsét hozzuk létre a Java-metódusok hívási specifikációjának megadásával:

```
CREATE OR REPLACE PACKAGE BODY reszl_kez AS
  PROCEDURE uj_reszl (reszl_nev VARCHAR2, reszl_cim VARCHAR2)
  AS LANGUAGE JAVA
  NAME 'ReszlegKezelo.ujReszl(java.lang.String, java.lang.String)';

  PROCEDURE torol_reszl (reszl_az NUMBER)
  AS LANGUAGE JAVA
  NAME 'ReszlegKezelo.torolReszl(int)';

  PROCEDURE cim_valt (reszl_az NUMBER, uj_cim VARCHAR2)
  AS LANGUAGE JAVA
  NAME 'ReszlegKezelo.cimValt(int, java.lang.String)';
END reszl_kez;
```

A `reszl_kez` csomagban levő tárolt alprogramokra a következő minősítéssel hivatkozhatunk:

```
CALL reszl_kez.uj_reszl('KONYVELES', 'DEBRECEN');
```

2.2.7. Hívási specifikáció objektumtípusban

Az Oracle-ben az objektumrelációs eszközök az objektumtípusokon alapulnak. Az objektumtípus egy felhasználó által definiált adattípus, amelyben az adatstruktúra és az adatokat kezelő függvények és eljárások vannak egységbe zárva. Az adatstruktúra elemeit attribútumoknak hívjuk, az adattípus viselkedését meghatározó függvényeket és eljárásokat pedig metódusoknak. Ezeket a metódusokat Javában is megírhatjuk.

A csomaghoz hasonlóan az objektumtípusnak is két része van: a specifikáció és a törzs. A specifikáció adja az interfészt az alkalmazásokhoz, deklarálja az adatstruktúrát és a metódusok specifikációját. A törzs implementálja a metódusokat PL/SQL-ben vagy Java hívási specifikációk megadásával. Ha minden metódus Javában van megírva, akkor a hívási specifikációt elhelyezhetjük az objektumtípus specifikációs részében és elhagyhatjuk a törzset.

AZ SQL*Plusban objektumtípusokat interaktívan definiálhatunk a következő szintaktikával:

```
CREATE [OR REPLACE] TYPE típus_név
  [AUTHID {CURRENT_USER | DEFINER}]
  {{IS | AS} OBJECT | UNDER szupertípus_név}
  (attribútum_név adattípus[, attribútum_név adattípus]...
  [{MAP | ORDER} MEMBER {függvény_spec | hívási_spec}]
  [{MEMBER | STATIC} {alprogram_spec | hívási_spec}
  [, {MEMBER | STATIC} {alprogram_spec | hívási_spec}]...]
);

[CREATE [OR REPLACE] TYPE BODY típus_név {IS | AS}
  { {MAP | ORDER} MEMBER függvény_deklaráció;
    | {MEMBER | STATIC} {alprogram_deklaráció | hívási_spec};}
  [{MEMBER | STATIC} {alprogram_deklaráció | hívási_spec};]...
END;]
```

Részleteket illetően lásd [4].

2.5. példa

Először két SQL-objektumtípust hozunk létre `Reszleg` és `Dolgozo` névvel. Először a `Reszleg` objektumtípus specifikációját írjuk meg. A törzset elhagyhatjuk, mert a specifikáció csak attribútumokat tartalmaz.

```
CREATE TYPE Reszleg AS OBJECT (
  reszlsorsz NUMBER(2),
  reszlnev   VARCHAR2(14),
  cim        VARCHAR2(13)
);
```

Aztán a `Dolgozo` objektumtípust adjuk meg. Az utolsó attribútum, a `reszlsorszam` egy referenciát tárol, amely egy `Reszleg` típusú objektumra hivatkozik. A referencia egy objektumtáblabeli objektumra mutat. Az objektumtábla egy olyan adatbázistábla, amelyben egy objektumtípus példányai vannak tárolva.

```
CREATE TYPE Dolgozo AS OBJECT (
  dolgsorsz  NUMBER(4),
  dolgnev    VARCHAR2(10),
  munkakor   VARCHAR2(9),
  fonok      NUMBER(4),
  belep_dat  DATE,
  fizetes    NUMBER(7,2),
  jutalom    NUMBER(7,2),
  reszlsorsz REF Reszleg
);
```

A következő példában bemutatjuk, hogyan hozunk létre `Reszleg` és `Dolgozo` típusú objektumokat tartalmazó objektumtáblákat. Először létrehozunk egy `reszlegek`

objektumtáblát, amely a `Reszleg` típus objektumait tartalmazza. Az objektumtáblát úgy tölthetjük fel, hogy a `reszl` relációs táblából leválogatott adatokat a konstruktornak adjuk. A konstruktor létrehoz egy objektumtípus példányt és visszatérési értéke ezen példányra hivatkozó referencia lesz.

```
CREATE TABLE reszlegek OF Reszleg AS
  SELECT Reszleg(reszlsorsz, reszlnev, cim) FROM reszl;
```

Aztán a `dolgozok` objektumtáblát hozzuk létre, amely `Dolgozo` objektumokat tartalmaz. A `dolgozok` objektumtábla utolsó oszlopa, amely megegyezik a `Dolgozo` objektumtípus utolsó attribútumával, hivatkozásokat tartalmaz a `Reszleg` típus objektumaira. A REF operátort használjuk arra, hogy a hivatkozások ebbe az oszlopba kerüljenek. A REF operátor a `reszlegek` tábla másodlagos nevét kapja argumentumként.

```
CREATE TABLE dolgozok OF Dolgozo AS
  SELECT Dolgozo(e.dolgsorsz, e.dolgnev, e.munkakor, e.fonok, e.belepdat,
         e.fizetes, e.jutalom,
         (SELECT REF(d) FROM reszlegek d
           WHERE d.reszlsorsz = e.reszlsorsz))
  FROM dolg e;
```

Írjuk meg Javában a `Kifizeto` osztályt, amelynek egy metódusa egy dolgozó jövedelmét számolja ki. A kódban az `oracle.sql.STRUCT` osztályban definiált `getAttributes()` metódust használjuk. Ez a metódus az alapértelmezett JDBC-t használja az attribútumtípusok lekérésére. Például a `NUMBER` a `BigDecimal`-nak felel meg.

```
import java.sql.*;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.*;
import oracle.oracore.*;
import oracle.jdbc2.*;
import java.math.*;

public class Kifizeto {
  public static BigDecimal jovedelem(STRUCT e)
    throws java.sql.SQLException {
    // Lekérjük a Dolgozó objektum attribútumát.
    Object[] attrib = d.getAttributes();
    BigDecimal fizetes = (BigDecimal)(attrib[5]);
    BigDecimal jutalom = (BigDecimal)(attrib[6]);
    BigDecimal jov = fizetes;
    if (jutalom != null) jov = jov.add(jutalom);
    return jov;
  }
}
```

A `jovedelem` metódus értékkel tér vissza, ezért a következő felső szintű hívási specifikációt írhatjuk rá:

```
CREATE OR REPLACE FUNCTION jovedelem (e Dolgozo) RETURN NUMBER AS
  LANGUAGE JAVA
  NAME 'Kifizeto.jovedelem(oracle.sql.STRUCT) return BigDecimal';
```

Az objektum attribútumértékeinek könnyebb elérése érdekében az objektumhoz egy Java-osztályt hozhatunk létre, amely az `SQLData` interfészt implementálja. Ekkor gondoskodnunk kell a `readSQL()` és `writeSQL()` metódusok implemetálásáról. A JDBC-meghajtó a `readSQL()` metódust hívja meg az adatbázis olvasásához.

A következő kód segítségével módosítjuk a `Kifizeto` osztályt, bővítjük a `fizEmel()` metódussal:

```
import java.sql.*;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.*;
import oracle.oracore.*;
import oracle.jdbc2.*;
import java.math.*;

public class Kifizeto implements SQLData {
  // Az adott típus attribútumainak és műveleteinek megadása.
  private BigDecimal dolgsorsz;
  private String dolgnev;
  private String munkakor;
  private BigDecimal fonok;
  private Date belep_dat;
  private BigDecimal fizetes;
  private BigDecimal jutalom;
  private Ref reszl;

  public static BigDecimal jovedelem(Kifizeto e) {
    BigDecimal jov = e.fiz;
    if (e.jutalom != null) jov = jov.add(e.jutalom);
    return jov;
  }

  public static void fizEmel(Kifizeto[] e, BigDecimal osszeg) {
    e[0].fizetes = e[0].fizetes.add(osszeg);
  }

 // Az SQLData interfész implementálása

  private String sql_tipus;

  public String getSQLTypeName() throws SQLException {
    return sql_tipus;
  }
```

```
  public void readSQL(SQLInput stream, String tipusNev)
    throws SQLException {
    sql_tipus = tipusNev;
    dolgsorsz = stream.readBigDecimal();
    dolgnev = stream.readString();
    munkakor = stream.readString();
    fonok = stream.readBigDecimal();
    belep_dat = stream.readDate();
    fizetes = stream.readBigDecimal();
    jutalom = stream.readBigDecimal();
    reszl = stream.readRef();
  }

  public void writeSQL(SQLOutput stream) throws SQLException {
    stream.writeBigDecimal(dolgsorsz);
    stream.writeString(dolgnev);
    stream.writeString(munkakor);
    stream.writeBigDecimal(fonok);
    stream.writeDate(belep_dat);
    stream.writeBigDecimal(fizetes);
    stream.writeBigDecimal(jutalom);
    stream.writeRef(reszl);
  }
}
```

Módosítanunk kell a `jovedelem` metódus hívási specifikációját is, mert a paraméterének típusa megváltozott `oracle.sql.STRUCT`-ról `Kifizeto`-re:

```
CREATE OR REPLACE FUNCTION jovedelem (e Dolgozo) RETURN NUMBER AS
  LANGUAGE JAVA
  NAME 'Kifizeto.jovedelem(Kifizeto) return BigDecimal';
```

Mivel az új `fizEmel` metódus `void`, ezért például a következő felső szintű hívási specifikációt írhatjuk hozzá:

```
CREATE OR REPLACE PROCEDURE fiz_emel (e IN OUT Dolgozo, r NUMBER)
  AS LANGUAGE JAVA
  NAME 'Kifizeto.fizEmel(Kifizeto[], java.math.BigDecimal)';
```

Adjunk most a `jovedelem()` és `fizEmel()` metódusokhoz objektumtípusban elhelyezett hívási specifikációkat. Természetesen előzőleg a fent létrehozott felső szintű hívási specifikációkat törölni kell.

```
CREATE TYPE Dolgozo AS OBJECT (
  dolgsorsz  NUMBER(4),
  dolgnev    VARCHAR2(10),
  munkakor   VARCHAR2(9),
  fonok      NUMBER(4),
  belep_dat  DATE,
  fizetes    NUMBER(7,2),
  jutalom    NUMBER(7,2),
```

```
    reszlsorsz REF Reszleg,
    MEMBER FUNCTION jovedelem RETURN NUMBER
      AS LANGUAGE JAVA
      NAME 'Kifizeto.jovedelem() return java.math.BigDecimal',
    MEMBER PROCEDURE fiz_emel (f NUMBER)
      AS LANGUAGE JAVA
      NAME 'Kifizeto.fizEmel(java.math.BigDecimal)'
);
```

Most viszont a `Kifizeto` osztályt módosíthatjuk, ugyanis nem kell tömböt átadni a `fizEmel()` metódusnak, mert az objektumtípusok metódusainak implicit IN OUT módú SELF paramétere közvetlenül megfelel a Java `this` pszeudóváltozójának.

```
import java.sql.*;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.*;
import oracle.oracore.*;
import oracle.jdbc2.*;
import java.math.*;

public class Kifizeto implements SQLData {
  // A típus attribútumainak és műveleteinek implementálása.
  private BigDecimal dolgsorsz;
  private String dolgnev;
  private String munkakor;
  private BigDecimal fonok;
  private Date belep_dat;
  private BigDecimal fizetes;
  private BigDecimal jutalom;
  private Ref reszl;

  public BigDecimal jovedelem() {
    BigDecimal jov = fizetes;
    if (jutalom != null) jov = jov.add(jutalom);
    return jov;
  }

  public void fizEmel(BigDecimal osszeg) {
    fizetes = fizetes.add(osszeg);
  }

  // Az SQLData interfész implementálása.

  String sql_tipus;

  public String getSQLTypeName() throws SQLException {
    return sql_tipus;
  }

  public void readSQL(SQLInput stream, String tipusNev)
    throws SQLException {
```

```
        sql_tipus = tipusNev;
        dolgsorsz = stream.readBigDecimal();
        dolgnev = stream.readString();
        munkakor = stream.readString();
        fonok = stream.readBigDecimal();
        belep_dat = stream.readDate();
        fizetes = stream.readBigDecimal();
        jutalom = stream.readBigDecimal();
        reszl = stream.readRef();
    }

    public void writeSQL(SQLOutput stream) throws SQLException {
        stream.writeBigDecimal(dolgsorsz);
        stream.writeString(dolgnev);
        stream.writeString(munkakor);
        stream.writeBigDecimal(fonok);
        stream.writeDate(belep_dat);
        stream.writeBigDecimal(fizetes);
        stream.writeBigDecimal(jutalom);
        stream.writeRef(reszl);
    }
}
```

2.2.8. Java tárolt alprogramok meghívása

Miután betöltöttük és publikussá tettük a Java tárolt alprogramokat, meghívhatjuk őket. Ez a fejezet bemutatja, hogyan hívhatjuk meg a Java tárolt alprogramokat a különböző környezetekből: felső szintről, adatbázistriggerekből, SQL DML-utasításokból, és PL/SQL-blokkokból.

Java tárolt alprogram önálló meghívása

Az SQL CALL utasítása meghívja a publikussá tett Java-metódust. SQL*Plusban futtathatjuk a CALL utasítást interaktívan a következő szintaxissal:

```
CALL [séma_név.][{csomag_név | objektumtípus_név}][@ablink_név]
{ eljárás_hívás| függvény_hívás INTO :gazdaváltozó};
```

A gazdaváltozó nem jelenhet meg kétszer ugyanabban a CALL utasításban. Egy paraméter nélküli alprogramot pedig üres paraméterlistáva kell meghívni.

```
CALL csere(:x, :x); - HIBA!!
CALL egyenleg() INTO :aktualis_egyenleg;
```

2.6. példa

Ebben a példában a `main()` metódus megkapja egy adatbázistábla nevét és egy tetszőleges feltételt. A metódus törli a táblából azokat a sorokat, amelyek eleget tesznek a feltételnek. Ha nem adunk meg feltételt, az összes sort kitörli a táblából.

```
import java.sql.*;
import oracle.jdbc.*;

public class Torol{
  public static void main (String[] arg) throws SQLException {
    Connection conn =
     DriverManager.getConnection("jdbc:default:connection:");
    String sql = "DELETE FROM " + arg[0];
    if (arg.length > 1) sql += " WHERE " + arg[1];
    try {
      Statement stmt = conn.createStatement();
      stmt.executeUpdate(sql);
      stmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }
}
```

A `main()` metódus egy vagy két argumentumot kaphat. A hívási specifikációban nem adhatunk kezdőértéket a paramétereknek, de a csomagbeli alprogramok nevei túlterhelhetők. Erre látunk példát a következőkben:

```
CREATE OR REPLACE PACKAGE csomag AS
  PROCEDURE sor_torol (tabla_nev VARCHAR2);
  PROCEDURE sor_torol (tabla_nev VARCHAR2, feltetel VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY csomag AS
  PROCEDURE sor_torol (tabla_nev VARCHAR2)
  AS LANGUAGE JAVA
  NAME 'Torol.main(java.lang.String[])';

  PROCEDURE sor_torol (tabla_nev VARCHAR2, feltetel VARCHAR2)
  AS LANGUAGE JAVA
  NAME 'Torol.main(java.lang.String[])';
END;
```

Most már meghívhatjuk a `sor_torol` eljárást:

```
SQL> CALL csomag.sor_torol('dolg', 'fizetes > 150000');

Call completed.

SQL> SELECT dolgnev, fizetes FROM dolg;
```

```
DOLGNEV   FIZETES
-------- --------
KOVÁCS       80000
EGYED       125000
VARGA       125000
BALOGH      150000
TÖRÖK       110000
HUGLI        95000
FEKETE      130000

7 rows selected.
```

2.7. példa

Feltesszük, hogy a következő futtatható Java-osztály az adatbázis-kezelő rendszerben tárolva van:

```
public class Fibonacci {
  public static int fib (int n) {
    if (n == 1 || n == 2)
      return 1;
    else
      return fib(n - 1) + fib(n - 2);
  }
}
```

A `Fibonacci` osztály `fib` nevű metódusa az n-edik Fibonacci-számot adja vissza. A Fibonacci-sorozat (1, 1, 2, 3, 5, 8, 13, 21, ...) rekurzív, először a nyulak szaporodásának modellezésére használták. A sorozat elemeit (a második után) úgy kaphatjuk meg, hogy az adott elemet megelőző két elem összegét vesszük. A metódus értékkel tér vissza, ezért függvényként publikáljuk.

```
CREATE OR REPLACE FUNCTION fib (n NUMBER) RETURN NUMBER
AS LANGUAGE JAVA
NAME 'Fibonacci.fib(int) return int';
```

Most deklaráljunk két SQL*Plus gazdaváltozót, és inicializáljuk az elsőt:

```
SQL> VARIABLE n NUMBER
SQL> VARIABLE f NUMBER
SQL> EXECUTE :n := 7;

PL/SQL procedure successfully completed.
```

Végül meghívhatjuk a `fib` függvényt.

```
SQL> CALL fib(:n) INTO :f;

Call completed.
```

```
SQL> PRINT f

         F
----------
        13
```

Java tárolt alprogram hívása triggerből

2.8. példa

Egy olyan triggert hozunk létre, amelyik a következő Java-osztályt használja a fizetésemelés naplózására:

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;

public class ABTrigger {
  public static void FizNapl (int dolgAZ, float regiFiz, float ujFiz)
  throws SQLException {
    Connection conn =
      DriverManager.getConnection("jdbc:default:connection:");

    String sql = "INSERT INTO fiz_kov VALUES (?, ?, ?)";
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setInt(1, dolgAZ);
      pstmt.setFloat(2, regiFiz);
      pstmt.setFloat(3, ujFiz);
      pstmt.executeUpdate();
      pstmt.close();
    } catch (SQLException e) {System.err.println(e.getMessage());}
  }
}
```

Az `ABTrigger` osztály `FizNapl()` metódusa egy sort szúr be a `fiz_kov` adatbázistáblába. A `FizNapl` egy `void` metódus, ezért eljárásként tesszük publikussá:

```
CREATE OR REPLACE PROCEDURE fiz_napl (
  dolg_az NUMBER, regi_fiz NUMBER, uj_fiz NUMBER)
AS LANGUAGE JAVA
NAME 'ABTrigger.FizNapl(int, float, float)';
```

Most létrehozzuk a `fiz_kov` adatbázistáblát:

```
CREATE TABLE fiz_kov (
  dolgsorsz  NUMBER,
  regifiz    NUMBER,
  ujfiz      NUMBER);
```

Végül pedig egy olyan triggert írunk, amelyik akkor lép működésbe, amikor a fizetésnövekedés 20 százalék fölé emelkedik:

```
CREATE OR REPLACE TRIGGER fiz_trig
AFTER UPDATE OF fizetes ON dolg
FOR EACH ROW
WHEN (new.fizetes > 1.2 * old.fizetes)
CALL fiz_napl(:new.dolgsorsz, :old.fiz, :new.fiz);
```

Az alábbi UPDATE utasítás a `dolg` táblában minden sort frissít. Azoknál a soroknál, amelyek eleget tesznek a trigger WHEN feltételének, a trigger működésbe lép, és a Java-metódus beszúr egy sort a `fiz_kov` táblába.

```
SQL> UPDATE dolg SET fizetes = fizetes + 30000;

SQL> SELECT * FROM fiz_kov;


 DOLGSORSZ    REGIFIZ      UJFIZ
---------- ---------- ----------
      7369      80000     110000
      7521     125000     155000
      7654     125000     155000
      7876     110000     140000
      7900      95000     125000
      7934     130000     160000

6 rows selected.
```

2.9. példa

Létrehozunk egy olyan triggert, amelyik a következő adatbázistáblába szúr be sorokat:

```
CREATE VIEW dolgozok AS
  SELECT dolgsorsz, dolgnev, 'Értékesítő' AS reszlnev FROM ertekesito
  UNION ALL
  SELECT dolgsorsz, dolgnev, 'Marketing' AS reszlnev FROM mktg;
```

ahol az `ertekesito` és `mktg` adatbázistáblák a következőképpen vannak megadva:

```
CREATE TABLE ertekesito (dolgsorsz NUMBER(4), dolgnev VARCHAR2(10));
CREATE TABLE mktg (dolgsorsz NUMBER(4), dolgnev VARCHAR2(10));
```

Egy INSTEAD OF triggert kell írnunk, mert abba a nézetbe, amely halmazműveleteket használ, nem lehet sorokat beszúrni. A triggerünk a sorokat az alaptáblába szúrja be.

Először (az előbb definiált példában szereplő) `ABTrigger` osztályba írjuk bele a következő Java-metódust:

```
public static void ujDolg (
  int dolgSz, String dolgNev, String reszlNev)
  throws SQLException {
  Connection conn =
    DriverManager.getConnection("jdbc:default:connection:");
  String tablaNev =
   (reszlNev.equals("Értékesítő") ? "ertekesito" : "mktg");
  String sql = "INSERT INTO " + tablaNev + " VALUES (?, ?)";
  try {
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, dolgSz);
    pstmt.setString(2, dolgNev);
    pstmt.executeUpdate();
    pstmt.close();
  } catch (SQLException e) {System.err.println(e.getMessage());}
}
```

Az `ujDolg` metódus egy sort szúr be az `ertekesito` vagy az `mktg` táblákba, a `reszlNev` paraméter értékétől függően. Ehhez a metódushoz a következő hívási specifikációt kell megírnunk:

```
CREATE OR REPLACE PROCEDURE uj_dolg (
  dolg_sz NUMBER, dolg_nev VARCHAR2, reszl_nev VARCHAR2)
AS LANGUAGE JAVA
NAME 'ABTrigger.ujDolg(int, java.lang.String, java.lang.String)';
```

Aztán létrehozzuk az INSTEAD OF triggert:

```
CREATE OR REPLACE TRIGGER dolgozok_trig
INSTEAD OF INSERT ON dolgozok
FOR EACH ROW
CALL uj_dolg(:new.dolgsorsz, :new.dolgnev, :new.reszlnev);
```

Ha a következő INSERT utasításokat futtatjuk, akkor a trigger működésbe lép és a Java-metódus a megfelelő alaptáblába beszúr egy-egy sort:

```
SQL> INSERT INTO dolgozok VALUES (8001, 'Ács', 'Értékesítő');
SQL> INSERT INTO dolgozok VALUES (8002, 'Bóta', 'Értékesítő');
SQL> INSERT INTO dolgozok VALUES (8003, 'Sike', 'Értékesítő');
SQL> INSERT INTO dolgozok VALUES (8004, 'Fülöp', 'Marketing');
SQL> INSERT INTO dolgozok VALUES (8005, 'Hugli', 'Marketing');
SQL> INSERT INTO dolgozok VALUES (8006, 'Rácz', 'Marketing');

SQL> SELECT * FROM ertekesito;

    DOLGSZ DOLGNEV
---------- ----------
      8001 Ács
      8002 Bóta
      8003 Sike
```

```
SQL> SELECT * FROM mktg;

    DOLGSZ DOLGNEV
---------- ----------
      8004 Fülöp
      8005 Hugli
      8006 Rácz

SQL> SELECT * FROM dolgozok;

    DOLGSZ DOLGNEV    RESZLNEV
---------- ---------- ----------
      8001 Ács        Értékesítő
      8002 Bóta       Értékesítő
      8003 Sike       Értékesítő
      8004 Fülöp      Marketing
      8005 Hugli      Marketing
      8006 Rácz       Marketing
```

Java tárolt alprogram meghívása DML-utasításban

Ha egy Java-metódust függvényként publikáltunk, akkor azt az INSERT, DELETE, UPDATE, MERGE, SELECT utasításokban meghívhatjuk.

2.10. példa

Tegyük fel, hogy a következő futtatható Java-osztály tárolva van az adatbázis-kezelő rendszerben:

```
public class Formazo {
  public static String formazDolg (String dolgNev, String beosztas) {
    dolgNev = dolgNev.substring(0,1).toUpperCase() +
      dolgozoNev.substring(1).toLowerCase();
    beosztas = beosztas.toLowerCase();
    if (beosztas.equals("elemző"))
      return (new String(dolgNev + " gyakorlott elemző"));
    else
      return (new String(dolgNev + " kezdő " + beosztas));
  }
}
```

A `Formazo` osztálynak a `formazDolg` nevű metódusa egy nevet és beosztást tartalmazó sztringgel tér vissza. Először írjuk meg a metódus hívási specifikációját:

```
CREATE OR REPLACE FUNCTION formaz_dolg (dolgnev VARCHAR2,
    munkakor VARCHAR2) RETURN VARCHAR2
AS LANGUAGE JAVA
NAME 'Formazo.formazDolg (java.lang.String, java.lang.String)
  return java.lang.String';
```

Azután hívjuk meg a `formaz_dolg` függvényt az alkalmazottak listájának kialakításához:

```
SQL> SELECT formaz_dolg(dolgnev, munkakor) AS "Dolgozók" FROM dolg
  2    WHERE munkakor NOT IN ('FONOK', 'ELNOK') ORDER BY dolgnev;

Dolgozók
-----------------------------------------
Farkas kezdő eladó
Fekete gyakorlott elemző
Kelemen kezdő eladó
Kiss kezdő eladó
Pusztai kezdő eladó
Sós kezdő értékesítő
Szilágyi kezdő értékesítő
Uzelman gyakorlott elemző
Vámos kezdő értékesítő
Varga kezdő értékesítő
```

Java-metódusokat akkor hívhatunk meg DML-utasításban, ha a metódus nem kérdezi le és nem módosítja azt a táblát, amelyet a DML-utasítás éppen manipulál, továbbá nem tartalmaz tranzakció-, munkamenet- és rendszervezérlő, valamint DDL-utasításokat.

Java tárolt alprogram hívása PL/SQL-ből

Java tárolt alprogramot minden PL/SQL-blokkból, alprogramból és csomagból meghívhatunk.

2.11. példa

Tegyük fel, hogy a következő futtatható Java-osztály tárolva van az adatbázis-kezelő rendszerben:

```
import java.sql.*;
import oracle.jdbc.*;

public class Novel {
  public static void fizEmeles (int dolgSz, float szazl)
  throws SQLException {
    Connection conn =
     DriverManager.getConnection("jdbc:default:connection:");
    String sql =
      "UPDATE dolg SET fizetes = fizetes * ? WHERE dolgsorsz = ?";
    try {
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setFloat(1, (1 + szazl / 100));
      pstmt.setInt(2, dolgSz);
      pstmt.executeUpdate();
```

```
        pstmt.close();
      } catch (SQLException e) {System.err.println(e.getMessage());}
    }
  }
```

A `Novel` osztálynak a `fizEmeles` metódusa egy megadott százalékkal növeli az alkalmazottak fizetését. A `fizEmeles` egy `void` metódus, ezért eljárásként tesszük publikussá:

```
CREATE OR REPLACE PROCEDURE fiz_emeles(dolgsorsz NUMBER, szl NUMBER)
AS LANGUAGE JAVA
NAME 'Novel.fizEmeles(int, float)';
```

A következő példában a `fiz_emeles` eljárást egy névtelen PL/SQL-blokkból hívjuk meg:

```
DECLARE
   dolg_az  NUMBER;
   szazl NUMBER;
BEGIN
   -- Értékadás a dolg_az és szazl változóknak
   fiz_emel(dolg_az, szazl);
   ...
END;
```

A következő példában a `sor_szam` függvényt hívjuk meg egy PL/SQL tárolt alprogramból:

```
CREATE PROCEDURE bonusz_szamol (dolg_az NUMBER, bonusz OUT NUMBER) AS
  dolg_szamol NUMBER;
  ...
BEGIN
  dolg_szamol := sor_szamol('dolg');
  ...
END;
```

Az utolsó példában a `dolgozo` objektumtípus metódusát hívjuk meg egy névtelen PL/SQL-blokkból:

```
DECLARE
  dolg_az NUMBER(4);
  v dolg_tipus;
BEGIN
  -- Értékadás a dolg_az változónak
  SELECT VALUE(e) INTO v FROM dolgozok e WHERE dolgsorsz = dolg_az;
  v.fiz_emel(500);
  UPDATE dolgozok e SET e = v WHERE dolgsorsz = dolg_az;
  ...
END;
```

2.2.9. PL/SQL tárolt alprogram hívása Javából

A Javából a JDBC és az SQLJ segítségével tudjuk meghívni a PL/SQL tárolt alprogramjait.

2.12. példa

Tekintsük a következő PL/SQL tárolt függvényt, amely egy adott bankszámla egyenlegével tér vissza:

```
FUNCTION egyenleg (szamla_az NUMBER) RETURN NUMBER IS
  szamla_egyenl NUMBER;
BEGIN
  SELECT egyenl INTO szamla_egyenl FROM szamlak
    WHERE szamla_sz = szamla_az;
  RETURN szamla_egyenl;
END;
```

Ha egy JDC-n keresztül hívjuk meg az `egyenleg` függvényt, akkor a hívás így nézhet ki:

```
CallableStatement cstmt = conn.prepareCall("{? = CALL egyenleg(?)}");
cstmt.registerOutParameter(1, Types.FLOAT);
cstmt.setInt(2, szamlaSz);
cstmt.executeUpdate();
float szamlaEgyenl = cstmt.getFloat(1);
```

SQLJ esetén viszont a hívás alakja a következő:

```
#sql szamlaEgyenl = {VALUES(egyenleg(:IN szamlaSz))};
```

2.2.10. Kivételkezelés tárolt alprogramok esetén

Amikor egy Java tárolt alprogram futtat egy SQL-utasítást és kivétel váltódik ki, a kivétel a `java.sql.SQLException` egy alosztályának példányaként jön létre. Az osztály a `getErrorcode()` és a `getMessage()` metódusai visszaadják az Oracle-hibakódot és -hibaüzenetet.

Ha SQL-ből vagy PL/SQL-ből meghívott Java tárolt alprogramban egy ott nem kezelt kivétel következik be, akkor a hívó a következő hibaüzenetet kapja:

```
ORA-29532 Java call terminated by uncaught Java exception
```

3. FEJEZET

XML az Oracle9*i*-ben

Ebben a fejezetben az Oracle9*i* első kiadásának XML-támogatását mutatjuk be. Terjedelmi korlátok miatt nem törekszünk teljes leírásra, csupán az általunk legfontosabbnak ítélt eszközökre szorítkozunk. A fejezet megértéséhez az Olvasónak rendelkeznie kell bizonyos előképzettséggel az XML nyelv és a hozzá kapcsolódó szabványok, szabványajánlások területén. A pontos megértéshez ismerni kell az XML és HTML nyelveket, az XML-dokumentumok átalakítására szolgáló XSLT (kiterjesztett stíluslap-transzformációs) nyelvet, az XML-dokumentumokban történő navigációra szolgáló XPath nyelvet, valamint tisztában kell lennie a szerver oldali alkalmazások készítésének alapfogalmaival is. A példák megértéséhez ezenfelül szükséges a PL/SQL nyelv ismerete is. Ezek az alapismeretek az [19], [4] és [9] művekben, illetve az interneten a World Wide Web Consortium (W3C) honlapjain (http://www.w3.org, http://w3c.sztaki.hu) találhatók meg.

A fejezet első részében az XML-dokumentumok Oracle9*i* adatbázisban történő tárolására és lekérdezésére szolgáló eszközöket ismertetjük. Bemutatjuk, hogyan lehet egy adatbázisban tárolt XML-dokumentum bizonyos részeit elérni és kinyerni az ott elhelyezkedő adatokat. Példákkal illusztrálva megmutatjuk, hogy hogyan lehet egy SQL-lekérdezés eredményét XML formátumúvá alakítani, és hogy egy XML-dokumentum hogyan szolgálhat egy relációs táblára vonatkozó DML-művelet alapjául.

A második részben a webes tartalomszolgáltatás kerül előtérbe, és bemutatjuk, hogyan publikálhatunk Oracle-eszközökkel könnyen és gyorsan az adatbázisban tárolt adatok alapján felépülő XML-dokumentumokat a weben.

XML

A továbbiakban egészen röviden összefoglaljuk az XML nyelv legfontosabb fogalmait.

Az XML nyelv (eXtensible Markup Language, kiterjesztett jelölőnyelv – l. [9]) az elektronikus dokumentumok strukturált leírásának igényéből fejlődött ki. Első specifikációja 1998 elején jelent meg.

Egy XML-dokumentum jelölőelemekből (markup) épül fel, amelyek a dokumentum egy azonosítható részét határolják. Egy XML-dokumentumnak fizikai és logikai szerkezete is van. A *fizikai felépítés* alapegységei az *egyedek*, amelyek (a dokumentum egyed kivételével) névvel ellátott információmorzsák. Az egyedek segítségével a

dokumentum egyes részeit külön állományokban helyezhetjük el, így egy egyedet akár több XML-dokumentumban is felhasználhatunk, amely így az újrafelhasználhatóság egyik megjelenési formája. A *logikai felépítés* alapegységei az *elemek*, amelyek a dokumentum struktúráját hivatottak leírni. Egy XML-dokumentum szerkezete legjobban egy fával jellemezhető: a dokumentumfa gyökerében helyezkedik el az ún. dokumentumelem. Az általa tartalmazott elemek a fában leszármazottként jelennek meg.

Minden XML-dokumentum logikai szerkezetének meg kell felelnie bizonyos szabályoknak. Egy ilyen szabály a *jólformáltság* kritériuma. Röviden azt mondhatjuk, hogy egy XML-dokumentumot akkor nevezünk jólformáltnak, ha létezik pontosan egy olyan eleme (a gyökérelem), amely nem része egyetlen más elemnek sem, és az elemek között nincs átlapolás, azaz minden elem pontosan az őt tartalmazó elembe van beágyazva (kivéve természetesen a gyökérelemet).

Egy másik fontos fogalom az *érvényesség*. Minden XML-dokumentumhoz tartozhat egy dokumentumséma, amely a dokumentum logikai szerkezetét (azaz magát a dokumentumfát) modellezi. Megadja, hogy mely elemek milyen más elemeket, milyen sorrendben tartalmazhatnak. A dokumentumséma tulajdonképpen az XML-dokumentum „nyelvtana", így ez teszi lehetővé, hogy az XML alapú kommunikáció résztvevői „egy nyelvet beszéljenek". A dokumentumséma leírására több nyelv is létezik, a legismertebb az XML-specifikációban megadott dokumentumtípus-definíciós nyelv (DTD), azonban az utóbbi néhány évben – mivel a DTD kifejezőereje elmarad az elvárttól – újabb és újabb, a dokumentumsémák leírására szolgáló nyelvek jöttek létre, ilyen például a W3C XM Schema nyelve. Egy XML-dokumentumot akkor nevezünk érvényesnek, ha megfelel a hozzá tartozó dokumentumsémában leírt szabályoknak.

3.1. XML-dokumentumok adatbázisban tárolása és lekérdezése

Ebben a részben az XML formátumú adatok Oracle9*i* adatbázis-kezelő rendszerben történő tárolására, és onnan történő lekérdezésére koncentrálunk. Megismerjük, hogyan használható ezekre az XML SQL Utility (XSU), a SYS.XMLType típus, és az adatbázis URI-hivatkozás. Megtudjuk, hogy az Oracle Text (*inter*Media Text) hogyan használható az XML-adatok keresésének és lekérdezésének finomhangolására és gyorsítására.

Az Oracle9*i* az XML-dokumentumok kezelésére egy új, rendszer szintű objektumtípust biztosít. A típus neve SYS.XMLType, és olyan beépített tagfüggvényeket tartalmaz, amelyek erőteljes mechanizmust biztosítanak XML-adatok létrehozására, kinyerésére és indexelésére. A SYS_XMLGEN és a SYS_XMLAGG SQL-függvények, valamint a DBMS_XMLGEN PL/SQL-csomag segítségével lehetőségünk van XML-dokumentumok XMLType objektumpéldányként történő generálására.

Az Oracle9*i* által natívan támogatott XML-jellemzőket a 3.1. táblázat foglalja össze.

3.1. táblázat. *Az Oracle9i által natívan támogatott XML-jellemzők*

XML-eszközök	*Leírás*
XMLType	Az `XMLType` egy előredefiniált tagfüggvényekkel rendelkező, az XML-adatokhoz történő hozzáférést biztosító, rendszer szintű adattípus. Az `XMLType` típus a következő feladatok elvégzésére alkalmas: • Létrehozhatunk `XMLType` típusú oszlopokat, és a típus példányain használhatók az `XMLType` tagfüggvényei. • `XMLType` típusú paraméterekkel és visszatérési értékkel rendelkező PL/SQL-eljárásokat, illetve függvényeket készíthetünk. • Lehetőségünk van `XMLType` típusú oszlopokban tárolt XML-adatok tárolására, indexelésére, kezelésére.
DBMS_XMLGEN	A DBMS_XMLGEN PL/SQL-csomag segítségével SQL-lekérdezések eredményeit kanonikus XML formátumba alakíthatjuk, azt XMLType-ként vagy CLOB-ként visszaadva. A DBMS_XMLGEN csomag C nyelven van megvalósítva. A DBMS_XMLGEN csomag funkcionalitásában hasonlít a DBMS_XMLQuery csomagra.
SYS_XMLGEN	A `SYS_XMLGEN` SQL-függvény az SQL-lekérdezéseken belül generál XML-t. A `DBMS_XMLGEN` csomag lekérdezés szinten operál, aggregált eredményeket ad a lekérdezés *egész* eredményhalmazára vonatkozóan. Ezzel szemben a `SYS_XMLGEN` egy olyan *soronkénti* függvény, amely egy SQL-lekérdezésen belül *egyetlen* argumentumon operál, és az eredményt XML-be konvertálja. A `SYS_XMLGEN` függvény bemenő paramétere XML-dokumentummá alakítandó skalár érték, illetve egy objektumtípus vagy XMLType típus példánya lehet. Megadható egy opcionális második – `XMLGenFormatType` típusú – paraméter is, amely a dokumentum formázására vonatkozó információkat tartalmazza. A `SYS_XMLGEN` függvény visszatérési értékének típusa `XMLType`.
SYS_XMLAGG	A `SYS_XMLAGG` függvény egy `XMLType`-ok halmazát aggregáló oszlopfüggvény. A `SYS_XMLAGG` a bemenő paraméterként kapott XML-dokumentumok/-dokumentumtöredékek összefűzésével és egy csúcs szintű tag hozzáadásával egyetlen XML-dokumentumot hoz létre.
UriType típuscsalád	Az UriType típuscsalád Uri-ref értékek adatbázisban tárolására és lekérdezésére szolgálnak. A SYS.UriType egy olyan absztrakt objektumtípus, amely az URL által mutatott adatokhoz történő hozzáférést biztosító függvényeket tartalmaz. Az UriType rendelkezik két altípussal: a SYS.HttpUriType HTTP URL-ek, míg a SYS.DBUriType adatbázison belüli hivatkozások tárolására képes. A különböző URL-protokollok kezelésére *saját (SYS.UriType) altípusokat hozhatunk létre.* UriFactory csomag: ez a csomag az URL-prefix (http://, ftp:// stb.) alapján képes a megfelelő UriType példány létrehozására. Az UriFactory saját altípusok regisztrációjára is alkalmas, ekkor meg kell adni a prefixet. Például egy gopher-protokollt kezelő altípus regisztrálható az UriFactory-val, csak meg kell adni, hogy a „gopher://" prefixszel rendelkező URL-eket az adott altípus segítségével kell kezelni. Az UriFactory ezek után az ezzel a prefixszel kezdődő URL-ek esetén a regisztrált altípus egy példányát fogja létrehozni. Az UriType típuscsaláddal bővebben [19] foglalkozik.

3.1.1. Az XMLType adattípus

Az XMLType egy új, táblák és nézetek oszlopaiban használható adattípus. XMLType típus lehet változónak, PL/SQL tárolt alprogramok paraméterének, függvény visszatérési értékének típusa is. Az XMLType típus szerver oldalon, PL/SQL-ben, SQL-ben és Javában használható, OCI-n keresztül azonban jelenleg nem.

Az XMLType típus hasznos, XML-tartalmat kezelő beépített tagfüggvényekkel rendelkezik. Például az `extract` függvény kinyer egy bizonyos csomópontot egy XMLType példányból.

Az olyan új, XML-dokumentumokat visszaadó SQL-függvények, mint amilyen például a SYS_XMLGEN, a dokumentumokat XMLType példányként adják vissza. Ez szigorú típusosság megvalósítását teszi lehetővé az SQL-utasításokban. Az XMLType ezáltal ugyanúgy használható SQL-lekérdezésekben, mint a rendszer bármely más, felhasználó által definiált adattípusa.

Az XMLType táblaoszlopok létrehozására használható. A típus rendelkezik egy `createXML` nevű statikus függvénnyel, amely a beszúrandó XMLType példányok létrehozását végzi. Az XML-dokumentumok XMLType-ként tárolása lehetőséget biztosít arra, hogy szabványos SQL-lekérdezések segítségével könnyen kereshessünk az XML-tartalomban.

XMLType típusú értéket tartalmazó táblákon a következőkben felsorolt műveletek végezhetők

1. Tábla létrehozása

Egy XMLType oszlop egy táblában ugyanúgy hozható létre, mint akármilyen más típusú oszlop. Például:

```
CREATE TABLE raktar (
  raktar_azon NUMBER(4),
  raktar_spec SYS.XMLTYPE,
  raktar_nev  VARCHAR2(35),
  hely_azon   NUMBER(4));
```

2. Beszúrás

Ahhoz, hogy az XMLType típusú oszlopba értéket tudjunk beszúrni, létre kell hoznunk egy XMLType példányt. Ezt akár egy VARCHAR-ból, akár pedig egy CLOB-ból az XMLType típus `createXML` nevű statikus függvényével tehetjük meg:

```
INSERT INTO raktar (raktar_azon, raktar_spec)
       VALUES (1001, SYS.XMLType.createXML(
                    '<Raktar raktarSzam="100">
                       <Epulet>Saját</Epulet>
                     </Raktar>'));
```

Ebben a példában az XMLType példányt egy sztringkonstansból hoztuk létre. A `createXML` bemenő paramétere azonban tetszőleges VARCHAR2 vagy CLOB típusú kifejezés lehet.

A `createXML` függvény leellenőrzi, hogy a paramétereként kapott XML-adat megfelel-e a jólformáltsági követelményeknek, azonban nem vizsgálja a dokumentumot az érvényesség szempontjából.

3. Lekérdezés

Az alábbi egyszerű SELECT utasítás bemutatja, hogy hogyan használható az XMLType egy lekérdezésben:

```
SELECT
  r.raktar_spec.extract('/Raktar/Epulet/text()').getStringVal() "Épület"
FROM raktar r;
```

ahol `raktar_spec` egy XMLType típusú oszlop, amelynek meghívjuk az `extract` tagfüggvényét. Ennek az egyszerű lekérdezésnek az eredménye egy sztring (VARCHAR2):

```
Épület
----------------
Saját
```

4. Felülírás

Ebben a kiadásban egy XMLType típussal reprezentált XML-dokumentum tárolása pakolt CLOB-ként történik. Ebből következően, a felülírás során a dokumentumot a saját helyén kell kicserélni. Ebben a kiadásban az XML-adat részenként történő frissítése nem támogatott, így csak a teljes dokumentum cseréjére van mód.

Az XML-dokumentum frissítése egy szabványos UPDATE utasítás segítségével történik, csak létre kell hoznunk egy XMLType példányt:

```
UPDATE raktar SET raktar_spec =
          SYS.XMLType.createXML(
                  '<Raktar raktarSzam="200">
                     <Epulet>Bérelt</Epulet>
                   </Raktar>');
```

Ebben a példában egy sztringkonstansból létrehozunk egy XMLType példányt, és az új értékkel frissítjük a `raktar_spec` oszlopot. Az UPDATE utasításra előírt összes trigger végrehajtásra kerül, így a triggerből elérhetjük és módosíthatjuk az XML-értéket.

5. Törlés

Egy XMLType típusú oszlopot tartalmazó sor törlése semmiben sem különbözik az egyéb sorok törlésétől.

A törlendő sorok azonosítására az `extract` és az `existsNode` függvényeket használhatjuk. Például az összes „Bérelt" sort a következő utasítással törölhetjük:

```
DELETE FROM raktar r
   WHERE r.raktar_spec.extract('//Epulet/text()').getStringVal()
         = 'Bérelt';
```

Az XMLType előnyei

Az XMLType használata egyrészt „összehozza" az XML és az SQL világát azáltal, hogy elősegíti mind az SQL-műveletek XML-tartalmakon, mind pedig az XML-műveletek SQL-tartalmakon történő végrehajtását, másrészt pedig megkönnyíti az implementációt, mivel beépített függvényeket, indexelési és navigációs támogatást tartalmaz.

Az XMLType kombináltan használható az SQL-utasításokban az egyéb oszlopokkal és adattípusokkal. Például, egy XMLType oszlop lekérdezését követően a kinyerés eredményét összekapcsolhatjuk egy relációs oszloppal, és az Oracle meg tudja határozni ezen kérdések végrehajtásának optimális módját.

Az XMLType úgy van optimalizálva, hogy az XML-adat csak akkor jelenjen meg faformában, amikor az szükséges. Így amikor egy XMLType példányon végrehajtunk egy SQL-lekérdezést, a dokumentum szerializált formában van jelen, és csak akkor kerül át faformába, amikor olyan műveleteket hajtunk végre rajta, amihez erre feltétlenül szükség van (ilyen például az `extract` és az `existsNode`). Így egyformán hozzáférhetünk a dokumentum fa- és szerializált reprezentációjához. Az XMLType belső szerkezetét egy optimalizált, DOM-hoz hasonló faszerkezet reprezentálja.

Az XMLType használatának másik nagy előnye, hogy az XMLType típusú oszlopok a gyorsabb keresések érdekében indexelhetők. Ehhez az Oracle9*i* szöveges indexelési technikáját fejlesztették tovább úgy, hogy az támogassa az XMLType típusú oszlopokat is. Az `existsNode` és `extract` függvényekre függvény alapú indexet hozhatunk létre a lekérdezések kiértékelésének meggyorsítására.

Mikor használjunk XMLType-ot?

- Ha XML-dokumentumokat egészben kell az adatbázisban tárolni és kinyerni/lekérdezni.
- Ha szükség van a dokumentum egészének vagy részének az SQL-lekérdezhetőségére. Az `existsNode` és az `extract` függvények biztosítják számunkra az XML-dokumentumok szükséges mértékű SQL-lekérdezhetőségét.
- Ha szigorú típusosság megvalósítására van szükségünk az SQL-utasításokban és PL/SQL-alprogramokban. A szigorú típusosság itt annak biztosítását

jelenti, hogy az átadott értékek nem tetszőleges szöveges adatok, hanem XML-értékek.

- Ha az XML-dokumentum feldolgozásához szükségünk van az `extract` és az `existsNode` függvények által biztosított XPath funkcionalitásra. Megjegyzendő, hogy az XMLType a beépített, C-ben írt XML-elemzőt és -feldolgozót használja, így jobb teljesítmény- és skálázhatósági mutatókkal rendelkezik, amennyiben szerver oldalon használjuk.
- Ha a dokumentumok XPath-kereséseit kell indexelnünk. Az XMLType a keresések optimalizálására használható függvény alapú indexek létrehozására alkalmas tagfüggvényeket tartalmaz.
- Ha nincs szükség a dokumentum részenként történő frissítésére.
- Ha szét kell választani az alkalmazásokat a tárolási modellektől. A későbbi verziókban/kiadásokban az XMLType különféle tárolási alternatívákat fog kínálni. Ha CLOB-ok vagy relációs tárolás helyett XMLType-ot használunk, akkor lehetővé tesszük alkalmazásaink számára, hogy később anélkül térjünk át egy másfajta, kényelmesebb tárolási módra, hogy ez az alkalmazásban bármilyen lekérdezést, vagy DML-utasítást befolyásolna.
- Ha előre felkészülünk a későbbi optimalizációra. Minden XML-lel kapcsolatos új funkcionalitás csak és kizárólag az XMLType-ot fogja támogatni. Mivel a szervert natív módon felkészítik arra, hogy az XMLType csak XML-adatot tartalmazhat, a jövőben jobb optimalizációs és indexelési technikák hajthatók végre rajta. XMLType-ot használó alkalmazások írásával ezek a fejlesztések a jövőben könnyedén elvégezhetők anélkül, hogy az alkalmazást újra kellene írni.

Hogyan használjuk az XMLType-ot?

XML-adatok XMLType típusú oszlopban történő tárolásakor a következő alapelveket kell figyelembe venni:

- *XMLType oszlop definiálása.* Elsőként meg kell adnunk egy XMLType típusú oszlopot. Az oszlopdefinícióhoz bizonyos opcionális tárolási jellemzőket is megadhatunk.
- *XMLType példány létrehozása.* Az XMLType konstruktora segítségével hozhatjuk létre az XML-példányt, a beszúrás előtt. A `SYS_XMLGEN` és a `SYS_XMLAGG` függvények segítségével közvetlenül is létrehozhatunk XMLType példányokat.
- *Egy bizonyos XMLType példány lekérdezése, a tartalom kinyerése.* Az XMLType olyan tagfüggvényeket biztosít, mint például az `extract` és az `existsNode`, amelyek segítségével egy bizonyos csomópont tartalmát kinyerhetjük, illetve ellenőrizhetjük, hogy egy adott csomópont létezik-e.
- *Oracle Text index definiálása.* Az XMLType oszlopokon lehetőségünk van Oracle Text (*inter*Media Text) indexek megadására. Ez lehetőséget biztosít számunkra, hogy a CONTAINS, HASPATH, INPATH, és egyéb szöveges operátorokat használhassuk az adott oszlopon. Minden, a LOB-okon operáló szöveges operátor és indexfüggvény az XMLType típusú oszlopokon is működik.

Egy XMLType típusú érték tárolására ebben a kiadásban egyetlen lehetséges mód van: *csak CLOB-ként tárolható.* A későbbi kiadásokban az Oracle egyéb más tárolási lehetőségeket is biztosít, például BLOB, NCLOB stb.

Egy XMLType típusú oszlop létrehozásakor automatikusan létrejön egy rejtett CLOB-oszlop, amelyben az XML-adat tárolásra kerül. Maga az XMLType oszlop egy virtuális oszlop ezen a rejtett CLOB-oszlopon. Nincs lehetőség a CLOB-oszlop közvetlen elérésére, azonban az oszlop tárolási jellemzőit az XMLType tárolási utasításrészében megadhatjuk.

Nem hozható létre XMLType-ot tartalmazó dinamikus tömb (VARRAY típusú kollekció). Ennek oka az, hogy az Oracle nem engedi meg LOB-lokátorok használatát dinamikus tömbben, jelenleg pedig az XMLType típusú értékek tárolása csak CLOB-ban történhet.

Példák

3.1. példa

XMLType típusú oszlop létrehozása

Ahogyan azt korábban már említettük, XML típusú oszlopot az XMLType típus segítségével hozhatunk létre. A következő utasítás létrehoz egy XMLType típusú rendelésdokumentum oszlopot:

```
CREATE TABLE rendeles_xml (
  rendeles_azon NUMBER,
  rendelesDok   SYS.XMLTYPE);
```

3.2. példa

XMLType típusú oszlop hozzáadása

Lehetőség van táblákhoz XMLType típusú oszlopot hozzáadni. Ez ugyanúgy történik, mint bármely más adattípus esetén. A következő utasítás egy új, vásárlói dokumentumot leíró oszlopot ad a `rendeles_xml` táblához:

```
ALTER TABLE rendeles_xml ADD (vasarloDok SYS.XMLType);
```

3.3. példa

XMLType típusú oszlop törlése

Táblákból XMLType típusú oszlopokat is törölhetünk. A következő utasítás törli az imént hozzáadott `vasarloDok` oszlopot:

```
ALTER TABLE rendeles_xml DROP (vasarloDok);
```

Mint ahogyan azt már korábban is említettük, az XMLType típusú oszlopban tárolt XML-adatok rejtett CLOB-oszlopban vannak tárolva. A CLOB-oszlophoz tárolási jellemzőket is megadhatunk, az alábbi szintaktikával:

```
XMLType [COLUMN] oszlopnév STORE AS CLOB {LOB_szegmensnév
[LOB_paraméterek] | LOB_paraméterek}
```

A 3.2. táblázatból kiderül, hogy melyik szintaktikai elem mit jelent.

3.2. táblázat. *Az XMLType típus tárolási utasításrészében található elemek leírása*

Szintaktikai elem	*Leírás*
oszlopnév	Megadja azt a LOB-oszlopnevet, vagy LOB-objektumattribútumot, amelyre explicit módon a táblára vonatkozó táblaterülettől és tárolási jellemzőtől különböző táblaterületet és tárolási jellemzőt akarunk megadni. Az adatbázis-kezelő rendszer automatikusan létrehoz egy, a rendszer által kezelt indexet minden egyes létrehozott *oszlopnév*re.
LOB_szegmensnév	A LOB-adatszegmens neve adható meg.
LOB_paraméterek	A *LOB_paraméterek* a LOB-tárolás különböző elemeinek megadását teszi lehetővé. • ENABLE STORAGE IN ROW: Ha ez a paraméter meg van adva, akkor amennyiben a LOB-érték mérete kb. 4000 bájtnál (mínusz rendszerinformációk) kisebb, a LOB-érték a sorban (inline módon) kerül tárolásra. Ez az alapértelmezés. Indexszervezett táblák esetén ez a paraméter csak akkor adható meg, ha az *index_org_table_clause* utasításrészben megadunk egy OVERFLOW szegmenst. • DISABLE STORAGE IN ROW: Amennyiben ezzel a paraméterrel letiltjuk a soron belüli tárolást, a LOB-érték – függetlenül a LOB méretétől – a soron kívül lesz letárolva. A LOB-lokátor mindig a soron belül kerül letárolásra, függetlenül attól, hogy a LOB-érték hol helyezkedik el. A STORAGE IN ROW paraméter értéke csak egyetlen esetben változtatható meg, ha átmozgatjuk a táblát. • CHUNK *egész*: Megadja a LOB kezeléséhez lefoglalandó bájtok számát. Ha *egész* nem többszöröse az adatbázis blokkméretének, az Oracle a legközelebbi többszörösre kerekíti fel (bájtokban értve). Ha például az adatbázis egy blokkjának mérete 2048 bájt, az *egész* pedig 2050, akkor az Oracle 4096 bájtot (2 blokkot) foglal le. A maximális *egész* érték 32 768 bájt (32 kilobájt) lehet. Az alapértelmezett CHUNK méret egy adatbázisblokk méretével egyezik meg. A CHUNK érték beállítása után nem változtatható meg. A CHUNK érték nem lehet nagyobb a NEXT értéknél. Ha ez mégis megtörténne, hibaüzenetet kapunk. • PCTVERSION *egész*: Megadja, hogy a LOB újabb változatainak létrehozásához a teljes LOB-tárolóterületnek maximum hány százalékát szabad felhasználni. Az alapértelmezett érték 10, ami azt jelenti, hogy a régebbi LOB-adatok egészen addig nem lesznek felülírva, amíg a teljes LOB-tárolóterületnek a 10%-a használatba nem kerül.

A tábla létrehozásakor a következőképpen adhatjuk meg az oszlopra vonatkozó tárolási jellemzőket:

```
CREATE TABLE rendeles_xml (
     rendeles_azon NUMBER(10),
     rendelesDok   SYS.XMLTYPE
     )
     XMLType COLUMN rendelesDok
        STORE AS CLOB (
            TABLESPACE lob_seg_ts
            STORAGE (INITIAL 4096 NEXT 4096)
            CHUNK 4096 NOCACHE LOGGING
         );
```

Tárolási utasításrészt megadhatunk egy oszlop táblához történő hozzáadásakor is. Ha táblánkat egy új, XMLType típusú oszloppal szeretnénk bővíteni, és megadni annak tárolási utasításrészét, akkor a következőképpen kell eljárnunk:

```
ALTER TABLE rendeles_xml ADD (
     vasarloDok SYS.XMLTYPE
   )
   XMLType COLUMN vasarloDok
      STORE AS CLOB (
           TABLESPACE lob_seg_ts
           STORAGE (INITIAL 4096 NEXT 4096)
           CHUNK 4096 NOCACHE LOGGING
         );
```

Egy XMLType típusú oszlopra is előírhatjuk a NOT NULL megszorítást:

```
CREATE TABLE rendeles_xml (
  rendeles_azon NUMBER(10),
  rendelesDok   SYS.XMLType NOT NULL
);
```

Ez nem engedi meg az alábbi beszúrást:

```
INSERT INTO rendeles_xml (rendelesDok) VALUES (NULL);
```

Ugyanúgy, mint a többi adattípus esetében, a NOT NULL információ megváltoztatását az ALTER TABLE utasítás segítségével érhetjük el:

```
ALTER TABLE rendeles_xml MODIFY (rendelesDok NULL);
ALTER TABLE rendeles_xml MODIFY (rendelesDok NOT NULL);
```

Alapértelmezett értékek és egyéb ellenőrző megszorítások nem alkalmazhatók ezen adattípus esetén.

XMLType függvények

Az Oracle9*i* két új, az XMLType értékeken végrehajtható új SQL-függvényt vezetett be, az `existsNode`-ot és az `extract`-ot.

Az `existsNode` SQL-függvény az XMLType típus `existsNode` tagfüggvénye segítségével van implementálva. Az `existsNode` SQL-függvény szintaktikája:

```
existsNode( XMLType_peldany IN SYS.XMLType,
            XPath_sztring   IN VARCHAR2) RETURN NUMBER;
```

Az `extract` függvény alkalmaz egy XPath-kifejezést, és az eredményül kapott XML-dokumentumtöredéket tartalmazó XMLType értéket adja vissza. A szintaktika a következő:

```
extract( XMLType_peldany IN SYS.XMLType,
         XPath_sztring   IN VARCHAR2) RETURN SYS.XMLType;
```

Az Oracle9*i* első kiadásában az `existsNode` és az `extract` SQL-függvények csak a funkcionális megvalósítást tartalmazzák, az újabb kiadásokban azonban ezek a függvények új indexeket fognak használni, és tovább lesznek optimalizálva.

A 3.3. táblázat az XMLType-hoz kapcsolódó SQL- és tagfüggvények szintaktikáját és leírását foglalja össze.

Az `existsNode` és `extract` tagfüggvények helyett tetszőleges SQL-utasításban használhatók az azonos nevű SQL-függvények. Minden XMLType függvény a beépített, C nyelven írt feldolgozót használja az XML-adatok elemzéséhez, ellenőrzéséhez, valamint az XPath-kifejezések ezeken történő alkalmazásához. A feldolgozás egy optimalizált, memóriában tárolt DOM-fa alapján történik.

3.3. táblázat. *Az XMLType típus statikus és tagfüggvényei*

XMLType függvény	*Szintaxis*	*Leírás*
`createXML`	`STATIC FUNCTION createXML (xmlval IN VARCHAR2) RETURN SYS.XMLType DETERMINISTIC`	Egy XMLType példány sztringből történő létrehozására szolgáló statikus függvény. Ellenőrzi, hogy az XML-adat jólformált-e. PARAMÉTEREK: `xmlval` (IN) – az XML-dokumentumot tartalmazó sztring, amelynek egy jólformált XML-dokumentumot kell tartalmaznia. VISSZATÉRÉSI ÉRTÉK: Az `xmlval` értéket reprezentáló XMLType példány.
`createXML`	`STATIC FUNCTION createXML (xmlval IN CLOB) RETURN SYS.XMLType DETERMINISTIC`	Egy XMLType példány CLOB-ból történő létrehozására szolgáló statikus függvény. Ellenőrzi, hogy az XML-adat jólformált-e. PARAMÉTEREK: `xmlval` (IN) – az XML-dokumentumot tartalmazó CLOB, amelynek egy jólformált XML-dokumentumot kell tartalmaznia. VISSZATÉRÉSI ÉRTÉK: Az `xmlval` értéket reprezentáló XMLType példány.

3.3. táblázat. *Az XMLType típus statikus és tagfüggvényei (folytatás)*

XMLType függvény	*Szintaxis*	*Leírás*
existsNode	MEMBER FUNCTION existsNode (xpath IN VARCHAR2) RETURN NUMBER DETERMINISTIC	Egy adott XPath-kifejezésre megadja, hogy a kifejezést a dokumentumra alkalmazva az érvényes csomóponttal tér-e vissza vagy sem. PARAMÉTEREK: xpath (IN) – a tesztelendő XPath-kifejezés. VISSZATÉRÉSI ÉRTÉK: 0, ha az XPath-kifejezés nem ad vissza egyetlen csomópontot sem, különben 1. Amennyiben az XPath-sztring NULL, vagy a dokumentum üres, a visszatérési érték 0.
extract	MEMBER FUNCTION extract(xpath IN VARCHAR2) RETURN SYS.XMLType DETERMINISTIC	Egy adott XPath-kifejezést alkalmaz a dokumentumra, és az így kapott dokumentumtöredéket XMLType-ként visszaadja. PARAMÉTEREK: xpath (IN) – az alkalmazandó XPath-kifejezés. VISSZATÉRÉSI ÉRTÉK: Az eredményül kapott csomóponto(ka)t tartalmazó XMLType példány. Amennyiben az XPath-kifejezés nem eredményez csomópontot, a visszatérési érték NULL.
isFragment	MEMBER FUNCTION isFragment() RETURN NUMBER	Megvizsgálja, hogy a példány által reprezentált XML-dokumentum töredék-e. Dokumentumtöredékhez akkor juthatunk, ha egy XML-dokumentumon egy extract vagy egyéb olyan műveletet hajtunk végre, amely több csomópontot eredményez. VISSZATÉRÉSI ÉRTÉK: 1, ha az XMLType példány egy dokumentumtöredéket, vagy egy jólformált dokumentumot tartalmaz, különben 0.
getClobVal	MEMBER FUNCTION getClobVal() RETURN CLOB DETERMINISTIC	Visszaadja a dokumentumot CLOB-ként. VISSZATÉRÉSI ÉRTÉK: A szerializált XML-reprezentációt tartalmazó CLOB, amelyet használat után fel kell szabadítani.
getStringVal	MEMBER FUNCTION getStringVal() RETURN VARCHAR2 DETERMINISTIC	Visszaadja a dokumentumot sztringként. VISSZATÉRÉSI ÉRTÉK: A szerializált XML-reprezentációt tartalmazó sztring, vagy, szöveges (text) csomópontok esetében, maga a szöveg. Ha az XML-dokumentum mérete nagyobb a VARCHAR2 típus maximális méreténél (4000 bájt), egy futási idejű hiba következik be.
getNumberVal	MEMBER FUNCTION getNumberVal() RETURN NUMBER DETERMINISTIC	Megadja az XMLType által mutatott numerikus értéket mint számot. Az XMLType-nak egy numerikus értéket tartalmazó létező szöveges csomópontot kell tartalmaznia. VISSZATÉRÉSI ÉRTÉK: Az XMLType példány által meghatározott szövegből létrehozott numerikus érték.

Mivel a `createXML` függvények egyetlen célja egy XMLType példány létrehozása, ezért ezeket a függvényeket az XMLType típus *konstruktorainak* is szokás nevezni.

XMLType-ként tárolt XML-adatok kezelése

Mivel az XMLType egy olyan felhasználói adattípus, amelyhez függvénydefiníciók is kapcsolódnak, az eredményt egyszerűen a függvények meghívásával kaphatjuk meg. XMLType-ot bárhol használhatunk, ahol felhasználó által definiált adattípus használata megengedett, azaz például táblák oszlopaiban, nézetekben, triggerek törzsében, típusdefiníciókban stb.

Egy XMLType típusú oszlopban tárolt XML-adaton a következő adatmanipulációs (DML) utasításokat hajthatjuk végre:

- XML-adat beszúrása,
- XML-adat megváltoztatása,
- XML-adat törlése.

XMLType típusú oszlopba többféleképpen is beszúrhatunk:

- az INSERT utasítás segítségével (SQL-ben, PL/SQL-ben, C-ben (OCI) és Javában),
- az SQL*Loader használatával.

Egy XMLType típusú oszlop csak jólformált XML-dokumentumok tárolására alkalmas. Dokumentumtöredékekkel, illetve nem jólformált XML-adatokkal nem végezhetünk bővítést.

Ahhoz, hogy INSERT utasítás segítségével el tudjuk végezni a beszúrást, először létre kell hozni a beszúrandó XML-dokumentumo(ka)t. Ezt vagy egy XMLType konstruktor (`SYS.XMLType.createXML`) használatával, vagy pedig a `SYS_XMLGEN` és a `SYS_XMLAGG` SQL-függvények segítségével tehetjük meg, SQL-ben, PL/SQL-ben, C-ben (OCI-n keresztül), és Javában egyaránt.

Példák

3.4. példa

A `createXML` használata CLOB típusú paraméterrel

A példa az INSERT...SELECT utasítást, valamint a `createXML` statikus függvényt használja. Először létrehozunk egy XML-dokumentumot CLOB-ként, majd azt egy XMLType típusú értéknek adjuk értékül.

Tartalmazza a `rendeles_clob` tábla az XML-dokumentumot tároló CLOB-ot:

```
CREATE TABLE rendeles_clob
(
  rendeles_azon NUMBER,
  rendelesClob  CLOB
);

-- Beszúrunk egy XML dokumentumot a rendeles_clob táblába.
INSERT INTO rendeles_clob
     VALUES(100, '<?xml version="1.0"?>
                    <RENDELES rend_szam="100">
                       <RNEV>Rend_1</RNEV>
                       <VASARLONEV>János</VASARLONEV>
                       <SZALLITASI_CIM>
                         <UTCA>Fő utca 13.</UTCA>
                         <VAROS>Napfalva</VAROS>
                         <ORSZAG>HU</ORSZAG>
                       </SZALLITASI_CIM>
                    </RENDELES>');
```

Ezek után egy rendelést leíró XML-dokumentumot könnyedén be tudunk szúrni a `rendeles_xml` táblába úgy, hogy a `rendeles_clob` táblában tárolt CLOB-adatból előállítunk egy XMLType példányt:

```
INSERT INTO rendeles_xml
       SELECT rendeles_azon, SYS.XMLType.createXML(rendelesClob)
       FROM rendeles_clob;
```

Megjegyzendő, hogy a CLOB-értéket egy tetszőleges kifejezésből is vehettük volna, beleértve a temporális CLOB-ok létrehozására képes függvényeket is, vagy a CLOB-ot akár egy másik táblából vagy nézetből is kiválaszthattuk volna.

3.5. példa

A `createXML` használata sztring paraméterrel

Ez a példa egy rendelést helyez el a `rendeles_xml` táblában:

```
INSERT INTO rendeles_xml
   VALUES(100, SYS.XMLType.createXML('<?xml version="1.0"?>
                    <RENDELES rend_szam="100">
                       <RNEV>Rend_1</RNEV>
                       <VASARLONEV>János</VASARLONEV>
                       <SZALLITASI_CIM>
                         <UTCA>Fő utca 13.</UTCA>
                         <VAROS>Napfalva</VAROS>
                         <ORSZAG>HU</ORSZAG>
                       </SZALLITASI_CIM>
                    </RENDELES>'));
```

Itt az XMLType példány a `createXML` függvénynek paraméterként átadott sztringkonstans alapján jön létre.

3.6. példa

Beszúrás SYS_XMLGEN segítségével

Az alábbi példa egy olyan rendelést helyez el a `rendeles_xml` táblában, amelyet a fejezet későbbi részében bemutatatandó SYS_XMLGEN SQL-függvény segítségével hozunk létre. Tételezzük fel, hogy `rendeles_on` egy rendelésobjektumot tartalmazó objektumnézet. A `rendeles_on` nézet teljes definíciója a 3.17. példában található.

```
INSERT INTO rendeles_xml
  SELECT SYS_XMLGEN(value(r),
                SYS.XMLGenFormatType.createFormat('RENDELES'))
  FROM rendeles_on r
  WHERE r.rendeles_szam=2001;
```

A SYS_XMLGEN függvény létrehoz egy XMLType értéket a rendelésobjektumból, amelyet aztán beszúrunk a `rendeles_xml` táblába.

Csak a teljes XML-dokumentum cseréjére van lehetőség, ezt azonban meg lehet tenni SQL-ből, PL/SQL-ből, C-ből (OCI-n keresztül) és Javából egyaránt.

3.7. példa

Módosítás `createXML` segítségével

Ez a példa a `createXML` statikus metódus segítségével módosítja az XMLType típusú oszlop értékét. Csak azokat a dokumentumokat módosítjuk, amelyek rendelési száma 100.

```
UPDATE rendeles_xml rx SET rx.rendelesDok = SYS.XMLType.createXML(
           '<?xml version="1.0"?>
            <RENDELES rend_szam="200">
              <RNEV>Rend_2</RNEV>
              <VASARLONEV>Béla</VASARLONEV>
              <SZALLITASI_CIM>
                <UTCA>Mellék utca 2.</UTCA>
                <VAROS>Holdváros</VAROS>
                <ORSZAG>HU</ORSZAG>
              </SZALLITASI_CIM>
            </RENDELES>')
WHERE rx.rendelesDok.extract('/RENDELES/@rend_szam').getNumberVal() =
100;
```

A módosítások jelenleg csak a teljes dokumentum szintjén valósíthatók meg, így ha csupán egy bizonyos dokumentum egy részét szeretnénk módosítani, akkor is le kell cserélnünk a teljes dokumentumot.

Egy XMLType típusú oszlopból ugyanúgy törlünk, mint bármilyen más adattípus esetén.

3.8. példa

Sorok törlése `extract` segítségével

Ha az összes olyan sort el szeretnénk távolítani a táblából, ahol a rendelés neve „Rend_2", akkor a következőképpen kell eljárnunk:

```
DELETE FROM rendeles_xml rx WHERE
rx.rendelesDok.extract('/RENDELES/RNEV/text()').getStringVal()='Rend_2';
```

XMLType oszlopokból a következő három módon nyerhetünk ki XML-adatot:

- az XMLType oszlop SQL, PL/SQL, C (OCI) vagy Java segítségével történő lekérdezésével;
- az XMLType oszlop közvetlen lekérdezésével, majd az `extract` vagy `existsNode` használatával vagy
- az XML-tartalom szöveges operátorokkal történő lekérdezésével.

XMLType típusú adatot PL/SQL-en vagy Javán keresztül kérdezhetünk le. Az XML-ben tárolt információt CLOB-ként, VARCHAR2-ként, illetve számként (NUMBER típus) rendre a `getClobVal`, a `getStringVal`, illetve a `getNumberVal` függvények segítségével kaphatjuk meg.

3.9. példa

Az XML-dokumentum kinyerése CLOB-ként

A példa bemutatja, hogyan kérdezzünk le egy XMLType típusú oszlopot SQL*Plusban:

```
SET LONG 2000

SELECT rx.rendelesDok.getClobVal() AS rendelesXML
  FROM rendeles_xml rx;

RENDELESXML
--------------------------------
<?xml version="1.0"?>
<RENDELES rend_szam="2">
   <RNEV>Rend_2</RNEV>
   <VASARLONEV>Béla</VASARLONEV>
   <SZALLITASI_CIM>
      <UTCA>Mellék utca 2.</UTCA>
      <VAROS>Holdváros</VAROS>
      <ORSZAG>HU</ORSZAG>
   </SZALLITASI_CIM>
</RENDELES>
```

A lekérdezés előtt kiadott SET LONG SQL*Plus-utasításra azért van szükség, hogy a teljes eredmény megjelenjen a képernyőn.

XMLType típusú adatok lekérdezése, és bizonyos részeinek kinyerése az `existsNode` és az `extract` függvények segítségével lehetséges. A dokumentumban történő navigálásra mindkét függvény a W3C XPath-szabványának (l. [9]) egy részhalmazát használja.

Az XPath az XML-dokumentumok navigálására biztosított W3C szabvány. Az XPath az XML-dokumentumot csomópontokból felépülő faként tekinti. Széles választékát biztosítja a fában történő „sétálásra", illetve a feltételek és csomóponttesztelő függvények megadására szolgáló műveleteknek. Egy XPath-kifejezés XML-dokumentumra történő alkalmazása csomópontok egy halmazát eredményezi. Például, a `/RENDELES/RNEV` XPath-kifejezés kiválasztja az összes olyan `RNEV` elemet a dokumentumban, amely a `RENDELES` gyökérelem közvetlen leszármazottja.

Most röviden áttekintjük az XPath jelölésrendszerének fontosabb elemeit:

A `/` jel a fa gyökerét jelképezi egy XPath-kifejezésben, például a `/RENDELES` a dokumentum logikai gyökerének azt a gyermekét jelenti, amelynek neve `RENDELES`.

A `/` jelnek ezenfelül elválasztó szerepe is van: egy adott csomópont gyermekeinek azonosításához elérésiútvonal-elválasztóként funkcionál, például a `/RENDELES/RNEV` kifejezés azon `RNEV` (rendelésnév) elemeket azonosítja/jelöli ki, amelyek a gyökérelem (`RENDELES`) közvetlen leszármazottjai.

A `//` egy adott csomópont összes leszármazottjának azonosítására szolgál, például a `RENDELES//UTCA` kifejezés minden `UTCA` elemet kiválaszt a `RENDELES` elemen belül.

A `*` helyettesítő karakter tetszőleges gyermekcsúcs nevére illeszkedik, például a `/RENDELES/*/UTCA` kifejezés kiválaszt minden, a `RENDELES` elem unokájaként szereplő `UTCA` elemet.

A `[]` állítástesztek jelölésére alkalmas. Az XPath támogat számos bináris operátort (OR, AND, NOT stb.) is. Például a `/RENDELES[@rend_szam=20 AND RNEV="Rend_2"]/SZALLITASI_CIM` kifejezés azon rendelések szállítási cím elemét választja ki, amelyek rendelési száma 20, a rendelés neve pedig „Rend_2".

A `[]`-t ezen túlmenően tömbindex jelölésre is használjuk. Például egy `/RENDELES/RENDELESSZAM[2]` kifejezés a `RENDELES` gyökérelem alatti második rendelésszám elemet azonosítja (a `RENDELESSZAM` minta által kijelölt listában elfoglalt helyet adja meg).

Az Oracle `extract` és `existsNode` függvényei az XPath-kifejezések halmazának csak egy részét támogatja. Az Oracle9*i* első kiadása által támogatott XPath-szerkezetek a következők:

- közvetlen leszármazott elérése, például `/RENDELES/SZALLITASI_CIM/UTCA`;
- attribútum elérése, például `/RENDELES/@rend_szam`;
- index alapú hozzáférés, például `/RENDELES/RENDELESSZAM[2]`;
- keresés helyettesítő karakterrel, például `/RENDELES/*/UTCA`;
- leszármazottak keresése, például `/RENDELES//UTCA`;
- csomóponttesztelő függvények, például `/RENDELES/RNEV/text()`.

A jelenlegi kiadás csak a leszármazottak felé történő navigációt biztosítja, nem támogatja az egyéb (testvérek, illetve ősök felé történő) navigációt.

Az `extract` és `existsNode` függvények egyelőre nem támogatnak több-bájtos karakterkészleteket.

Végezetül, egy XPath-kifejezésnek kötelező módon elemeknek, szöveges, vagy attribútum csomópontoknak (akár egyelemű) halmazát kell kijelölnie, a kifejezés kiértékelésének eredménye nem lehet logikai érték.

Az XMLType `existsNode` függvénye arról nyújt információt, hogy az adott XPath-kifejezés kiértékelése eredményez-e legalább egy XML-elemet vagy szöveges csomópontot. Ha igen, akkor 1-gyel tér vissza, különben 0-val. Tekintsük például a következő XML-dokumentumot:

```
<RENDELES rend_szam="1">
  <RNEV>Rend_1</RNEV>
  <VASARLONEV>János</VASARLONEV>
  <SZALLITASI_CIM>
    <UTCA>Fő utca 13.</UTCA>
    <VAROS>Napfalva</VAROS>
    <ORSZAG>HU</ORSZAG>
  </SZALLITASI_CIM>
</RENDELES>
```

A `/RENDELES/RNEV` XPath-kifejezés egyetlen csomópontot ad vissza, így az `existsNode` 1-et fog visszaadni. Ugyanez a helyzet a `/RENDELES/RNEV/text()` kifejezés esetén, amely egy egyszerű szöveges csomópontot jelöl ki. A `RENDELES/@rend_szam` kifejezés a `RENDELES` elem `rend_szam` attribútumának értékét eredményezi.

Az olyan XPath-kifejezések esetén, amelyek kiértékelése egyetlen csomópontot sem eredményez (például `/RENDELES/RTIPUS`), az `existsNode` függvény 0-t ad vissza.

Az `existsNode` tagfüggvény egyaránt használható lekérdezésekben (mint ahogyan az a következő néhány példában látható), illetve segítségével függvény alapú indexeket is felépíthetünk, amelyekkel a lekérdezések kiértékelése gyorsítható fel.

Az XMLType `extract` függvénye az adott XPath-kifejezés által azonosított csomópont(ok) tartalmának kinyerésére szolgál. A kinyert csomópontok elemek, attribútumok és szöveges csomópontok lehetnek. A kinyerést követően az összes szöveges csomópont egyetlen szöveges csomóponttá vonódik össze („összeesik").

Lehetséges, hogy egy XPath-kifejezés alapján az `extract` segítségével kinyert információ nem jólformált XML-dokumentum. Ez esetben az eredményül kapott XMLType érték több csomópontot, illetve bizonyos esetekben egyszerű skalár adatot tartalmazhat, amelynek kinyerése az XMLType `getStringVal` és `getNumberVal` módszereivel történhet.

Például a `RENDELES/RNEV` XPath-kifejezés a fenti XML-dokumentum `RNEV` nevű elemét azonosítja. A `RENDELES/RNEV/text()` pedig az `RNEV` elem tartalmát jelentő szöveges csomópontra utal. Ez utóbbi is XMLType típusú érték (azaz az `extract(rendelesDok, '/RENDELES/RNEV/text()')` függvényhívás egy XMLType

példányt ad vissza), még akkor is, ha ez a példány csak szöveget tartalmaz. Az eredményül kapott szöveges csomópontból magát a szövegértéket (VARCHAR2 típusú értékként) a `getStringVal` metódus meghívásával kaphatjuk meg.

Az elemekben leírt szöveges csomópontok értékéhez történő hozzáféréshez a `text()` csomóponttesztelő függvényt kell meghívni. Az így kapott szöveges csomópontot csak ezután, a `getStringVal`, illetve `getNumberVal` függvény segítségével tudjuk SQL-adattá alakítani. Amennyiben a `text()`-et nem hívjuk meg, egy XML-dokumentumtöredéket kapunk. Például a `/RENDELES/RNEV` XPath-kifejezés a `<RENDELES>Rend_1</RENDELES>` dokumentumtöredéket azonosítja, míg a `/RENDELES/RNEV/text()` kifejezés a „Rend_1" szöveges értéket reprezentáló csomópontot jelöli ki.

Ha egy XML-dokumentumban egy elem többször is ismétlődik, akkor az elhelyezkedési tesztek [lásd [9], 150. oldal] segítségével a kifejezés által kijelölt dokumentumrészek listájából indexek segítségével azonosíthatjuk az egyes elemeket. Például a

```
<RENDELES>
  <RENDELESSZAM>100</RENDELESSZAM>
  <RENDELESSZAM>200</RENDELESSZAM>
</RENDELES>
```

XML-dokumentumban a `//RENDELESSZAM[1]` kifejezés az első (a 100 értéket tartalmazó) `RENDELESSZAM` elemet azonosítja, míg a `//RENDELESSZAM[2]` kifejezés a dokumentum második `RENDELESSZAM` elemére utal.

Az `extract` visszatérési értékének típusa minden estben XMLType. Ha az XPath-kifejezés alkalmazása egy üres halmazt eredményez, az `extract` NULL értékkel tér vissza.

Az `extract` tagfüggvény leggyakoribb alkalmazási területei a következők:

- olyan numerikus értékek kinyerése, amelyeken a feldolgozást gyorsító függvény alapú indexek építhetők;
- SQL-utasítások FROM részében használható kollekciókifejezések kinyerése;
- olyan dokumentumtöredékek kinyerése, amelyek később különböző dokumentumok létrehozásának alapjául szolgálhatnak (a kapott dokumentumtöredék egy új dokumentum egy része lesz).

Példák

3.10. példa

Az `extract` és az `existsNode` használata

A példában a korábban létrehozott `rendeles_xml` táblát használjuk, amely két oszlopból áll: egy rendelésazonosítót és egy, a rendelést leíró XML-dokumentumot tartalmaz. Feltesszük, hogy a következő két sort beszúrtuk a táblába:

```
INSERT INTO rendeles_xml VALUES (100,
   SYS.XMLType.createXML('<?xml version="1.0"?>
                          <RENDELES rend_szam="221">
                            <RNEV>Első rendelés</RNEV>
                          </RENDELES>'));

INSERT INTO rendeles_xml VALUES (200,
   SYS.XMLType.createXML('<?xml version="1.0"?>
                          <RENDELES>
                            <RNEV>Második rendelés</RNEV>
                          </RENDELES>'));
```

A rendelések neveit sztringként az `extract` függvény segítségével kaphatjuk meg:

```
SELECT rx.rendelesDok.extract('//RNEV/text()').getStringVal() AS
"Rendelésnév" FROM rendeles_xml rx
WHERE rx.rendelesDok.existsNode('/RENDELES/RNEV') = 1;
```

Itt az `extract` függvény az `RNEV` elem tartalmát nyeri ki. Az `existsNode` azon `RNEV` csomópontokat keresi ki, amelyek a dokumentum gyökéreleményének, a `RENDELES` elemnek gyermekei.

Megjegyezzük, hogy a `text()` függvény a szöveges csomópontot adja vissza, a `getStringVal` függvény pedig csupán a csomópontban található szöveges érték lekérésére szolgál.

3.11. példa

Adatok kinyerése XML-dokumentumból

Ebből a példából kiderül, hogy egy rendelést leíró XML-dokumentumból hogyan nyerhetjük ki az adatokat, és hogyan helyezhetjük el azokat egy relációs táblába. Hozzuk létre a következő két relációs táblát:

```
CREATE TABLE vasarlo
(
  vasarlo_azon NUMBER PRIMARY KEY,
  vasarlo_nev  VARCHAR2(20)
);

INSERT INTO vasarlo VALUES (1001, 'Kovács János');

CREATE TABLE rendeles
(
  rendeles_szam NUMBER,
  rnev          VARCHAR2(100),
  vasarlo_azon  NUMBER REFERENCES vasarlo,
  szc_utca      VARCHAR2(100), -- a szállítási cím utcája
  szc_varos     VARCHAR2(30),  -- a szállítási cím városa
  szc_irszam    VARCHAR2(20)   -- a szállítási cím irányítószáma
);
```

Az alábbi XML-dokumentumban tárolt adatokat egy egyszerű PL/SQL-blokk segítségével szúrhatjuk be a relációs táblába.

```
<?xml version="1.0"?>
<RENDELES rend_szam="2001">
  <RNEV>Kovács rendelése</RNEV>
  <VASARLO vasarlonev="Kovács János"/>
  <SZALLITASI_CIM>
    <UTCA>Akác fasor 22.</UTCA>
    <VAROS>Debrecen</VAROS>
    <ORSZAG>HU</ORSZAG>
  </SZALLITASI_CIM>
</RENDELES>
```

Feltételezve, hogy a fenti XML-dokumentum jelen van a `rendeles_xml` táblában, az SQL-lekérdezés a következő:

```
INSERT INTO rendeles
SELECT rx.rendelesDok.extract('/RENDELES/@rend_szam').getNumberVal(),
rx.rendelesDok.extract('/RENDELES/RNEV/text()').getStringVal(),
       -- A vásárló nevéhez tartozó vásárlóazonosító lekérdezése
       ( SELECT vasarlo_azon
         FROM   vasarlo v
         WHERE  v.vasarlo_nev =
rx.rendelesDok.extract('/RENDELES/VASARLO/@vasarlonev').getStringVal()
        ),
rx.rendelesDok.extract('/RENDELES/SZALLITASI_CIM/UTCA/
text()').getStringVal(),
       rx.rendelesDok.extract('//VAROS/text()').getStringVal(),
       rx.rendelesDok.extract('//IRANYITOSZAM/text()').getStringVal()
FROM rendeles_xml rx;
```

A `rendeles` tábla most a következő értékeket tartalmazza:

```
RENDELES_SZAM RNEV                VASARLO_AZON SZC_UTCA       SZC_VAROS SZC_IRSZAM
------------- ------------------ ------------ -------------- --------- ----------
         2001 Kovács rendelése           1001 Akác fasor 22. Debrecen
```

A `rendeles` tábla `SZC_IRSZAM` attribútumának értéke NULL, mert az XML-dokumentum nem tartalmazott `IRANYITOSZAM` nevű elemet. Megjegyezzük még, hogy a `//VAROS` kifejezés tetszőleges mélységben keresi a várost leíró `VAROS` elemet.

Ugyanezt a beszúrást egy PL/SQL-blokkban is megtehetjük:

```
DECLARE
  rendelesxml   SYS.XMLType;
  vasarlonev    VARCHAR2(200);
  rendelesszam NUMBER;
  rnev          VARCHAR2(100);
  szc_utca      VARCHAR2(100);
  szc_varos     VARCHAR2(30);
```

```
  szc_irszam   VARCHAR2(20);
BEGIN
  SELECT rx.rendelesDok INTO rendelesxml FROM rendeles_xml rx WHERE
    rx.rendelesDok.extract('/RENDELES/@rend_szam').getNumberVal() = 100;
  vasarlonev := rendelesxml.extract
     ('//VASARLO/@vasarlonev').getStringVal();
  rendelesszam := rendelesxml.extract
     ('/RENDELES/@rend_szam').getNumberVal();
  rnev := rendelesxml.extract('/RENDELES/RNEV/text()').getStringVal();
  szc_utca := rendelesxml.extract
     ('/RENDELES/SZALLITASI_CIM/UTCA/text()').getStringVal();
  szc_varos := rendelesxml.extract('//VAROS/text()').getStringVal();
  szc_irszam := rendelesxml.extract
     ('//IRANYITOSZAM/text()').getStringVal();
  INSERT INTO rendeles VALUES (rendelesszam, rnev,
      (SELECT vasarlo_azon FROM vasarlo WHERE vasarlo_nev = vasarlonev),
       szc_utca, szc_varos, szc_irszam);
END;
/
```

3.12. példa

Keresés `extract` segítségével

Az `extract` és `existsNode` függvények segítségével számos különféle műveletet hajthatunk végre az XMLType típusú oszlopokon. Tekintsük a következő lekérdezést:

```
SELECT rx.rendelesDok.extract('/RENDELES/RNEV/text()').getStringVal() rnev
FROM rendeles_xml rx
WHERE rx.rendelesDok.existsNode
   ('/RENDELES/SZALLITASI_CIM[VAROS="Debrecen"]') = 1 AND
      rx.rendelesDok.extract('//@rend_szam').getNumberVal() = 2001 AND
      rx.rendelesDok.extract('//@vasarlonev').getStringVal()
        LIKE '%János%';
```

Ez az SQL-utasítás azon rendelések nevét határozza meg, amelyeket leíró XML-dokumentumok debreceni szállítási címet tartalmaznak, rendelésszámuk 2001, valamint a vásárlót valamilyen Jánosnak hívják.

XMLType típusú adatok szöveges operátorokkal történő lekérdezése

Az Oracle szöveges indexelési mechanizmusa CLOB és VARCHAR típusú oszlopokon működik. Az Oracle9*i* ezt terjeszti ki annak érdekében, hogy az XMLType típusú oszlopokon is alkalmazható legyen. Ha egy XMLType típusú oszlopra szöveges indexet építünk, automatikusan XML-szakaszok jönnek létre, valamint alkalmazhatóvá válik a CONTAINS operátor, amelyet úgy bővítettek ki, hogy támogassa az XPath-t.

A lekérdezések kiértékelésének felgyorsítása céljából a következőkben felsorolt indexeket hozhatjuk létre egy XMLType-on.

XMLType-ra épülő függvény alapú indexek

A lekérdezések egy XML-dokumentum `existsNode` vagy `extract` módszere által meghatározott értékek alapján felépített függvény alapú indexek segítségével gyorsíthatók fel. Egy `extract` műveletre a függvény alapú index felépítése például a következőképpen zajlik. Ahhoz, hogy a

```
SELECT * FROM rendeles_xml rx
WHERE rx.rendelesDok.extract('//@rend_szam').getNumberVal()= 100;
```

lekérdezést fel tudjuk gyorsítani, létre kell hoznunk egy függvény alapú indexet az `extract` függvényre:

```
CREATE INDEX rszam_index ON rendeles_xml
       (rendelesDok.extract('//@rend_szam').getNumberVal());
```

Ezen index segítségével az SQL-lekérdezés végrehajtásakor a WHERE feltétel kiértékelése a függvény alapú index alapján történik meg, nem pedig az XML-dokumentum elemzésével, majd az XPath-kifejezés kiértékelésével.

A műveletek végrehajtását egy bittérkép alapú függvény alapú index segítségével is felgyorsíthatjuk. Erre legalkalmasabb az `existsNode` függvény, mivel ennek visszatérési értéke 1 vagy 0, attól függően, hogy az XPath-kifejezés ráillik-e a dokumentumra, avagy sem.

Például azon lekérdezések gyorsítására, amelyek azt tudakolják, hogy az XML-dokumentum tartalmaz-e szállítási címre vonatkozó információt, és a lekérdezés alakja

```
SELECT * FROM rendeles_xml rx
WHERE rx.rendelesDok.existsNode('//SZALLITASI_CIM') = 1;
```

létrehozhatunk egy bittérkép alapú függvény alapú indexet az `existsNode` függvényre:

```
CREATE INDEX szall_index ON rendeles_xml
       (rendelesDok.existsNode('//SZALLITASI_CIM'));
```

Szöveges indexek létrehozása XMLType típusú oszlopokra

Szöveges indexet ugyanúgy a CREATE INDEX utasítás segítségével és az INDEXTYPE megadásával hozhatunk létre, mint bármilyen CLOB vagy VARCHAR típusú oszlop esetén. Mivel azonban az XMLType virtuális oszlopként van megvalósítva, a szöveges index a függvény alapú indexelő mechanizmust használva épül fel.

A szöveges indexek létrehozásához, és lekérdezésekben történő használatához ez megköveteli, hogy ne csak az indexek és szöveges indexek létrehozásához szükséges jogosultsággal rendelkezzünk, de legyünk birtokában a függvény alapú indexek létrehozásához szükséges jogoknak is. Ez a következőket jelenti:

- A QUERY REWRITE jogosultság szükséges ahhoz, hogy saját sémánkban egy XMLType típusú oszlopra szöveges indexet építsünk. Amennyiben olyan XMLType oszlopokra vagy olyan táblákra szeretnénk szöveges indexet felépíteni, amely nem a mi sémánkban helyezkedik el, a GLOBAL QUERY REWRITE jogosultsággal kell rendelkeznünk.
- A QUERY_REWRITE_ENABLED inicializációs paramétert igazra („true") kell beállítani.
- A QUERY_REWRITE_INTEGRITY paramétert megbízhatóra („trusted") kell állítani ahhoz, hogy a szerver felülírhassa a lekérdezéseket az index felhasználása céljából.

A `rendelés_xml` tábla `rendelesDok` oszlopára a következő utasítással építhetünk szöveges indexet:

```
CREATE INDEX rendeles_szoveges_index ON
             rendeles_xml(rendelesDok) INDEXTYPE IS ctxsys.context;
```

Ezután ezen az oszlopon értelmezettek a CONTAINS és a SCORE műveletek. Az Oracle9*i* CONTAINS függvénye két új művelet, az INPATH és a HASPATH segítségével támogatja az XPath-t.

Az INPATH azt vizsgálja, hogy egy adott szó szerepel-e a megadott útvonalon, míg a HASPATH azt ellenőrzi, hogy az adott XPath út jelen van-e a dokumentumban. Nézzünk egy példát.

```
SELECT * FROM rendeles_xml rx
WHERE CONTAINS(rx.rendelesDok,
                    'haspath(//VASARLO[@vasarlonev="Kovács János"])') > 0;
```

Vannak bizonyos különbségek a CONTAINS-en belül, illetve az `existsNode`-ban és az `extract`-ban támogatott XPath-eszközök között.

- Ebben a kiadásban az Oracle Text index az XPath-szabvány nagyobb részét támogatja, mint a funkcionális megvalósítás: itt bizonyos tesztfeltételeket is megadhatunk.
- Mivel az Oracle Text index nem veszi figyelembe a szóközöket, az olyan XPath-kifejezések, ahol a szóközök jelentőséggel bírnak, esetleg nem megfelően pontos eredményt adnak.
- Az Oracle Text index támogat bizonyos sztringegyenlőség-vizsgálattal kapcsolatos logikai kifejezéseket, de nem támogatja a numerikus értékek és tartományok összehasonlítását.
- Egy másik korlátozás, hogy az Oracle Text index rossz eredményt adhat abban az esetben, ha egy XML-dokumentum csak tag-ek és attribútumok neveiből áll, és nem tartalmaz semmilyen szöveget. Például az

```
<A>
  <B>
    <C>
    </C>
  </B>
  <D>
    <E>
    </E>
  </D>
</A>
```

dokumentum esetén az `A/B/E` XPath-kifejezés hibásan, de illeszkedni fog az XML-dokumentumra.

- Mind a függvény alapú, mind pedig a szöveges index támogatja a navigációt. Így a szöveges indexet mintegy elsődleges szűrőként használhatjuk, amellyel kiválasztjuk azokat a dokumentumokat, amelyek potenciálisan illeszkednek a megadott kritériumra, majd másodlagos szűrőket (például `existsNode` vagy `extract` műveleteket) alkalmazhatunk a fennmaradó dokumentumokra.

XMLType használata triggerekben

A triggerekben a NEW és OLD korrelációs nevek használatával hozzáférhetünk az XMLType típusú oszlopok értékéhez ugyanúgy, mint a többi adattípus esetén. INSERT és UPDATE utasítások esetén a NEW korrelációs név értékének megváltoztatásával megváltoztathatjuk magát a beszúrandó értéket.

3.13. példa

XMLType triggerben

Az alábbi trigger megváltoztatja a rendelést abban az esetben, ha az nem tartalmaz szállítási címet, illetve ha úgy cseréljük le a rendelést leíró XML-dokumentumot, hogy az eredeti és az új rendelési szám nem egyezik meg.

```
CREATE OR REPLACE TRIGGER rendeles_trigger
  BEFORE INSERT OR UPDATE ON rendeles_xml
  FOR EACH ROW
DECLARE
  rendeles_szam NUMBER;
BEGIN
  IF INSERTING THEN
    IF :NEW.rendelesDok.existsNode('//SZALLITASI_CIM') = 0 THEN
       :NEW.rendelesDok := SYS.XMLType.createXML
       ('<RENDELES>ÉRVÉNYTELEN RENDELÉS</RENDELES>');
    END IF;
  END IF;
  -- Módosításkor, ha a régi és az új rendelésDok
  -- különböző rendelésszámmal
  -- rendelkezik, akkor ez érvénytelen rendelés.
```

```
    IF UPDATING THEN
      IF :OLD.rendelesDok.extract('/RENDELES/@rend_szam').getNumberVal()
         !=
         :NEW.rendelesDok.extract('/RENDELES/@rend_szam').getNumberVal()
      THEN
        :NEW.rendelesDok := SYS.XMLType.createXML
        ('<RENDELES>ÉRVÉNYTELEN RENDELÉS</RENDELES>');
      END IF;
    END IF;
  END rendeles_trigger;
  /
```

Ez az egyszerű példa természetesen csak illusztrációs célokat szolgál. Az XMLType típusú értékeket olyan hasznos tevékenységek elvégzésére is használhatjuk a triggeren belül, mint például az üzleti logika feltételeinek, vagy bizonyos olyan feltételeknek az ellenőrzése, amelyeknek az XML-dokumentumnak meg kell felelnie.

3.1.2. A DBMS_XMLGEN csomag

A DBMS_XMLGEN egy tetszőleges SQL-lekérdezésből, a lekérdezés eredményeinek XML-be képzésével tud XML-dokumentumokat létrehozni. Az XML-dokumentumot CLOB-ként kapja meg. Egy ún. fetch" felület segítségével megadhatjuk a maximális, illetve a kihagyandó sorok számát. Ez nagyon hasznos olyan webalkalmazások esetén, amelyek oldalanként egyszerre csak bizonyos számú rekordot jelenítenek meg.

A DBMS_XMLGEN csomag `getXML` módszerének eredménye egy CLOB típusú érték. Az alapértelmezett leképezés a következőképpen zajlik:

- A lekérdezés minden sora egy alapértelmezett, `ROW` névvel ellátott XML-elemre képződik le.
- A lekérdezés teljes eredményét egy `ROWSET` nevű elem tartalmazza. Ezeket az alapértelmezett elemneveket a DBMS_XMLGEN csomag `setRowTagName` és `setRowSetTagName` eljárásaival változtathatjuk meg.
- A lekérdezés eredményének minden oszlopa a `ROW` elem gyermekelemeként jelenik meg.
- A CURSOR-kifejezések kivételével a `DBMS_XMLGEN` csomag minden adattípust támogat. A bináris adatok hexadecimális reprezentációval jelennek meg.

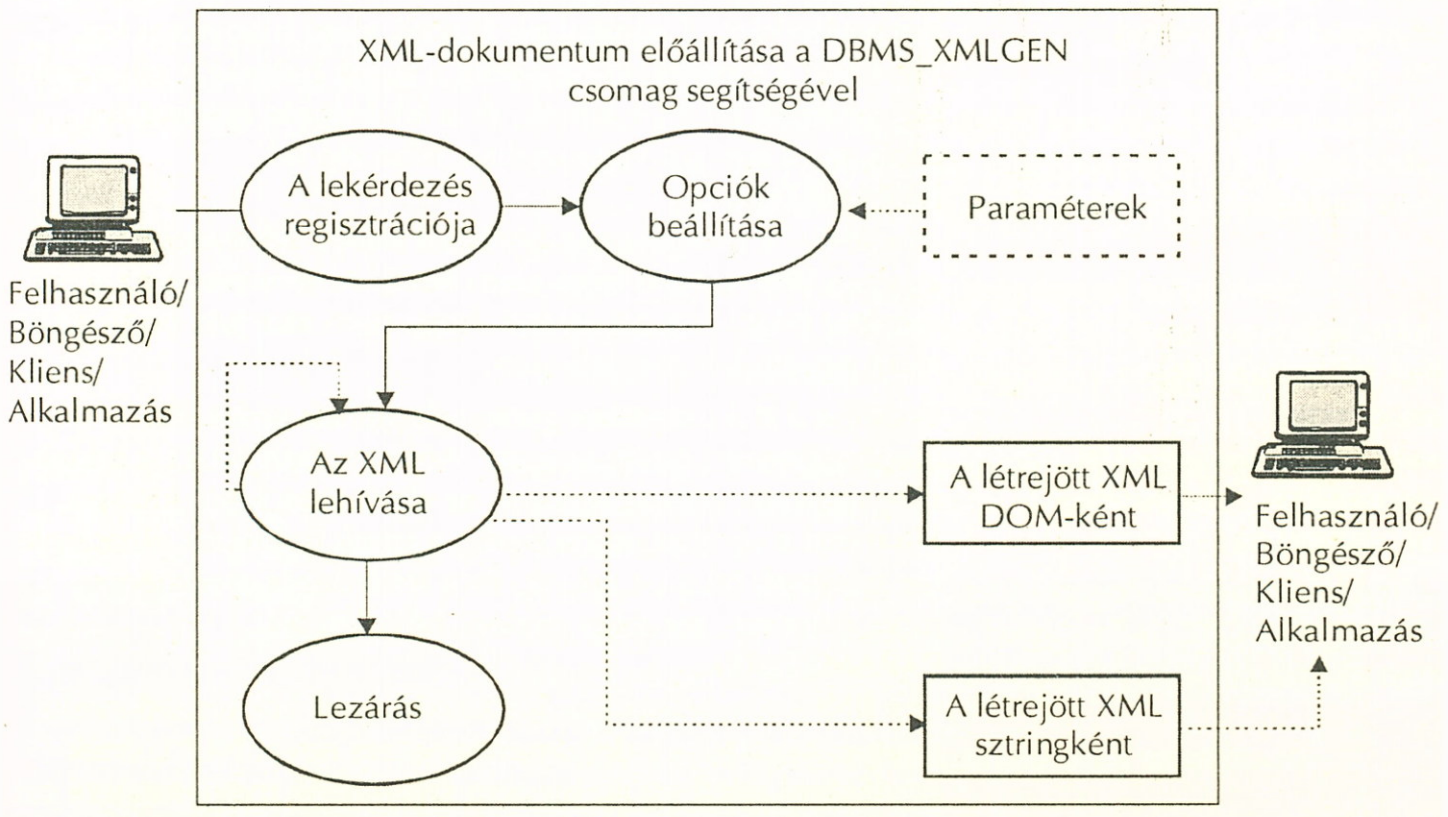
Mivel a dokumentum CLOB-ban van tárolva, karakterkódolása az adatbázis-kezelő rendszer karakterkódolásával egyezik meg, azaz, ha például a szerver karakterkódolása SHIFTJIS, akkor az XML-dokumentum karakterkódolása is SHIFTJIS lesz. A 3.1. ábra a DBMS_XMLGEN hívási sorrendjét mutatja be.

1. A csomag által nyújtott lehetőségek kiaknázáshoz elsőként a `newContext` függvénynek paraméterként átadjuk a lekérdezés szövegét. A függvény a lekérdezés alapján létrehoz egy ún. *lekérdezési környezetet*, amelyet a későbbi műve-

letek majd fel fognak használni. A lekérdezési környezetet egy `ctxHandle` típusú érték reprezentálja, amit a továbbiakban *környezetkezelőnek* nevezünk. A `newContext` függvény ezt a környezetkezelőt adja vissza.

2. Ezt a környezetet a csomag minden alprogramjának eljuttatjuk, hogy különféle opciókat beállíthassanak. Például, egy `ROW` elem nevének beállításához a `setRowTag(ctx)`-et kell meghívnunk, ahol `ctx` az előző `newContext` hívásból kapott környezetkezelő.
3. A `getXML` segítségével megkaphatjuk az XML formátumú eredményt. Az egyszerre leválogatható sorok maximális számát a `setMaxRows` hívással állíthatjuk be. Ezt a függvényt többször is meghívhatjuk, így minden `getXML` hívás előtt beállíthatjuk, hogy az adott hívásban legfeljebb hány sort szeretnénk eredményül kapni. Amennyiben a lekérdezés már nem tartalmaz további sorokat, a függvény NULL-lal tér vissza.
   A `getXML` minden esetben egy XML-dokumentumot ad vissza, még akkor is, ha a lekérdezés nem tartalmaz eredménysort. Azt, hogy hány sort kaptunk eredményül, a `getNumRowsProcessed` függvény segítségével tudhatjuk meg.
4. Szükség esetén újra végrehajthatjuk a lekérdezést, és ismételhetjük a 3. lépésben leírtakat.
5. A `closeContext` meghívásával a lefoglalt (környezetkezelőhöz kapcsolódó) erőforrások felszabadulnak.

A 3.4. táblázat összefoglalja a DBMS_XMLGEN csomagban található típus- és alprogramspecifikációkat.



3.1. ábra. *A DBMS_XMLGEN hívási sorrendje*

3.4. táblázat. *A DBMS_XMLGEN csomag típusai, függvényei és eljárásai*

Típusdefiníció	*Leírás*
`SUBTYPE ctxHandle IS NUMBER;`	A csomag alprogramjai által használt környezetkezelő típusa.
Nevesített konstans	*Leírás*
`NONE CONSTANT NUMBER:= 0;`	A `getXML` függvényben annak megadására szolgál, hogy kell-e DTD-t vagy XML Schema-t generálni, avagy sem. Az Oracle9*i* első kiadásában csak a `NONE` választható.
`DTD CONSTANT NUMBER:= 1;`	
`SCHEMA CONSTANT NUMBER:= 2;`	
Alprogram	*Leírás*
`FUNCTION newContext(queryString IN VARCHAR2) RETURN ctxHandle;`	Egy adott lekérdezésre generál egy új környezetkezelőt, és azt visszaadja. Ez a visszaadott érték használható majd a további függvényhívások esetén. PARAMÉTEREK: `queryString` (IN) – a lekérdezés, amelynek eredményeit XML-be kell alakítani. VISSZATÉRÉSI ÉRTÉK: a környezetkezelő. Ezt a függvényt kell először meghívni ahhoz, hogy hozzájussunk egy kezelőhöz, amelyet aztán a `getXML` és egyéb függvények az eredmény XML-be alakításához használnak.
`PROCEDURE setRowTag(ctx IN ctxHandle, rowTag IN VARCHAR2);`	Beállítja a sorokat elválasztó elem nevét. Az alapértelmezett elemnév: `ROW`. Amennyiben NULL-t adunk meg második paraméterként, a sorokat elválasztó elem hiányozni fog. Ha mind a sorokat, mind pedig a sorok halmazát elhatároló tag nevének NULL-t adunk, és a lekérdezés eredménye több, mint egy sor vagy oszlop, hiba lép fel. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő, `rowTag` (IN) – a sort leíró elem neve. Amennyiben ez a paraméter NULL, az azt jelenti, hogy nem akarjuk, hogy ez az elem egyáltalán megjelenjen.
`PROCEDURE setRowSetTag(ctx IN ctxHandle, rowSetTag IN VARCHAR2);`	Beállítja a dokumentum gyökérelemének nevét. Az alapértelmezett érték: `ROWSET`. Amennyiben NULL-t adunk meg második paraméterként, az elem nem kerül kiírásra. Ha mind a sorokat, mind pedig a sorok halmazát elhatároló tag nevének NULL-t adunk, és a lekérdezés eredménye több, mint egy sor vagy oszlop, hiba lép fel. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő, `rowsetTag` (IN) – a dokumentumelem neve. NULL érték átadásával jelölhetjük, ha nem szeretnénk megjeleníteni ezt az elemet.

3.4. táblázat. *A DBMS_XMLGEN csomag típusai, függvényei és eljárásai (folytatás)*

Alprogram	*Leírás*
`PROCEDURE getXML(ctx IN ctxHandle, clobval IN OUT NOCOPY CLOB, dtdOrSchema IN NUMBER:= NONE);`	A megadott maximális számú sor beolvasásával elkészíti az eredmény XML-dokumentumot, amelyet a paramétereként kapott CLOB-hoz fűz. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő, `clobval` (IN OUT) – az a CLOB-érték, amelyhez az XML-dokumentumot hozzá kell fűzni, `dtdOrSchema` (IN) – megadja, hogy kell-e DTD-t vagy XML Schema-t generálni. Ez a paraméter ebben a kiadásban nem támogatott. Akkor célszerű ezt az eljárást használni, ha egymást követő hívások során ugyanazt a CLOB-ot szeretnénk felhasználni, és el akarjuk kerülni a felesleges CLOB-másolgatást. A `getXML` ezen verziója hatékonyabb, mint a táblázat következő sorában megadott, habár ez esetben nekünk kell létrehoznunk a LOB-lokátort. Az XML-kimenet előállítása során a `setSkipRows` hívás során beállított számú sort átugorva maximum a `setMaxRows` hívással beállított számú sor (illetve ha nincs megadva, a teljes eredményhalmaz) beolvasásra kerül, és ez lesz XML-lé alakítva. A `getNumRowsProcessed` függvénnyel kaphatunk információt arról, hogy mennyi sort sikerült beolvasni.
`FUNCTION getXML(ctx IN ctxHandle, dtdOrSchema IN NUMBER:= NONE) RETURN CLOB;`	Létrehoz egy XML-dokumentumot, amit CLOB-ként visszaad. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő, `dtdOrSchema` (IN) – megadja, hogy kell-e DTD-t vagy XML Schema-t generálni. Ez a paraméter ebben a kiadásban nem támogatott. VISSZATÉRÉSI ÉRTÉK: a dokumentumot tartalmazó temporális CLOB. Ezt a temporális CLOB-ot használat után a `DBMS_LOB.freeTemporary` függvény meghívásával fel kell szabadítani.
`FUNCTION getXMLType(ctx IN ctxHandle, dtdOrSchema IN NUMBER:= NONE) RETURN SYS.XMLType;`	Létrehoz egy XML-dokumentumot, amit XMLType típusú objektumként visszaad. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő, `dtdOrSchema` (IN) – megadja, hogy kell-e DTD-t vagy XML Schema-t generálni. Ez a paraméter ebben a kiadásban nem támogatott. VISSZATÉRÉSI ÉRTÉK: a dokumentumot tartalmazó XMLType példány.

3.4. táblázat. *A DBMS_XMLGEN csomag típusai, függvényei és eljárásai (folytatás)*

Alprogram	*Leírás*
`FUNCTION getNumRowsProcessed (ctx IN ctxHandle) RETURN NUMBER;`	Megadja, hogy a `getXML` híváskor hány sor került feldolgozásra. Az átugrott sorok nem számítanak. PARAMÉTEREK: `ctx` (IN) – a `newContext` hívásból kapott környezetkezelő. VISSZATÉRÉSI ÉRTÉK: a `getXML` utolsó hívásakor feldolgozott sorok száma. Ez a szám nem tartalmazza az átugrott sorokat. Ezt a függvényt kilépési feltételként használhatjuk abban az esetben, ha a `getXML`-t ciklusban hívjuk. Ne feledjük, hogy a `getXML` mindig generál XML-dokumentumot, még akkor is, ha az nem tartalmaz sorokat.
`PROCEDURE setMaxRows(ctx IN ctxHandle, maxRows IN NUMBER);`	Beállítja, hogy maximálisan hány sort akarunk lekérni egy `getXML` hívás során. PARAMÉTEREK: `ctx` (IN) – a végrehajtott lekérdezés környezetének környezetkezelője, `maxRows` (IN) – az egy `getXML` hívás során maximálisan megkapható sorok száma. A `maxRows` paraméter akkor használható, ha lapozható eredményeket szeretnénk előállítani. Például egy XML- vagy HTML-oldal generálásakor megadhatjuk, hogy egyszerre hány sort alakítsunk XML-be, a következő sorokat pedig további hívásokkal kaphatjuk meg. Ez gyorsabb válaszidőt is eredményezhet.
`PROCEDURE setSkipRows(ctx IN ctxHandle, skipRows IN NUMBER);`	A `getXML` minden hívásakor, még az XML-kimenet előállítása előtt átugorja a megadott számú sort. PARAMÉTEREK: `ctx` (IN) – a végrehajtott lekérdezés környezetének környezetkezelője, `skipRows` (IN) – a `getXML` hívások során átugrandó sorok száma. A `skipRows` paramétert akkor érdemes használni, amikor állapot nélküli weblapok számára készítünk lapozható eredményeket. Az első XML- vagy HTML-oldal generálásakor a `skipRows`-t nullára állítjuk, míg később annyira, ahány sort előzőleg lekérdeztünk.
`PROCEDURE setConvertSpecialChars (ctx IN ctxHandle, conv IN boolean);`	Beállítja, hogy az XML-adatokban található speciális karaktereket le kell-e cserélni az őket reprezentáló beépített XML-egyedekre (pl. a < jelet át kell-e alakítani `<`-re) vagy sem. Alapértelmezés szerint a konverzió megtörténik. PARAMÉTEREK: `ctx` (IN) – a használandó környezetkezelő, `conv` (IN) – az igaz érték jelzi a konverzió szükségességét.

3.4. táblázat. *A DBMS_XMLGEN csomag típusai, függvényei és eljárásai (folytatás)*

Alprogram	*Leírás*
	Ezt a függvényt abban az esetben célszerű használni, amikor biztosak vagyunk abban, hogy a bemenő adatok nem tartalmaznak olyan speciális karaktereket, mint pl. a <, >, ", ' stb. Ez gyorsítja az XML-feldolgozást, mivel – különösen nagy adatfolyam esetén – a helyettesítendő karakterek megkeresése igen időigényes is lehet. Ha azonban biztosan tudjuk, hogy a bejövő adat nem tartalmaz ilyen lecserélendő karaktereket, a keresés kiiktatásával növelhetjük a teljesítményt.
`PROCEDURE useItemTagsForColl(ctx IN ctxHandle);`	Beállítja a kollekcióelemeket leíró XML-elem nevét. Az alapértelmezés a kollekciót alkotó elemek típusának neve. Ez az alapértelmezés írható felül ezen eljárás segítségével. PARAMÉTEREK: `ctx` (IN) – a környezetkezelő. Egy NUMBER típusú értékeket tartalmazó kollekciót feltételezve, a kollekcióelemek leírására szolgáló XML-elem alapértelmezett neve NUMBER. Az eljárást meghíva ezen XML-elem neve úgy áll elő, hogy a kollekció típusú oszlop nevéhez hozzáfűzzük az `_ITEM` sztringet.
`PROCEDURE restartQuery(ctx IN ctxHandle);`	Újra végrehajtja a lekérdezést, és az XML-kimenet generálását az első sornál kezdi. PARAMÉTEREK: `ctx` (IN) – a végrehajtott lekérdezésnek megfelelő környezetkezelő. Ezt az eljárást meghíva anélkül hajthatjuk végre újra a lekérdezést, hogy új környezetet kellene létrehoznunk.
`PROCEDURE closeContext(ctx IN ctxHandle);`	Felszabadítja az adott környezethez kapcsolódó összes erőforrást – beleértve az SQL-kurzort, a kötéseket, a puffereket stb. is – és lezárja a környezetet. PARAMÉTEREK: `ctx` (IN) – a lezárandó környezetkezelő. Az eljárás végrehajtását követően a kezelő a továbbiakban már nem használható egyetlen másik DBMS_XMLGEN függvényhívásban sem.

Példák

A fejezet hátralévő részének példái egy vállalat alkalmazottait, részlegeit, és éppen futó projektjeit nyilvántartó adatbázis-sémán alapulnak. A séma létrehozását a következő utasításokkal végeztük:

```
CREATE SEQUENCE reszleg_seq START WITH 10 INCREMENT BY 10;
CREATE SEQUENCE alkalmazott_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE projekt_seq START WITH 1 INCREMENT BY 1;
```

```
CREATE OR REPLACE TYPE nyelvtudas_tip IS VARRAY(10) OF VARCHAR2(10);
/
CREATE TABLE reszleg (
  kod               NUMBER(2)        PRIMARY KEY,
  nev               VARCHAR2(20),
  hely              VARCHAR2(15));

CREATE TABLE alkalmazott (
  azon              NUMBER(4)        PRIMARY KEY,
  nev               VARCHAR2(30),
  szul_datum        DATE,
  beosztas          VARCHAR2(20),
  fonok_azon        NUMBER(4),
  fizetes           NUMBER(9,2),
  reszleg           NUMBER(2)        REFERENCES reszleg,
  nyelvtudas        nyelvtudas_tip);

CREATE TABLE projekt (
  projektkod        NUMBER(4)        PRIMARY KEY,
  projektnev        VARCHAR2(20),
  projekt_hely      VARCHAR2(15));

CREATE TABLE projekten_dolgozik (
  alkalmazott_azon  NUMBER(4)        REFERENCES alkalmazott,
  projekt_azon      NUMBER(4)        REFERENCES projekt,
  oraszam           NUMBER(2),
  PRIMARY KEY (alkalmazott_azon, projekt_azon));
```

A táblákat a következő INSERT utasítások segítségével töltöttük fel értékekkel:

```
INSERT INTO reszleg VALUES (reszleg_seq.NEXTVAL, 'Számlázás',
'Nyíregyháza');
INSERT INTO reszleg VALUES (reszleg_seq.NEXTVAL, 'Kutatás', 'Debrecen');
INSERT INTO reszleg VALUES (reszleg_seq.NEXTVAL, 'Értékesítés',
'Budapest');

INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Szabó Péter',
to_date('1954.11.24.','YYYY.MM.DD.'), 'cégvezető', NULL, 400000, NULL,
nyelvtudas_tip('angol','német','orosz','spanyol'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Kovács János',
to_date('1970.01.04.','YYYY.MM.DD.'), 'osztályvezető', 1, 200000, 30,
nyelvtudas_tip('angol','olasz','német'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Nagy Béla',
to_date('1963.07.11.','YYYY.MM.DD.'), 'előadó', 2, 130000, 30, NULL);
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Gipsz Jakab',
to_date('1960.03.15.','YYYY.MM.DD.'), 'osztályvezető', 1, 230000, 20,
nyelvtudas_tip('angol'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Kiss József',
to_date('1966.12.09.','YYYY.MM.DD.'), 'könyvelő', 1, 175000, 10, NULL);
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Balogh Csaba',
to_date('1972.12.30.','YYYY.MM.DD.'), 'jogász', 1, 160000, 10,
nyelvtudas_tip('latin','német'));
```

```
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Tóth Andrea',
to_date('1974.07.30.','YYYY.MM.DD.'), 'titkárnő', 1, 110000, 10,
nyelvtudas_tip('orosz'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL,
'Horváth Ildikó', to_date('1944.10.03.','YYYY.MM.DD.'), 'takarítónő',
4, 200000, 20, NULL);
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Kis András',
to_date('1975.02.28.','YYYY.MM.DD.'), 'fejlesztő', 4, 190000, 20,
nyelvtudas_tip('angol'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Szabó Sándor',
to_date('1976.04.04.','YYYY.MM.DD.'), 'fejlesztő', 4, 200000, 20,
nyelvtudas_tip('angol','francia'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Nagy Ferenc',
to_date('1974.05.01.','YYYY.MM.DD.'), 'fejlesztő', 4, 210000, 20,
nyelvtudas_tip('angol','német'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL,
'József István', to_date('1965.09.11.','YYYY.MM.DD.'), 'üzletkötő',
2, 165000, 30, nyelvtudas_tip('német'));
INSERT INTO alkalmazott VALUES (alkalmazott_seq.NEXTVAL, 'Komor Ágnes',
to_date('1969.06.30.','YYYY.MM.DD.'), 'üzletkötő', 2, 185000, 30,
nyelvtudas_tip('portugál','finn'));

INSERT INTO projekt VALUES (projekt_seq.NEXTVAL, 'Alkalmazásfejlesztés',
'Debrecen');
INSERT INTO projekt VALUES (projekt_seq.NEXTVAL, 'Termékreklámozás',
'Budapest');
INSERT INTO projekt VALUES (projekt_seq.NEXTVAL, 'Pályázatkészítés',
'Szeged');
INSERT INTO projekt VALUES (projekt_seq.NEXTVAL, 'Év végi zárás',
'Nyíregyháza');

INSERT INTO projekten_dolgozik VALUES (4, 1, 40);
INSERT INTO projekten_dolgozik VALUES (9, 1, 10);
INSERT INTO projekten_dolgozik VALUES (10, 1, 50);
INSERT INTO projekten_dolgozik VALUES (11, 1, 35);
INSERT INTO projekten_dolgozik VALUES (2, 2, 45);
INSERT INTO projekten_dolgozik VALUES (3, 2, 30);
INSERT INTO projekten_dolgozik VALUES (4, 2, 15);
INSERT INTO projekten_dolgozik VALUES (12, 2, 15);
INSERT INTO projekten_dolgozik VALUES (13, 2, 35);
INSERT INTO projekten_dolgozik VALUES (1, 3, 30);
INSERT INTO projekten_dolgozik VALUES (4, 3, 10);
INSERT INTO projekten_dolgozik VALUES (6, 3, 25);
INSERT INTO projekten_dolgozik VALUES (5, 3, 20);
INSERT INTO projekten_dolgozik VALUES (7, 3, 30);
INSERT INTO projekten_dolgozik VALUES (11, 3, 10);
INSERT INTO projekten_dolgozik VALUES (1, 4, 10);
INSERT INTO projekten_dolgozik VALUES (3, 4, 10);
INSERT INTO projekten_dolgozik VALUES (5, 4, 60);
INSERT INTO projekten_dolgozik VALUES (6, 4, 30);
INSERT INTO projekten_dolgozik VALUES (7, 4, 20);
```

3.14. példa

Egyszerű szerkezetű XML-dokumentumok generálása

A példa az `alkalmazott` tábla adatait lekérdezve előállít egy XML-dokumentumot, amelyet CLOB-ként elhelyez egy átmeneti táblában.

```
-- Létrehozunk egy táblát, amelyben az eredmény
-- dokumentumot tároljuk majd
CREATE TABLE temp_clob (eredmeny CLOB);

DECLARE
   kornyezet DBMS_XMLGEN.ctxHandle;
   eredmeny  CLOB;
BEGIN
  kornyezet := DBMS_XMLGEN.newContext('SELECT * FROM alkalmazott');
  -- A sorok információit körülölelő elem nevét
  -- ALKALMAZOTT-ra állítjuk
  DBMS_XMLGEN.setRowTag(kornyezet, 'ALKALMAZOTT');
  -- Lekérjük az eredményt
  eredmeny := DBMS_XMLGEN.getXML(kornyezet);
  INSERT INTO temp_clob VALUES(eredmeny);
END;
/
```

A példa által generált kimenet a következő:

```
SET PAGESIZE 1000
SET LONG 10000

SELECT * FROM temp_clob;
EREDMENY
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
 <ALKALMAZOTT>
  <AZON>1</AZON>
  <NEV>Szabó Péter</NEV>
  <SZUL_DATUM>54-NOV-24</SZUL_DATUM>
  <BEOSZTAS>cégvezető</BEOSZTAS>
  <FIZETES>400000</FIZETES>
  <NYELVTUDAS>
   <VARCHAR2>angol</VARCHAR2>
   <VARCHAR2>német</VARCHAR2>
   <VARCHAR2>orosz</VARCHAR2>
   <VARCHAR2>spanyol</VARCHAR2>
  </NYELVTUDAS>
 </ALKALMAZOTT>
 <ALKALMAZOTT>
  <AZON>2</AZON>
  <NEV>Kovács János</NEV>
  <SZUL_DATUM>70-JAN-04</SZUL_DATUM>
  <BEOSZTAS>osztályvezető</BEOSZTAS>
```

```
    <FONOK_AZON>1</FONOK_AZON>
    <FIZETES>200000</FIZETES>
    <RESZLEG>30</RESZLEG>
    <NYELVTUDAS>
     <VARCHAR2>angol</VARCHAR2>
     <VARCHAR2>olasz</VARCHAR2>
     <VARCHAR2>német</VARCHAR2>
    </NYELVTUDAS>
   </ALKALMAZOTT>
   ...
</ROWSET>
```

3.15. példa

Egyszerű szerkezetű, lapozható XML-dokumentumok generálása

Sok esetben szükségessé válhat, hogy egyszerre csak rögzített számú sort kérdezzünk le. Ez csökkenti a válaszidőt, és kisebb dokumentumokat is eredményez, ami igen hasznos lehet akkor, ha az eredményen valamilyen DOM-műveletet kell végrehajtani, hiszen jóval kisebb lesz a memóriában tárolt DOM-fa mérete. Különösen sok sorból álló válaszokat eredményező kérdések esetén van ennek jelentősége, illetve, ha az eredmények megjelenítését laponként végezzük. Ez utóbbi esetben nem kell feleslegesen lekérnünk az adatbázisszerverről az összes sort, csupán annyit, amennyit egyszerre megjelenítünk. Az eredményhalmaz további elemeit elegendő csupán akkor lekérnünk, amikor a felhasználó lapozni akar.

Jelen példában bemutatjuk, hogy hogyan használható a DBMS_XMLGEN csomag az `alkalmazott` tábla sorainak ötösével történő lekérdezésére.

```
-- Létrehozunk egy táblát, amelyben az eredmény dokumentumot tároljuk majd
CREATE TABLE temp_clob (eredmeny CLOB);

DECLARE
   kornyezet DBMS_XMLGEN.ctxHandle;
   eredmeny CLOB;
BEGIN
  -- A lekérdezés környezetének lekérése.
  kornyezet := DBMS_XMLGEN.newContext('SELECT * FROM alkalmazott');
  -- A sorok maximális számát beállítjuk 5-re.
  DBMS_XMLGEN.setMaxRows(kornyezet, 5);
  LOOP
    -- Lekérjük az eredményt.
    eredmeny := DBMS_XMLGEN.getXML(kornyezet);
    -- Ha már nincsenek további sorok, kilépünk.
    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(kornyezet) = 0;
    -- A LOB adat feldolgozása következik. E példában feldolgozás gyanánt
    -- az eredményeket egy táblában helyezzük el.
    INSERT INTO temp_clob VALUES(eredmeny);
  END LOOP;
END;
/
```

Itt az eredményhalmaz minden öt sorára egy külön XML-dokumentumot kapunk eredményül. Mivel eredetileg az alkalmazott táblába 13 sort szúrtunk be, így összesen 3 dokumentumot kapunk eredményül, ebből kettő a tábla öt-öt sorát, egy pedig a maradék hármat reprezentálja.

3.16. példa

Bonyolultabb szerkezetű XML-dokumentumok generálása felhasználói típusok segítségével

A beágyazott szerkezeteket objektumtípussal reprezentálva összetett XML-dokumentumokat is generálhatunk.

```
CREATE TYPE alkalmazott_tip AS OBJECT (
  "@alkalmazott_azon"      NUMBER(4),
  nev                      VARCHAR2(30)
)
/
CREATE TYPE alkalmazottlista_tip AS TABLE OF alkalmazott_tip
/
CREATE TYPE reszleg_tip AS OBJECT (
  "@reszleg_azon"   NUMBER(2),
  reszlegnev        VARCHAR2(20),
  alkalmazottlista  alkalmazottlista_tip
)
/
DECLARE
   kornyezet DBMS_XMLGEN.ctxHandle;
   eredmeny  CLOB;
BEGIN
  -- A lekérdezés környezetének lekérése.
  kornyezet := DBMS_XMLGEN.newContext
               ('SELECT reszleg_tip(kod, nev,
                   CAST(MULTISET(SELECT a.azon, a.nev
                                 FROM   alkalmazott a
                                 WHERE  a.reszleg = r.kod)
                                AS alkalmazottlista_tip)) AS reszleg_xml
                 FROM reszleg r');
  DBMS_XMLGEN.setRowTag(kornyezet, NULL);
  eredmeny := DBMS_XMLGEN.getXML(kornyezet);
  INSERT INTO temp_clob VALUES (eredmeny);
END;
/
```

A MULTISET operátor az adott részlegen dolgozó alkalmazottak halmazát listaként tekinti, a CAST operátor pedig ezt a listát a megfelelő kollekciótípusúvá alakítja. Ezután létrehozzuk a részleg példányt, amiből a SYS_XMLGEN segítségével előállítjuk az objektumpéldánynak megfelelő XML-dokumentumot. Az eredményül kapott XML-dokumentum a következő:

```
SELECT * FROM temp_clob;

EREDMENY
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
 <RESZLEG_XML reszleg_azon="10">
  <RESZLEGNEV>Számlázás</RESZLEGNEV>
  <ALKALMAZOTTLISTA>
   <ALKALMAZOTT_TIP alkalmazott_azon="5">
    <NEV>Kiss József</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="6">
    <NEV>Balogh Csaba</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="7">
    <NEV>Tóth Andrea</NEV>
   </ALKALMAZOTT_TIP>
  </ALKALMAZOTTLISTA>
 </RESZLEG_XML>
 <RESZLEG_XML reszleg_azon="20">
  <RESZLEGNEV>Kutatás</RESZLEGNEV>
  <ALKALMAZOTTLISTA>
   <ALKALMAZOTT_TIP alkalmazott_azon="4">
    <NEV>Gipsz Jakab</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="8">
    <NEV>Horváth Ildikó</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="9">
    <NEV>Kis András</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="10">
    <NEV>Szabó Sándor</NEV>
   </ALKALMAZOTT_TIP>
   <ALKALMAZOTT_TIP alkalmazott_azon="11">
    <NEV>Nagy Ferenc</NEV>
   </ALKALMAZOTT_TIP>
  </ALKALMAZOTTLISTA>
 </RESZLEG_XML>
 ...
</ROWSET>
```

Relációs adatokat használva az eredmény nem egymásba ágyazott XML-szerkezeteket tartalmazó dokumentum lesz. Ahhoz, hogy egymásba ágyazott XML-szerkezeteket kapjunk, objektumrelációs adatokkal kell dolgoznunk. Az XML-re történő leképezés a következőképpen történik:

- Maguk az objektumtípusok XML-elemekre képződnek le.
- Az objektumtípusok attribútumai az őket tartalmazó objektumtípusnak mint szülőelemnek a leszármazottai lesznek.

Mivel egy objektumtípus attribútumának típusa lehet egy másik objektumtípus, így gyakorlatilag tetszőleges mélységű egymásba ágyazást megvalósíthatunk.

Ilyen összetett dokumentumszerkezetet objektumtípusok, objektumnézetek és objektumtáblák segítségével hozhatunk létre. Az objektumpéldányok XML-re történő leképezése egy kanonikus leképezés segítségével történik.

A @ jellel kezdődő nevű oszlopok és attribútumok a leképezés során az őket tartalmazó XML-elem attribútumai lesznek.

3.17. példa

Egy XML formátumú rendelés létrehozása

Ebben a példában egy relációs adatbázisból objektumnézetek segítségével egy XML formátumú rendelést hozunk létre.

```
CREATE TYPE telszamok_tip AS VARRAY(10) OF VARCHAR2(20)
/

CREATE TYPE cim_tipus AS OBJECT (
  utca          VARCHAR2(200),
  varos         VARCHAR2(200),
  orszag        CHAR(2),
  irszam        VARCHAR2(20)
)
/

CREATE TYPE vasarlo_tip AS OBJECT (
  vasarlo_azon      NUMBER,
  vasarlo_nev       VARCHAR2(200),
  cim               cim_tipus,
  telszam           telszamok_tip
)
/

CREATE TYPE arucikk_tip AS OBJECT (
  "@arucikk_azon"     NUMBER,
  ar                  NUMBER,
  ado                 NUMBER
)
/

CREATE TYPE rendelesi_tetel_tip AS OBJECT (
  "@tetel_azon"    NUMBER,
  arucikk          arucikk_tip,
  mennyiseg        NUMBER,
  kedvezmeny       NUMBER
)
/

CREATE TYPE rendelesi_tetelek_tip AS TABLE OF rendelesi_tetel_tip
/
```

```
CREATE TYPE rendeles_tip AUTHID CURRENT_USER AS OBJECT (
  rendelesszam          NUMBER,
  vasarlo               vasarlo_tip,
  rendeles_datuma       DATE,
  szallitas_datuma      TIMESTAMP,
  rendelesi_tetelek     rendelesi_tetelek_tip,
  szallitasi_cim        cim_tipus
)
/

-- A rendelési rendszer adatbázistáblái.

-- A vásárlók adatait tartalmazó tábla.
CREATE TABLE vasarlo (
  vasarlo_azon          NUMBER      PRIMARY KEY,
  vasarlo_nev           VARCHAR2(200),
  utca                  VARCHAR2(200),
  varos                 VARCHAR2(200),
  orszag                CHAR(2),
  irszam                VARCHAR2(20),
  telszam1              VARCHAR2(20),
  telszam2              VARCHAR2(20),
  telszam3              VARCHAR2(20)
)
ORGANIZATION INDEX OVERFLOW;

-- A rendelési adatokat tartalmazó tábla.
CREATE TABLE rendeles (
   rendelesszam         NUMBER    PRIMARY KEY,
   vasarlo_azon         NUMBER    REFERENCES vasarlo,
   rendeles_datuma      DATE,
   szallitas_datuma     TIMESTAMP,
   utca                 VARCHAR2(200), /* ahová szállítani kell */
   varos                VARCHAR2(200),
   orszag               CHAR(2),
   irszam               VARCHAR2(20)
);

-- Az árucikkeket leíró adatok.
CREATE TABLE arucikk (
  arucikk_azon      NUMBER    UNIQUE,
  ar                NUMBER,
  ado               NUMBER
);

-- A rendelési tételeket tartalmazó tábla.
CREATE TABLE rendelesi_tetel (
  rt_azon           NUMBER,
  rendeles_azon     NUMBER      REFERENCES rendeles,
  arucikk           NUMBER,
  mennyiseg         NUMBER,
  kedvezmeny        NUMBER,
  PRIMARY KEY (rendeles_azon, rt_azon)
);
```

```
-- Objektumnézetek létrehozása.

-- A vásárló objektumnézet.
CREATE OR REPLACE VIEW vasarlo_on OF vasarlo_tip
   WITH OBJECT IDENTIFIER(vasarlo_azon)
   AS SELECT v.vasarlo_azon, v.vasarlo_nev,
             cim_tipus(v.utca, v.varos, v.orszag, v.irszam),
             telszamok_tip(telszam1, telszam2, telszam3)
        FROM vasarlo v;

-- A rendelés objektumnézet.
CREATE OR REPLACE VIEW rendeles_on OF rendeles_tip
  WITH OBJECT IDENTIFIER (rendelesszam)
  AS SELECT r.rendelesszam,
            vasarlo_tip(v.vasarlo_azon, v.vasarlo_nev, v.cim, v.telszam),
            r.rendeles_datuma,
            r.szallitas_datuma,
            CAST( MULTISET(
                  SELECT rendelesi_tetel_tip( t.rt_azon,
                         arucikk_tip(t.arucikk, a.ar, a.ado),
                         t.mennyiseg, t.kedvezmeny)
                  FROM rendelesi_tetel t, arucikk a
                  WHERE t.rendeles_azon = r.rendelesszam AND
                        a.arucikk_azon=t.arucikk )
            AS rendelesi_tetelek_tip),
            cim_tipus(r.utca, r.varos, r.orszag, r.irszam)
     FROM rendeles r, vasarlo_on v
     WHERE r.vasarlo_azon = v.vasarlo_azon;

CREATE TABLE rendeles_xml (
  rendeles_azon     NUMBER,
  rendelesDok       SYS.XMLType
);

-------------------
-- Feltöltés
-------------------
-- A készlet létrehozása

INSERT INTO arucikk VALUES(1004, 6750.00, 2) ;
INSERT INTO arucikk VALUES(1011, 4500.23, 2) ;
INSERT INTO arucikk VALUES(1534, 2234.00, 2) ;
INSERT INTO arucikk VALUES(1535, 3456.23, 2) ;

-- Vásárlók bejegyzése

INSERT INTO vasarlo
  VALUES (1, 'Nagy Béla', 'Mellék utca 2.',
          'Holdváros', 'HU', '1111',
          '555-1212', NULL, NULL) ;
```

```
INSERT INTO vasarlo
  VALUES (2, 'Kovács János', 'Akác fasor 22.',
         'Debrecen', 'HU', '4032',
         '555-460', '555-211', NULL) ;

-- Rendelések beszúrása

INSERT INTO rendeles
  VALUES (1001, 1, to_date('1997.04.10.','YYYY.MM.DD.'),
         to_date('1997.05.10.','YYYY.MM.DD.'),
         NULL, NULL, NULL, NULL) ;

INSERT INTO rendeles
  VALUES (2001, 2, to_date('1997.04.20.','YYYY.MM.DD.'),
         to_date('1997.05.20.','YYYY.MM.DD.'),
         'Fő utca 13.', 'Napfalva', 'HU', '2222') ;

-- A részletes rendelési adatok beszúrása

INSERT INTO rendelesi_tetel VALUES(1, 1001, 1534, 12,  0) ;
INSERT INTO rendelesi_tetel VALUES(2, 1001, 1535, 10, 10) ;
INSERT INTO rendelesi_tetel VALUES(1, 2001, 1004,  1,  0) ;
INSERT INTO rendelesi_tetel VALUES(2, 2001, 1011,  2,  1) ;


-----------------------------------------------------------
-- A DBMS_XMLGEN csomag segítségével XML formátumú
-- rendeléseket készítünk, és letároljuk a rendeles_xml
-- táblában, mint SYS.XMLType értéket
-----------------------------------------------------------

DECLARE
   kornyezet DBMS_XMLGEN.ctxHandle;
   rend_dok  SYS.XMLType;
BEGIN
  -- A lekérdezési környezet lekérése
  kornyezet := DBMS_XMLGEN.newContext('
                    SELECT rendelesszam, vasarlo, r.rendeles_datuma,
                           r.szallitas_datuma,
                           rendelesi_tetelek, szallitasi_cim
                    FROM rendeles_on r'
               );
  -- A maximálisan lekérendő sorok számát állítsuk egyre.
  DBMS_XMLGEN.setMaxRows(kornyezet, 1);
  -- Az eredményül kapott sorok halmazát közrefogó elemet kihagyjuk, a
  -- sorokat pedig Rendelés elemek fogják elválasztani
  DBMS_XMLGEN.setRowSetTag(kornyezet, NULL);
  DBMS_XMLGEN.setRowTag(kornyezet, 'Rendelés');
  LOOP
    -- Most lekérjük a rendelést XML formátumban.
    rend_dok := DBMS_XMLGEN.getXMLType(kornyezet);
    -- Ha nincsenek további sorok, kilépünk.
    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(kornyezet) = 0;
```

```
    -- A rendeles_xml táblában letároljuk az XML-dokumentumot.
    INSERT INTO rendeles_xml (rendeles_azon, rendelesDok)
       VALUES(rend_dok.extract('//RENDELESSZAM/text()').getNumberVal(),
              rend_dok);
  END LOOP;
END;
/
```

A

```
SELECT rendelesDok FROM rendeles_xml;
```

lekérdezés a következő eredményt adja:

```
RENDELESDOK
--------------------------------------------------------------------------------
<RENDELES>
  <RENDELESSZAM>1001</RENDELESSZAM>
  <VASARLO>
    <VASARLO_AZON>1</VASARLO_AZON>
    <VASARLO_NEV>Nagy Béla</VASARLO_NEV>
    <CIM>
     <UTCA>Mellék utca 2.</UTCA>
     <VAROS>Holdváros</VAROS>
     <ORSZAG>HU</ORSZAG>
     <IRANYITOSZAM>1111</IRANYITOSZAM>
    </CIM>
    <TELSZAM>
     <VARCHAR2>555-1212</VARCHAR2>
    </TELSZAM>
  </VASARLO>
  <RENDELES_DATUMA>97-ÁPR-10</RENDELES_DATUMA>
  <SZALLITAS_DATUMA>97-MÁJ-10 00.00.00.000000</SZALLITAS_DATUMA>
  <RENDELESI_TETELEK>
    <RENDELESI_TETEL_TIP tetel_azon="1">
     <ARUCIKK arucikk_azon="1534">
      <AR>2234</AR>
      <ADO>2</ADO>
     </ARUCIKK>
     <MENNYISEG>12</MENNYISEG>
     <KEDVEZMENY>0</KEDVEZMENY>
   </RENDELESI_TETEL_TIP>
   <RENDELESI_TETEL_TIP tetel_azon="2">
    <ARUCIKK arucikk_azon="1535">
     <AR>3456.23</AR>
     <ADO>2</ADO>
    </ARUCIKK>
    <MENNYISEG>10</MENNYISEG>
    <KEDVEZMENY>10</KEDVEZMENY>
   </RENDELESI_TETEL_TIP>
  </RENDELESI_TETELEK>
  <SZALLITASI_CIM/>
</RENDELES>
```

```
<RENDELES>
  <RENDELESSZAM>2001</RENDELESSZAM>
  <VASARLO>
    <VASARLO_AZON>2</VASARLO_AZON>
    <VASARLO_NEV>Kovács János</VASARLO_NEV>
    <CIM>
     <UTCA>Akác fasor 22.</UTCA>
     <VAROS>Debrecen</VAROS>
     <ORSZAG>HU</ORSZAG>
     <IRANYITOSZAM>4032</IRANYITOSZAM>
    </CIM>
    <TELSZAM>
     <VARCHAR2>555-460</VARCHAR2>
     <VARCHAR2>555-211</VARCHAR2>
    </TELSZAM>
  </VASARLO>
  <RENDELES_DATUMA>97-ÁPR-20</RENDELES_DATUMA>
  <SZALLITAS_DATUMA>97-MÁJ-20 00.00.00.000000</SZALLITAS_DATUMA>
  <RENDELESI_TETELEK>
    <RENDELESI_TETEL_TIP tetel_azon="1">
      <ARUCIKK arucikk_azon="1004">
       <AR>6750</AR>
       <ADO>2</ADO>
      </ARUCIKK>
      <MENNYISEG>1</MENNYISEG>
      <KEDVEZMENY>0</KEDVEZMENY>
    </RENDELESI_TETEL_TIP>
    <RENDELESI_TETEL_TIP tetel_azon="2">
     <ARUCIKK arucikk_azon="1011">
       <AR>4500.23</AR>
       <ADO>2</ADO>
     </ARUCIKK>
     <MENNYISEG>2</MENNYISEG>
     <KEDVEZMENY>1</KEDVEZMENY>
    </RENDELESI_TETEL_TIP>
  </RENDELESI_TETELEK>
  <SZALLITASI_CIM>
    <UTCA>Fő utca 13.</UTCA>
    <VAROS>Napfalva</VAROS>
    <ORSZAG>HU</ORSZAG>
    <IRANYITOSZAM>2222</IRANYITOSZAM>
  </SZALLITASI_CIM>
</RENDELES>
```

3.1.3. A SYS_XMLGEN függvény

Az Oracle9*i* egy új SQL-függvényt vezet be az SQL-kérdésekből történő XML-generálásra. Ennek a függvénynek a neve `SYS_XMLGEN`. A DBMS_XMLGEN csomag a lekérdezések szintjén operál, a *teljes* lekérdezésre ad összesített (aggregált) eredményt. Ezzel szemben a SYS_XMLGEN soronkénti függvényként az SQL-lekérde-

zésben paramétereként kapott értéket XML formátumúvá alakítja. A függvény szintaktikája az alábbi:

SYS_XMLGEN (*kif* [*formátum*])

A SYS_XMLGEN első paramétere az XML-dokumentummá konvertálandó skalár érték, objektumpéldány vagy XMLType példány. A függvénynek van egy opcionálisan megadható, XMLGenFormatType típusú második paramétere is, amelynek segítségével az eredményül kapott XML-dokumentum formázására vonatkozó előírásokat állíthatjuk be. Ha a második paraméter NULL, vagy hiányzik, akkor a formázás alapértelmezés szerint megy végbe. A SYS_XMLGEN függvény egy XMLType típusú értéket ad vissza.

- Ha *kif* skalár érték, a függvény egy, ezt a skalárt tartalmazó XML-elemmel tér vissza.
- Ha *kif* egy objektumtípus példánya, a függvény a típus attribútumait XML-elemekre képezi le.
- Ha *kif* egy XMLType példány, akkor a függvény ezt a példányt egy olyan XML-elembe ágyazza be, amelynek alapértelmezett neve ROW.

Alaphelyzetben az XML-dokumentum elemei a *kif* elemeinek felelnek meg, azaz, ha például *kif* egy oszlop neve, akkor az XML-elem neve is ugyanez lesz. Ha másképpen szeretnénk formázni az XML-dokumentumot, akkor meg kell adnunk az XMLGenFormatType típusú *formátum* paramétert.

A függvény segítségével SQL-lekérdezéseken belül tudunk XML-példányokat létrehozni és lekérdezni:

```
SELECT SYS_XMLGEN(azon) FROM alkalmazott WHERE nev LIKE 'Kovács%';
```

A lekérdezés eredménye az alábbi XML-dokumentum:

```
<?xml version="1.0"?>
 <AZON>2</AZON>
```

A következő példa a 2-es azonosítóval rendelkező alkalmazott beosztását kérdezi le, és ebből egy BEOSZTAS elemet tartalmazó XMLType példányt állít elő.

```
SELECT SYS_XMLGEN(beosztas).getStringVal()
   FROM alkalmazott
   WHERE azon = 2;

SYS_XMLGEN(BEOSZTAS).GETSTRINGVAL()
--------------------------------------------------------------------------
<?xml version="1.0"?>
 <BEOSZTAS>osztályvezető</BEOSZTAS>
```

A SYS_XMLGEN és SYS_XMLAGG függvények paramétereit egy XMLGenFormatType típusú objektum segítségével formázhatjuk kedvünkre. A 3.5. táblázat az XMLGenFormatType attribútumait mutatja be.

3.5. táblázat. *XMLGenFormatType attribútumok*

XMLGenFormatType attribútum	*Leírás*
`enclTag`	A körülölelő XML-elem neve.
`schemaType`	Jelenleg még nem támogatott, de a későbbi kiadásokban annak megadására szolgál, hogy hogyan kell az XML Schema-t generálni.
`processingIns`	Olyan feldolgozási utasításokat adhatunk meg segítségével, amelyeket a dokumentum elején kell elhelyezni (pl. stíluslap megadására szolgáló feldolgozási utasítás).
`Dburl`	Jelenleg nem használt.
`targetNameSpace`	Jelenleg nem használt, de a későbbiekben az XML Schema generálásban lesz szerepe.

Egy formázó objektumot az XMLGenFormatType `createFormat` nevű statikus tagfüggvényével hozhatunk létre. A függvény minden paramétere rendelkezik alapértelmezett értékkel.

```
-- A formázó objektumot létrehozó függvény.
STATIC MEMBER FUNCTION createFormat(
      enclTag IN VARCHAR2 := NULL,
      schemaType IN VARCHAR2 := 'NO_SCHEMA'
      schemaName IN VARCHAR2 := NULL,
      targetNameSpace IN VARCHAR2 := NULL,
      dburl IN VARCHAR2 := NULL,
      processingIns IN VARCHAR2 := NULL)
  RETURN XMLGenFormatType;
```

Példák

3.18. példa

Egy skalár érték XML-elemtartalommá alakítása

Ha a SYS_XMLGEN egy skalár értéket kap paramétereként, akkor a függvény azt egy, a skalár értéket tartalmazó elemmé alakítja. Például a

```
SELECT SYS_XMLGEN(azon) FROM alkalmazott WHERE rownum = 1;
```

lekérdezés egy olyan XML-dokumentumot ad eredményül, amely az `AZON` oszlop értékét XML-elemként tartalmazza:

```
SYS_XMLGEN(AZON)
----------------
<AZON>1</AZON>
```

A körülölelő elem neve – jelen esetben AZON – a függvény által kapott paraméterből származik. Megjegyezzük, hogy a SELECT utasítás eredménye egy XMLType típusú értéket tartalmazó sor.

Jelenleg az SQL*Plus az XMLType típusú értékek megjelenítésekor a dokumentumelemen kívül elhelyezkedő XML-konstrukciókat (például XML-deklaráció, dokumentumtípus-deklaráció, stíluslap-deklaráció) nem jeleníti meg, ezért ha ezeket is szeretnénk a kimenetben látni, a getClobVal függvény segítségével megjelenítés előtt ki kell nyernünk a dokumentumot LOB formátumban. Például a

```
SELECT SYS_XMLGEN(azon).getClobVal() FROM alkalmazott WHERE rownum = 1;
```

lekérdezés eredménye:

```
SYS_XMLGEN(AZON).GETCLOBVAL()
-----------------------------
<?xml version="1.0"?>
 <AZON>1</AZON>
```

Amennyiben az oszlop neve közvetlenül nem határozható meg, mint az alábbi esetben, az XML-elem neve alapértelmezés szerint <ROW> lesz.

```
SELECT SYS_XMLGEN(azon * 2).getClobVal() FROM alkalmazott WHERE rownum =
1;
```

Mivel a kifejezésből nem határozható meg név, ezért a lekérdezés eredménye:

```
SYS_XMLGEN(AZON*2).GETCLOBVAL()
-------------------------------
<?xml version="1.0"?>
 <ROW>2</ROW>
```

Az alapértelmezett ROW elemnevet egy XMLGenFormatType objektum megadásával írhatjuk felül:

```
SELECT SYS_XMLGEN(azon *2,
SYS.XMLGenFormatType.createFormat('ALKALMAZOTT_AZONOSITO')).getClobVal()
FROM alkalmazott WHERE rownum = 1;
```

Ez a következő XML-dokumentumot eredményezi:

```
SYS_XMLGEN(AZON*2,SYS.XMLGENFORMATTYPE.CREATEFORMAT('ALKALMAZOTT_AZONOSITO')).GE
-------------------------------------------------------------------------------
<?xml version="1.0"?>
 <ALKALMAZOTT_AZONOSITO>2</ALKALMAZOTT_AZONOSITO>
```

A SYS_XMLGEN függvény paramétere nem lehet CURSOR-kifejezés.

3.19. példa

Egy felhasználói adattípus XML-lé alakítása

Ha a SYS_XMLGEN első paramétere egy felhasználói adattípus, akkor az XML-re történő átalakítás egy kanonikus leképezés alapján megy végbe. A kanonikus leképezés során a felhasználói típus attribútumai XML-elemekre lesznek leképezve. Ez alól kivételt képeznek azon attribútumok, amelyek nevének kezdő karaktere `@`: ezek a típus nevével fémjelzett XML-elem attribútumai lesznek.

A 3.17. példát felhasználva, egy hierarchikus szerkezetű XML-dokumentumot a következő módon tudunk létrehozni:

```
SELECT SYS_XMLGEN(
  reszleg_tip(kod, nev,
     CAST(MULTISET(
          SELECT azon, nev
          FROM alkalmazott a
          WHERE a.reszleg = r.kod) AS
alkalmazottlista_tip))).getClobVal()
     AS reszleg_xml
FROM reszleg r;
```

A MULTISET operátor az adott részlegen dolgozó alkalmazottak halmazát listaként tekinti, a CAST operátor pedig ezt a listát a megfelelő kollekciótípusúvá alakítja. Ezután létrehozzuk a részleg példányt, amiből a SYS_XMLGEN segítségével előállítjuk az objektumpéldánynak megfelelő XML-dokumentumot.

A fenti lekérdezés eredménye egy XML-dokumentum a `reszleg` tábla *minden sorára*:

```
<?xml version="1.0"?>
<ROW reszleg_azon="10">
 <RESZLEGNEV>Számlázás</RESZLEGNEV>
 <ALKALMAZOTTLISTA>
  <ALKALMAZOTT_TIP alkalmazott_azon="5">
   <NEV>Kiss József</NEV>
  </ALKALMAZOTT_TIP>
  <ALKALMAZOTT_TIP alkalmazott_azon="6">
   <NEV>Balogh Csaba</NEV>
  </ALKALMAZOTT_TIP>
  <ALKALMAZOTT_TIP alkalmazott_azon="7">
   <NEV>Tóth Andrea</NEV>
  </ALKALMAZOTT_TIP>
 </ALKALMAZOTTLISTA>
</ROW>
```

Ebből a példából látható a DBMS_XMLGEN és a SYS_XMLGEN közötti különbség is: az előbbi a teljes eredményhalmazon dolgozik, míg az utóbbi sor szintű függvényként egy sor oszlopain, illetve az azokból képzett kifejezéseken operál.

3.20. példa

XMLType példányok újrafelhasználása

Ha a SYS_XMLGEN első paramétere egy XML formátumú adat, a függvény bezárja a dokumentumot, vagy dokumentumtöredéket egy – alapértelmezés szerint ROW nevű – XML-elembe. Ennek segítségével a dokumentumtöredékeket jólformált dokumentumokká alakíthatjuk.

Egy dokumentumból egy XPath-kifejezés segítségével egy dokumentumtöredéket nyerhetünk ki. Tekintsük például a 3.17. példában eredményül kapott dokumentumok közül a 2001-es rendelési számhoz tartozót.

Ebből a dokumentumból a

```
SELECT rx.rendelesDok.extract('//ARUCIKK')
   FROM rendeles_xml rx WHERE rx.rendeles_azon=2001;
```

lekérdezést végrehajtva a következő dokumentumtöredéket kapjuk:

```
<ARUCIKK arucikk_azon="1004">
  <AR>6750</AR>
  <ADO>2</ADO>
</ARUCIKK>
<ARUCIKK arucikk_azon="1011">
  <AR>4500.23</AR>
  <ADO>2</ADO>
</ARUCIKK>
```

Ez nem jólformált XML-dokumentum, hiszen egy jólformált dokumentumban a legkülső szinten csak egy elem (a dokumentumelem) helyezkedhet el. Ezt a dokumentumtöredéket azonban egy SYS_XMLGEN hívással könnyedén jólformálttá tudjuk alakítani, hiszen így egy, az eddig meglévő adatokat egy új elemmel közre tudjuk fogni. A

```
SELECT SYS_XMLGEN(rx.rendelesDok.extract('//ARUCIKK')).getClobVal()
   FROM rendeles_xml rx WHERE rx.rendeles_azon=2001;
```

lekérdezésben szereplő SYS_XMLGEN függvényhívás eredményeként a következő XML-dokumentumot kapjuk:

```
<?xml version="1.0"?>
<ROW>
<ARUCIKK arucikk_azon="1004">
  <AR>6750</AR>
  <ADO>2</ADO>
</ARUCIKK>
<ARUCIKK arucikk_azon="1011">
  <AR>4500.23</AR>
  <ADO>2</ADO>
</ARUCIKK>
</ROW>
```

Itt is igazak a korábban leírtak, tehát amennyiben a paraméter egy tábla vagy nézet valamely oszlopa, akkor az XML-elem annak nevéből képződik, illetve az opcionálisan megadható formázó objektum segítségével a körülölelő elem neve könnyedén felülírható.

3.21. példa

A SYS_XMLGEN használata objektumnézetek segítségével

Ebben a példában a 3.17. példában megismert `rendeles_on` objektumnézet alapján végezzük a lekérdezést:

```
SELECT SYS_XMLGEN(value(r),
SYS.XMLGenFormatType.createFormat('Rendelés')).getClobVal()
rendeles
FROM rendeles_on r
WHERE r.rendelesszam=1001;
```

Eredményül a következő dokumentumot kapjuk:

```
RENDELES
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<RENDELES>
 <RENDELESSZAM>1001</RENDELESSZAM>
 <VASARLO>
  <VASARLO_AZON>1</VASARLO_AZON>
  <VASARLO_NEV>Nagy Béla</VASARLO_NEV>
  <CIM>
   <UTCA>Mellék utca 2.</UTCA>
   <VAROS>Holdváros</VAROS>
   <ORSZAG>HU</ORSZAG>
   <IRANYITOSZAM>1111</IRANYITOSZAM>
  </CIM>
  <TELSZAM>
   <VARCHAR2>555-1212</VARCHAR2>
  </TELSZAM>
 </VASARLO>
 <RENDELES_DATUMA>97-ÁPR-10</RENDELES_DATUMA>
 <SZALLITAS_DATUMA>97-MÁJ-10 00.00.00.000000</SZALLITAS_DATUMA>
 <RENDELESI_TETELEK>
  <RENDELESI_TETEL_TIP tetel_azon="1">
   <ARUCIKK arucikk_azon="1534">
    <AR>2234</AR>
    <ADO>2</ADO>
   </ARUCIKK>
   <MENNYISEG>12</MENNYISEG>
   <KEDVEZMENY>0</KEDVEZMENY>
  </RENDELESI_TETEL_TIP>
  <RENDELESI_TETEL_TIP tetel_azon="2">
   <ARUCIKK arucikk_azon="1535">
```

```
      <AR>3456.23</AR>
      <ADO>2</ADO>
     </ARUCIKK>
     <MENNYISEG>10</MENNYISEG>
     <KEDVEZMENY>10</KEDVEZMENY>
    </RENDELESI_TETEL_TIP>
   </RENDELESI_TETELEK>
   <SZALLITASI_CIM/>
  </RENDELES>
```

3.1.4. A SYS_XMLAGG függvény

A SYS_XMLAGG függvény a bemenő dokumentumokat aggregálja, és előállít egy XML-dokumentumot. A SYS_XMLAGG egy XMLType típusú objektumot kap paraméterként, és a sorokban található értékeket csoportonként XML-dokumentummá alakítja.

A SYS_XMLAGG függvény szintaktikája a következő:

`SYS_XMLAGG(`*kif* [*formátum*]`)`

A függvény a *kif* által reprezentált XML-dokumentumokat, illetve dokumentumtöredékeket aggregálja, és egyetlen XML-dokumentumot állít elő. A dokumentumelem neve alapértelmezés szerint `ROWSET` lesz, az opcionális – XMLGenFormatType típusú – *formátum* paramétert megadva azonban ezt megváltoztathatjuk.

3.22. példa

XML formátumú adat relációsból történő aggregálása
Nézzük a következő lekérdezést:

```
SELECT nvl(reszleg, 0) reszlegkod, SYS_XMLAGG(SYS_XMLGEN(nev),
   SYS.XMLGenFormatType.createFormat
   ('AlkalmazottCsoport')).getClobVal() reszleg_dolgozoi
  FROM alkalmazott
  GROUP BY reszleg
  ORDER BY nvl(reszleg, 0);
```

A lekérdezés eredménye részlegenként egy XML-dokumentum:

```
RESZLEGKOD RESZLEG_DOLGOZOI
---------- ------------------------------------------------------------
         0  <?xml version="1.0"?>
           <AlkalmazottCsoport>
             <NEV>Szabó Péter</NEV>
           </AlkalmazottCsoport>

        10 <?xml version="1.0"?>
           <AlkalmazottCsoport>
```

```
        <NEV>Kiss József</NEV>
        <NEV>Balogh Csaba</NEV>
        <NEV>Tóth Andrea</NEV>
      </AlkalmazottCsoport>

20 <?xml version="1.0"?>
   <AlkalmazottCsoport>
      <NEV>Gipsz Jakab</NEV>
      <NEV>Horváth Ildikó</NEV>
      <NEV>Kis András</NEV>
      <NEV>Szabó Sándor</NEV>
      <NEV>Nagy Ferenc</NEV>
   </AlkalmazottCsoport>

30 <?xml version="1.0"?>
   <AlkalmazottCsoport>
      <NEV>Kovács János</NEV>
      <NEV>Nagy Géza</NEV>
      <NEV>József István</NEV>
      <NEV>Komor Ágnes</NEV>
   </AlkalmazottCsoport>
```

3.23. példa

Táblákban tárolt XMLType példányok aggregálása

A SYS_XMLAGG függvény táblákban tárolt, illetve függvények által visszaadott XMLType példányok aggregálására is használható. A `rendeles_xml` táblának a 3.17. példa kódjának végrehajtása után előállt két sorának egyetlen XML-dokumentumba történő aggregálása a következő lekérdezéssel történhet:

```
SELECT SYS_XMLAGG(rendelesDok,
SYS.XMLGenFormatType.createFormat('RENDELESEK')).getClobVal()
FROM rendeles_xml;
```

A lekérdezés eredménye:

```
<?xml version="1.0"?>
<RENDELESEK>
<RENDELES rend_szam="1001">
  <RNEV>Rend_1</RNEV>
  <VASARLO>
    <VASARLO_AZON>1</VASARLO_AZON>
    <VASARLO_NEV>Nagy Béla</VASARLO_NEV>
    <CIM>
      <UTCA>Mellék utca 2.</UTCA>
      <VAROS>Holdváros</VAROS>
      <ORSZAG>HU</ORSZAG>
      <IRANYITOSZAM>1111</IRANYITOSZAM>
    </CIM>
    <TELSZAM>
      <VARCHAR2>555-1212</VARCHAR2>
```

```
      </TELSZAM>
    </VASARLO>
    <RENDELES_DATUMA>97-ÁPR-10</RENDELES_DATUMA>
    <SZALLITAS_DATUMA>97-MÁJ-10 00.00.00.000000</SZALLITAS_DATUMA>
    <RENDELESI_TETELEK>
      <RENDELESI_TETEL_TIP tetel_azon="1">
        <ARUCIKK arucikk_azon="1534">
          <AR>2234</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>12</MENNYISEG>
        <KEDVEZMENY>0</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
      <RENDELESI_TETEL_TIP tetel_azon="2">
        <ARUCIKK arucikk_azon="1535">
          <AR>3456.23</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>10</MENNYISEG>
        <KEDVEZMENY>10</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
    </RENDELESI_TETELEK>
    <SZALLITASI_CIM/>
  </RENDELES>
  <RENDELES rend_szam="2001">
    <RNEV>Rend_2</RNEV>
    <VASARLO>
      <VASARLO_AZON>2</VASARLO_AZON>
      <VASARLO_NEV>Kovács János</VASARLO_NEV>
      <CIM>
        <UTCA>Akác fasor 22.</UTCA>
        <VAROS>Debrecen</VAROS>
        <ORSZAG>HU</ORSZAG>
        <IRANYITOSZAM>4032</IRANYITOSZAM>
      </CIM>
      <TELSZAM>
        <VARCHAR2>555-460</VARCHAR2>
        <VARCHAR2>555-211</VARCHAR2>
      </TELSZAM>
    </VASARLO>
    <RENDELES_DATUMA>97-ÁPR-20</RENDELES_DATUMA>
    <SZALLITAS_DATUMA>97-MÁJ-20 00.00.00.000000</SZALLITAS_DATUMA>
    <RENDELESI_TETELEK>
      <RENDELESI_TETEL_TIP tetel_azon="1">
        <ARUCIKK arucikk_azon="1004">
          <AR>6750</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>1</MENNYISEG>
        <KEDVEZMENY>0</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
      <RENDELESI_TETEL_TIP tetel_azon="2">
        <ARUCIKK arucikk_azon="1011">
```

```
          <AR>4500.23</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>2</MENNYISEG>
        <KEDVEZMENY>1</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
    </RENDELESI_TETELEK>
    <SZALLITASI_CIM>
      <UTCA>Fő utca 13.</UTCA>
      <VAROS>Napfalva</VAROS>
      <ORSZAG>HU</ORSZAG>
      <IRANYITOSZAM>2222</IRANYITOSZAM>
    </SZALLITASI_CIM>
  </RENDELES>
  </RENDELESEK>
```

3.24. példa

XMLType töredékek aggregálása

Az XMLType típus `extract` függvényével kapott dokumentumtöredékeket is aggregálhatjuk a SYS_XMLAGG függvény segítségével. Például, ha az előző példából csak a SZALLITAS_DATUMA elemeket szeretnénk kinyerni, azokat a következőképpen aggregálhatjuk:

```
SELECT SYS_XMLAGG(rx.rendelesDok.extract('//
SZALLITAS_DATUMA')).getClobVal() szallitasi_datumok
FROM rendeles_xml rx;
```

Ennek eredményeként a következő dokumentumot kapjuk:

```
<?xml version="1.0"?>
<ROWSET>
<SZALLITAS_DATUMA>97-MÁJ-10 00.00.00.000000</SZALLITAS_DATUMA>
<SZALLITAS_DATUMA>97-MÁJ-20 00.00.00.000000</SZALLITAS_DATUMA>
</ROWSET>
```

3.25. példa

Az összes rendelés egyetlen XML-dokumentumba történő aggregálása

A `rendeles_on` objektumnézeten végrehajtott

```
SELECT SYS_XMLAGG(SYS_XMLGEN(value(r),
SYS.XMLGenFormatType.createFormat('Rendelés'))).getClobVal()
rendeles
FROM rendeles_on r;
```

lekérdezés a rendeléseket tartalmazó dokumentumtöredékek (lásd 3.21. példa) aggregálásával az összes rendelést egyetlen XML-dokumentumba fogja össze, amelynek dokumentumeleme az alapértelmezett `ROWSET` lesz:

```
<?xml version="1.0"?>
<ROWSET>
<RENDELES rend_szam="1001">
  <RNEV>Rend_1</RNEV>
  <VASARLO>
    <VASARLO_AZON>1</VASARLO_AZON>
    <VASARLO_NEV>Nagy Béla</VASARLO_NEV>
    <CIM>
      <UTCA>Mellék utca 2.</UTCA>
      <VAROS>Holdváros</VAROS>
      <ORSZAG>HU</ORSZAG>
      <IRANYITOSZAM>1111</IRANYITOSZAM>
    </CIM>
    <TELSZAM>
      <VARCHAR2>555-1212</VARCHAR2>
    </TELSZAM>
  </VASARLO>
  <RENDELES_DATUMA>97-ÁPR-10</RENDELES_DATUMA>
  <SZALLITAS_DATUMA>97-MÁJ-10 00.00.00.000000</SZALLITAS_DATUMA>
  <RENDELESI_TETELEK>
    <RENDELESI_TETEL_TIP tetel_azon="1">
      <ARUCIKK arucikk_azon="1534">
        <AR>2234</AR>
        <ADO>2</ADO>
      </ARUCIKK>
      <MENNYISEG>12</MENNYISEG>
      <KEDVEZMENY>0</KEDVEZMENY>
    </RENDELESI_TETEL_TIP>
    <RENDELESI_TETEL_TIP tetel_azon="2">
      <ARUCIKK arucikk_azon="1535">
        <AR>3456.23</AR>
        <ADO>2</ADO>
      </ARUCIKK>
      <MENNYISEG>10</MENNYISEG>
      <KEDVEZMENY>10</KEDVEZMENY>
    </RENDELESI_TETEL_TIP>
  </RENDELESI_TETELEK>
  <SZALLITASI_CIM/>
</RENDELES>
<RENDELES rend_szam="2001">
  <RNEV>Rend_2</RNEV>
  <VASARLO>
    <VASARLO_AZON>2</VASARLO_AZON>
    <VASARLO_NEV>Kovács János</VASARLO_NEV>
    <CIM>
      <UTCA>Akác fasor 22.</UTCA>
      <VAROS>Debrecen</VAROS>
      <ORSZAG>HU</ORSZAG>
      <IRANYITOSZAM>4032</IRANYITOSZAM>
    </CIM>
    <TELSZAM>
      <VARCHAR2>555-460</VARCHAR2>
      <VARCHAR2>555-211</VARCHAR2>
```

```
      </TELSZAM>
    </VASARLO>
    <RENDELES_DATUMA>97-ÁPR-20</RENDELES_DATUMA>
    <SZALLITAS_DATUMA>97-MÁJ-20 00.00.00.000000</SZALLITAS_DATUMA>
    <RENDELESI_TETELEK>
      <RENDELESI_TETEL_TIP tetel_azon="1">
        <ARUCIKK arucikk_azon="1004">
          <AR>6750</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>1</MENNYISEG>
        <KEDVEZMENY>0</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
      <RENDELESI_TETEL_TIP tetel_azon="2">
        <ARUCIKK arucikk_azon="1011">
          <AR>4500.23</AR>
          <ADO>2</ADO>
        </ARUCIKK>
        <MENNYISEG>2</MENNYISEG>
        <KEDVEZMENY>1</KEDVEZMENY>
      </RENDELESI_TETEL_TIP>
    </RENDELESI_TETELEK>
    <SZALLITASI_CIM>
      <UTCA>Fő utca 13.</UTCA>
      <VAROS>Napfalva</VAROS>
      <ORSZAG>HU</ORSZAG>
      <IRANYITOSZAM>2222</IRANYITOSZAM>
    </SZALLITASI_CIM>
  </RENDELES>
  </ROWSET>
```

3.1.5. TABLE függvények

Az előző szakaszban az Oracle új függvényeiről volt szó, amelyek XML-példányok XPath-szerű szintaktikán alapuló lekérdezését tették lehetővé. Szükség lehet azonban a fordított irányra is, azaz egy XML-dokumentum tartalmát esetleg relációs, illetve objektumrelációs táblákba kell beszúrni. Ezt a TABLE függvények segítségével tehetjük meg.

A TABLE függvényeket az Oracle9*i* vezette be, tetszőleges (akár adatbázisbeli, akár külső) adatok SQL-sorok kollekciójaként történő modellezésére. A TABLE függvények a minél jobb teljesítmény elérése érdekében csővezetékszerűen és párhuzamosan hajtódnak végre. TABLE függvények segítségével az XML-dokumentumot SQL-sorokká tudjuk alakítani, amelyeket aztán a hagyományos SQL-lekérdezések segítségével fel tudunk dolgozni, és hagyományos relációs, illetve objektumrelációs táblákba be tudjuk szúrni.

Az Oracle8*i*-ben már rendelkezésünkre állt egy kollekciót visszaadó függvény, amelyet a SELECT utasítás FROM részében használhattunk, azonban ennek a függ-

vénynek elő kellett állítania a teljes kollekciót, mielőtt a külső lekérdezőblokk hozzáférhetett volna.

A TABLE függvények esetén mindez csővezetékelvűen és párhuzamosan zajlik, így a függvénynek nem kell a teljes kollekciót egyszerre a memóriában tárolnia. Ehelyett egy csővezetéken azonnal a külső lekérdezőblokk felé továbbítja az eredményeket, így a válaszidő gyorsabb, a memóriaigény pedig kisebb lesz.

XML-dokumentumok feldolgozására célszerű egy kollekció típusú értéket visszaadó TABLE függvényt írni, amely az Oracle9*i* XML-elemzőjét használva akár SAX típusú feldolgozást végezhet, vagy az `extract` függvény segítségével az XML-dokumentum bizonyos részeit kinyerheti.

3.26. példa

Rendelési információk kinyerése XML-dokumentumból és azok relációs táblába történő beszúrása

Amennyiben egy, a korábbi példákban bemutatott rendelési információkat leíró dokumentumban található információkat (azaz a rendelés adatait) relációs táblába szeretnénk beszúrni, először létre kell hoznunk az eredmény szerkezetét leíró típusokat:

```
CREATE TYPE rendelesSor_tip AS OBJECT
(
  rendeles_nev VARCHAR2(20),
  utca         VARCHAR2(20),
  varos        VARCHAR2(20),
  orszag       CHAR(2),
  irszam       CHAR(10)
)
/
CREATE TYPE rendelesSor_lista AS TABLE OF rendelesSor_tip
/
```

Ezen típusok használatával a következő TABLE függvényt készíthetjük el:

```
CREATE OR REPLACE FUNCTION rendeles_szetszor (p IN SYS.XMLType) RETURN
rendelesSor_lista PIPELINED IS
  eredmeny rendelesSor_tip;
  rend_xml SYS.XMLType;
  i        BINARY_INTEGER := 1;
  uj_p     SYS.XMLType := p;
BEGIN
  LOOP
    -- Kinyerjük az i-edik rendelés adatait.
    rend_xml := p.extract('//RENDELES['||i||']');
    EXIT WHEN rend_xml IS NULL;
    eredmeny := rendelesSor_tip('Rend_'||
                  rend_xml.extract('/RENDELES/@rend_szam'),
                  rend_xml.extract('/RENDELES/SZALLITASI_CIM/' ||
                                   'UTCA/text()').getStringVal(),
                  rend_xml.extract('/RENDELES/SZALLITASI_CIM/' ||
                                   'VAROS/text()').getStringVal(),
```

```
                    rend_xml.extract('/RENDELES/SZALLITASI_CIM/' ||
                                     'ORSZAG/text()').getStringVal(),
                    rend_xml.extract('/RENDELES/SZALLITASI_CIM/' ||
                                     'IRANYITOSZAM/text()').getStringVal());
    PIPE ROW(eredmeny);
    i := i + 1;
  END LOOP;
  RETURN;
END rendeles_szetszor;
/
```

Ezt a függvényt egy lekérdezés FROM részében hívhatjuk meg:

```
SELECT *
   FROM TABLE( CAST(
      rendeles_szetszor(
       SYS.XMLType.createXML(
         '<?xml version="1.0"?>
          <RENDELESEK>
           <RENDELES rend_szam="3001">
            <RNEV>Rend_3</RNEV>
            <SZALLITASI_CIM>
              <UTCA>Akác fasor 22.</UTCA>
              <VAROS>Debrecen</VAROS>
              <ORSZAG>HU</ORSZAG>
              <IRANYITOSZAM>4032</IRANYITOSZAM>
            </SZALLITASI_CIM>
           </RENDELES>
           <RENDELES rend_szam="4001">
            <RNEV>Rend_4</RNEV>
            <SZALLITASI_CIM>
              <UTCA>Mellék utca 40.</UTCA>
              <VAROS>Vásárhely</VAROS>
              <ORSZAG>HU</ORSZAG>
              <IRANYITOSZAM>9408</IRANYITOSZAM>
            </SZALLITASI_CIM>
           </RENDELES>
          </RENDELESEK>')
      ) AS rendelesSor_lista));
```

Az előző példában az XML-dokumentum létrehozására az XMLType statikus konstruktorát használtuk, de ez természetesen tetszőleges XMLType típusú értékkel (változóval, illetve lekérdezés eredményével is) helyettesíthető.

A fenti lekérdezés a következő eredményt adja:

```
RENDELES_NEV         UTCA                 VAROS                OR IRSZAM
-------------------- -------------------- -------------------- -- ------
Rend_3001            Akác fasor 22.       Debrecen             HU 4032
Rend_4001            Mellék utca 40.      Vásárhely            HU 9408
```

Ezt az eredményt egy INSERT utasítás segítségével könnyedén relációs táblába szúrhatjuk.

3.2. Az XML SQL Utility (XSU)

Az utóbbi időszakban az XML szabványos adatcsere formátummá nőtte ki magát. Alkalmazások egymás közötti, valamint az alkalmazások és az ügyfelek adatcseréje egyaránt XML formátumban történik. A kicserélendő adatok túlnyomó többsége azonban relációs és objektumrelációs adatbázis-kezelő rendszerekben tárolódik, így a küldő oldalon szükség van az adatbázisban tárolt adatok XML formátumúra alakítására, míg a fogadó oldalon az XML formátumú adatok (objektum)relációs háttérrendszerbe történő beszúrására kell megfelelő válaszokat adni.

Az Oracle XML SQL Utility eszközrendszere ezen feladatok Java és PL/SQL nyelveken történő megvalósítását teszi lehetővé. Az XSU olyan Java-osztályok gyűjteménye, amelyek tetszőleges SQL-lekérdezések eredményeit automatikusan és dinamikusan kanonikus XML formátumúvá tudják alakítani. Az XSU tetszőlegesen strukturált felhasználói objektumtípusokon és objektumnézeteken működő lekérdezések esetén is használható. Ezen túlmenően, egy kanonikus XML-dokumentumot automatikusan beszúrhatunk bármely létező táblába, nézetbe, objektumtáblába vagy objektumnézetbe. Az XSLT-transzformációk segítségével elvileg bármely XML-dokumentumot beszúrhatunk az adatbázisba. Az XSU-ról bővebben [19] ír.

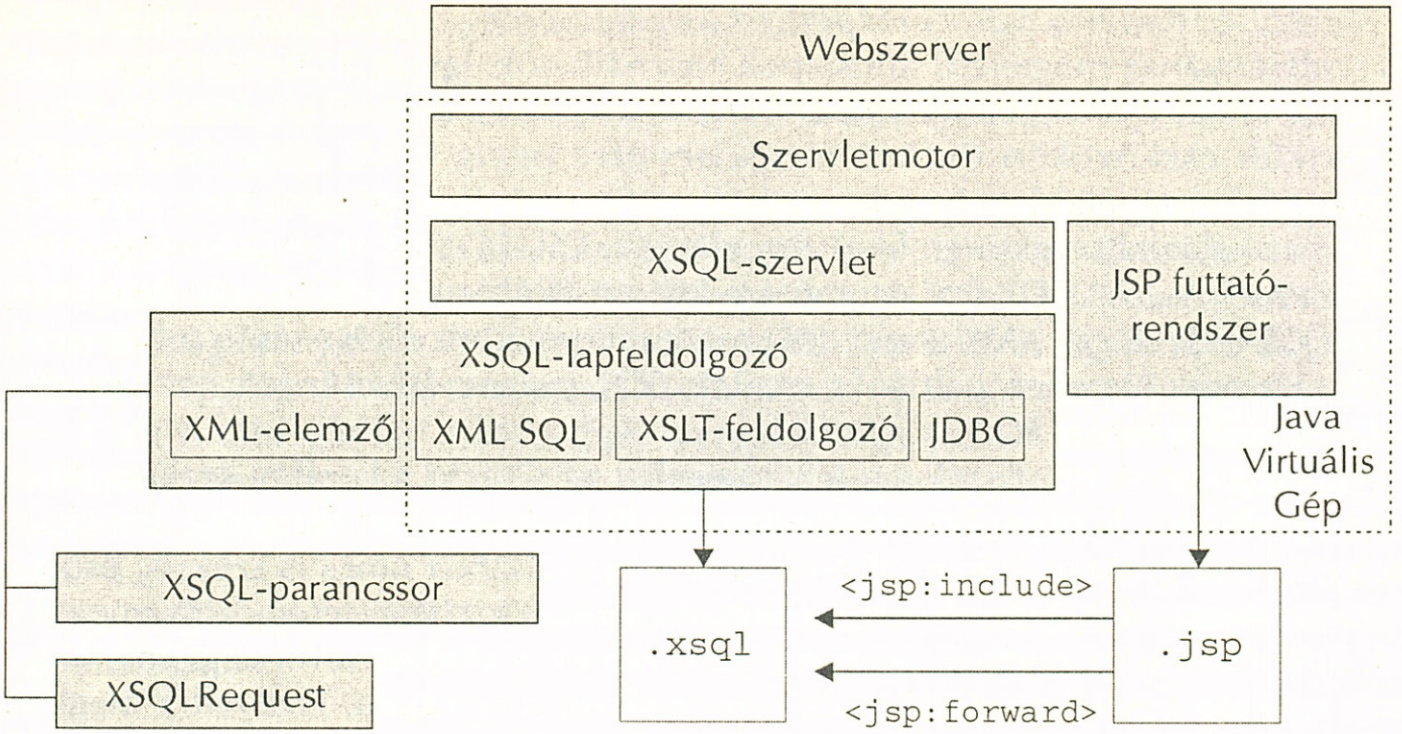
3.3. Tartalomszolgáltatás Oracle XSQL-lapok segítségével

Az Oracle XSQL nevű publikációs keretrendszere elősegíti az XML-információk közzétételét tetszőleges kívánt formában. Az XSQL-lapok az SQL, XML és XSLT erejének kombinálásával nagyban leegyszerűsítik az adatbázisban tárolt információkon alapuló dinamikus webtartalmak közzételét, így nagyszerűen alkalmazható tartalomszolgáltatást megvalósító rendszerek esetén.

Az XSQL-keretrendszert használva az SQL-t ismerő fejlesztők könnyedén létrehozhatnak és használhatnak *XSQL-lapoknak* nevezett „dinamikus sablonokat", amelyek parametrizált SQL-lekérdezéseken alapuló dinamikus XML-adatcsomagok összeállítására, és ezeknek egy hozzákapcsolt XSL-transzformáció segítségével a kívánt XML, HTML, illetve szöveges formátumúra történő átalakítására szolgálnak.

Sem az összeállítás, sem pedig a transzformáció nem igényel programozást. Ezek a tevékenységek deklaratív módon mennek végbe, de természetesen abban az esetben, ha valamelyik beépített tulajdonság nem megfelelő, akkor a keretrendszer egyéni adatforrások integrálásával, illetve az igényelt szerver oldali feldolgozásnak a Javában történő megvalósításával könnyedén kiterjeszthető.

Az XSQL-keretrendszer a közzétételre váró összegyűjtött információt különválasztja annak megjelenítésétől, ami több előnnyel is jár. Ily módon ugyanazt az információt



3.2. ábra. *Az XSQL-keretrendszer felépítése*

sokféleképpen meg lehet jeleníteni, beleértve a megjelenítésnek a megjelenítést végző eszközre (böngésző, mobiltelefon, PDA stb.) szabását is. A megjelenítés a bemutatandó tartalomtól teljesen függetlenül átlakítható, továbbfejleszthető, csinosítgatható. Ezenfelül meglévő lapok új lapokba történő bevonásával az információ újrafelhasználása is lehetővé válik.

A szerver oldalon elhelyezkedő XSQL-lapsablonok (`.xsql` kiterjesztésű állományok) segítségével tetszőleges információt tetszőleges formátumban és tetszőleges eszközön jeleníthetünk meg. Az XSQL-feldolgozómotor értelmezi és feldolgozza az XSQL-lap tartalmát. Az XSQL-feldolgozómotor használatára – mint ahogyan az a 3.2. ábrán is látható – négy lehetőség kínálkozik. Használható parancssorból, illetve kötegelt módon, az XSQL parancssori segédeszköz segítségével; a kedvenc webszerverünkbe épített XSQL-szervlet segítségével weben keresztül; egy JSP-alkalmazás részeként (egy `<jsp:include>` utasítás segítségével egy lapsablont ágyazhatunk a JSP-oldalba); illetve egy Java API-n keresztül az `XSQLRequest` objektum segítségével.

Egy XSQL-lapsablon a fenti forgatókönyvek bármelyikében, vagy akár mindegyikében is felhasználható. Az XSQL-feldolgozás menete – függetlenül attól, hogy milyen módon használjuk – a következő:

1. A feldolgozómotorhoz beérkezik egy XSQL-sablon feldolgozására vonatkozó kérelem.
2. A feldolgozómotor egy vagy több SQL-lekérdezés segítségével összeállít egy XML-adatcsomagot.
3. Ezt az adatcsomagot visszaadja a kérelmezőnek.
4. Szükség esetén a feldolgozómotor az adatcsomagot XML, HTML vagy szöveges formába alakíthatja.

Az utolsó (transzformációs) lépésben az adatcsomag átalakítására a W3C XSLT 1.0 szabványának megfelelő stíluslapok használhatók. Így az XML-adatcsomagból könnyen előállíthatunk böngészőben történő megjelenítésre HTML formájú, vezeték nélküli eszközökön történő megjelenítésre pedig WML (Wireless Markup Language) formájú kódot. Az adatvezérelt grafikonok és diagramok megjelenítését SVG-vel (skálázható vektorgrafika) végezhetjük, XSL-FO (XSL formázóobjektumok) segítségével viszont PDF formátumú eredményt állíthatunk elő. A stíluslaptranszformációk tetszőleges XML alapú dokumentumformátum elkészítésén túl használhatók szöveges kimenet (például e-mailek, SQL-parancsállományok, Java-programok stb.) előállítására is.

Az XSQL-lapok mindezeket a lehetőségeket az Oracle a korábbi szakaszokban már tárgyalt XML komponenseinek automatizált használatával biztosítja. Ennek segítségével sok gyakran felmerülő probléma megoldásához nincs is szükség programozásra. A speciális igények kielégítéséhez azonban természetesen programozni is kell, amit ha a keretrendszer beépített eszközeinek a kibővítésével végzünk, akkor nincs szükség egy teljes publikációs keretrendszer megírására. Ehelyett a meglévő keretrendszerbe integrálhatjuk a saját adatforrásainkból XML-adatcsomagokat készítő, illetve az adatcsomagokat a kívánt formába átalakító kódjainkat.

Az XSQL-lapok feldolgozását az Oracle9*i*-be épített XSQL-szervlet végzi. Az XSQL-lapok futtatásához szükség van az Oracle XML-elemzőjére és XSLT-feldolgozójára, valamint az Oracle XML SQL Utilityre (XSU-ra).

A lapok végrehajtásához szükséges adatbázis-kapcsolato(ka)t az `XSQLConfig.xml` nevű konfigurációs állományban kell megadni.

```
<connectiondefs>
   <connection name="pelda">
     <username>kovacs</username>
     <password>kj01ps</password>
     <dburl>jdbc:oracle:thin:@localhost:1521:tesztAB</dburl>
     <driver>oracle.jdbc.driver.OracleDriver</driver>
     <autocommit>true</autocommit>
   </connection>
   <connection name="lite">
     <username>system</username>
     <password>manager</password>
     <dburl>jdbc:Polite:POlite</dburl>
     <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
    </connection>
</connectiondefs>
```

Minden kapcsolat esetén meg kell adni a csatlakozáshoz használandó felhasználónevet és jelszót, a JDBC-kapcsolatot leíró kapcsolatsztringet, és a kapcsolódáshoz szükséges JDBC-meghajtó nevét. Opcionálisan engedélyezhetjük, illetve tilthatjuk az automatikus jóváhagyást is (ha ez elmarad, akkor az XSQL-feldolgozómotor a JDBC-meghajtó alapértelmezését veszi át). A kapcsolatokat leíró konfigurációs állományban tetszőleges számú kapcsolat definiálható, és később, egy XSQL-lap létrehozásakor az itt megadott kapcsolatnevek közül kell egyet megadnunk az XSQL-lap doku-

mentumeleme `connection` attribútumának értékeként, mint ahogyan az a 3.3.1. alfejezet példáiban is látható.

A futtatáshoz szükséges további beállításokról az Olvasó az XSQL dokumentációjában [19] olvashat.

XSQL-lapokat legkönnyebben az Oracle JDeveloper nevű integrált fejlesztőkörnyezete segítségével hozhatunk létre. A JDeveloper a 3.1 verziótól kezdve támogatja a szintaktikai elemek színekkel történő kiemelését, az XML-dokumentumok ellenőrzését, és megkönnyíti az XSQL-lapok tesztelését. Ezenfelül az újabb JDeveloper-verziók az XSQL-lapok hibakeresését és XSQL-akciók varázslók segítségével történő létrehozását is támogatják. A könyvben szereplő XSQL-példák mindegyikét Oracle9*i* JDeveloper segítségével készítettük és hajtottuk végre. Az XSQL-lapok létrehozásának, futtatásának és hibakeresésének mikéntjéről az Olvasó [19]-ben olvashat bővebben.

Egy XSQL-lapot az XSQL parancssori segédprogram segítségével parancssorból is végrehajthatunk:

```
$ java oracle.xml.xsql.XSQLCommandLine xsql_lap [kimenő_állomány_neve]
[paraméter=érték]...
```

Ha a kimenő állomány nevét megadjuk, akkor az `xsql_lap` feldolgozásának eredménye ebbe az állományba kerül, különben pedig a szabványos kimeneti csatornára. Tetszőleges számú paramétert átadhatunk a feldolgozómotornak, de van néhány előredefiniált viselkedéssel rendelkező paraméter is, amelyekről részletesen [19] ír.

3.3.1. XML-adatcsomagok létrehozása SQL-lekérdezésekből

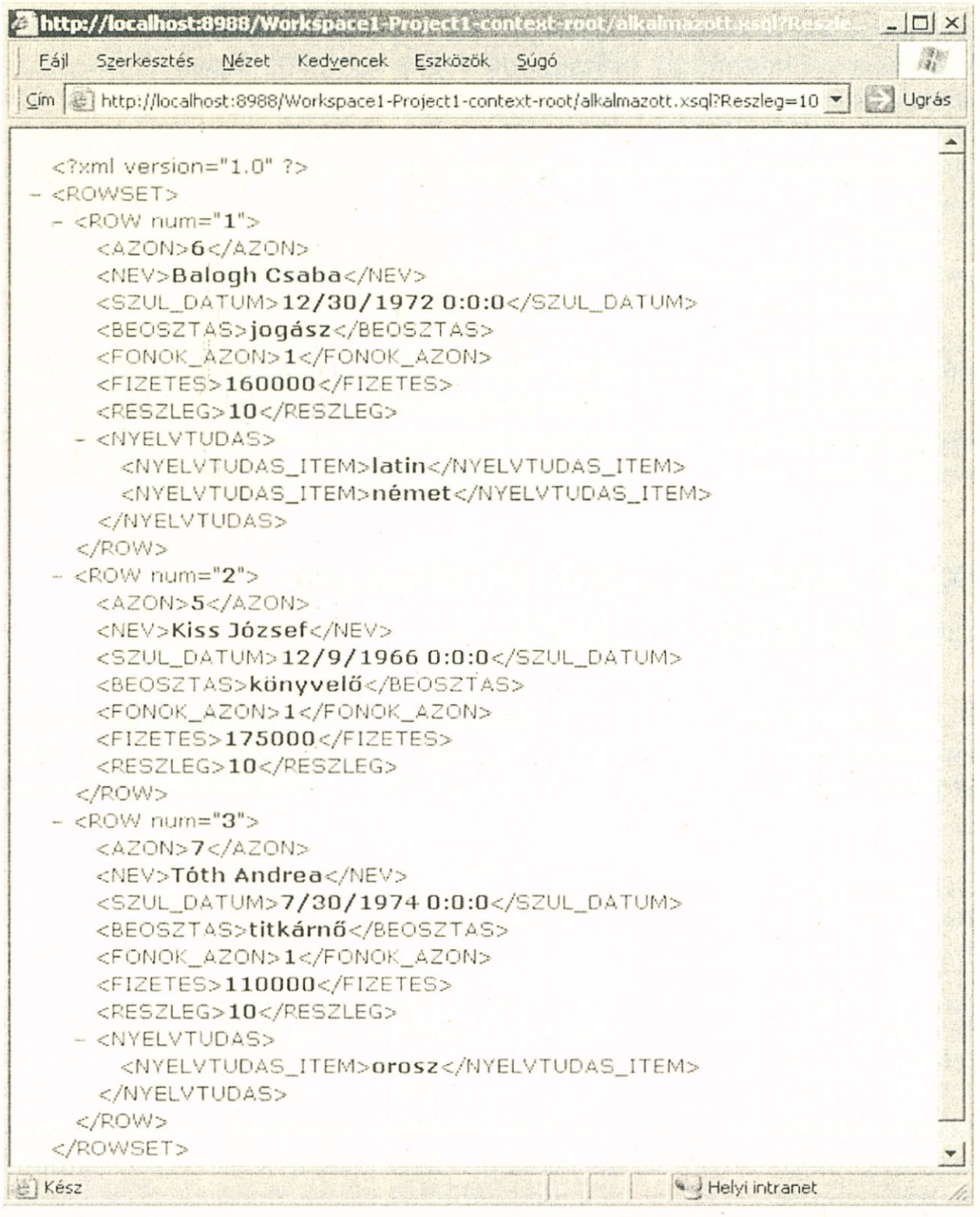
XSQL-lapok segítségével az adatbázisban tárolt információkat könnyedén megjeleníthetjük a weben. Például az egy adott részlegen dolgozó alkalmazottak adataiból valósidőben összállíthatunk egy XML-adatcsomagot az alábbi lapsablon segítségével:

```
<?xml version="1.0"?>
<xsql:query connection="pelda" bind-params="Reszleg"
xmlns:xsql="urn:oracle-xsql">
    SELECT * FROM alkalmazott
     WHERE reszleg = ?    /* A ? helyére a Reszleg paraméter */
     ORDER BY nev         /* értéke helyettesítődik majd be. */
</xsql:query>
```

Amennyiben az XSQL-szervletet megfelelően beállítottuk webszerverünkön, a fenti `alkalmazott.xsql` állományt csupán be kell másolnunk a webszerver virtuális könyvtárhierarchiájába, és a sablont azonnal elérhetjük egy webböngésző segítségével, például a következő URL-ről:

```
http://localhost:8988/Workspace1-Project1-context-root/
alkalmazott.xsql?Reszleg=10
```

Az XSQL-lapba ágyazott lekérdezés eredményeit az XSQL-motor automatikusan XML-adatcsomag formájában adja meg. Ezt általában egy másik (például egy XSL-transzformációt végző) szerver oldali programnak adjuk tovább, de egy XML-dokumentumot megjeleníteni képes böngésző (például Internet Explorer 5.0) közvetlenül is képes megjeleníteni az eredményt, mint ahogyan az a 3.3. ábrán látható.



3.3. ábra. *Az* `alkalmazott.xsql` *XSQL-lap eredménye XML-ben*

Vizsgáljuk meg közelebbről az XSQL-lapsablon felépítését! A lap egy XML-deklarációval kezdődik:

```
<?xml version="1.0"?>
```

mivel ez egy olyan, `.xsql` kiterjesztéssel rendelkező XML-állomány, amely a statikus XML-tartalmak és az XSQL-„akcióelemek" keverékét tartalmazza. A fenti példa nem tartalmaz statikus XML-elemeket, csupán egyetlen XSQL-akcióelemet, az `<xsql:query>`-t, így ez a lehető legegyszerűbben használható XSQL-lap, mivel csupán egyetlen lekérdezést tartalmaz.

A példa dokumentumelemében (`<xsql:query>`) az utolsó attribútum (`xmlns:xsql`) az `xsql` névtér előtagot deklarálja, amely így a továbbiakban rövid névként használható az XSQL `urn:oracle-xsql` nevű névtér-azonosítója helyett.

Az XSQL-állomány dokumentumeleme tartalmaz egy `connection` attribútumot is, amely az adatbázis-kapcsolat azonosítására szolgál. Az attribútum az `XSQLConfig.xml` konfigurációs állományban megadott előredefiniált kapcsolatok nevei közül veheti fel értékét. A fenti példában ez `pelda`. A kapcsolatra vonatkozó információkat – a felhasználónevet, a jelszót, az adatbázis és a kapcsolódáshoz használandó JDBC-meghajtó nevét – a fenti konfigurációs állományban, központosítva kell megadni.

Az XSQL-állományban megadott lekérdezések – amelyek az XML-adatcsomagok előállításának alapját képezik – paraméterezhetők. A lekérdezés szövegében ezeket a paramétereket kérdőjelek jelölik. Az XSQL-szerverlap futtatásakor ezeket a paramétereket a szerverlap nevétől egy kérdőjel, a paramétereket egymástól pedig egy `&` jel választja el. Az `<xsql:query>` elem `bind-params` attribútuma a paraméterek nevének megadására szolgál. Példánkban egyetlen paraméter szerepel, amelynek neve `Reszleg`. A végrehajtáskor a paraméter aktuális értékét meg kell adnunk:

```
http://localhost:8988/Workspace1-Project1-context-root/
alkalmazott.xsql?Reszleg=10
```

Itt hívjuk fel a figyelmet, hogy a paraméterek neveiben a kis- és nagybetűk különbözőnek számítanak!

Ha a lekérdezésnek több paramétere van, akkor az `<xsql:query>` elem `bind-params` attribútumának értéke a paraméterek szóközzel elválasztott neve. Ezek a paraméterek a felírás sorrendjében rendelődnek a lekérdezés szövegében szereplő kérdőjelekhez. Egy adott részlegen dolgozók közül csak a bizonyos összegnél többet kereső alkalmazottak lekérdezését a következő XSQL-lap segítségével végezhetjük:

```
<?xml version="1.0"?>
<xsql:query connection="pelda" bind-params="Reszleg Fizetes"
xmlns:xsql="urn:oracle-xsql">
    SELECT * FROM alkalmazott
     WHERE reszleg = ?   /* A reszleg paraméter */
       AND fizetes > ?   /* A fizetes paraméter */
     ORDER BY nev
</xsql:query>
```

A paraméterek aktuális értékeinek megadása a következő módon történhet:

```
http://localhost:8988/Workspace1-Project1-context-root/
alkalmazott.xsql?Reszleg=10&Fizetes=140000
```

Amennyiben egy lapon egynél több kérdést szeretnénk elhelyezni, egy új elemet kell bevezetnünk, amely körülöleli a többi elemet. Az új elem neve tetszőleges szabályos XML-elemnév lehet (a következő példában ez `lap`), és ez az elem – mint statikus XML-tartalom – az eredményül kapott dokumentumban is megjelenik.

```
<?xml version="1.0"?>
<lap connection="pelda" xmlns:xsql="urn:oracle-xsql">
  <xsql:query bind-params="Reszleg">
    SELECT * FROM alkalmazott
     WHERE reszleg = ?
     ORDER BY nev
  </xsql:query>
  <!-- Ide jöhetnek a további xsql:query akciók. -->
</lap>
```

Felhívjuk a figyelmet, hogy – amint az a példából is kitűnik – a névtér deklarációja és a `connection` attribútum mindig a dokumentumelemhez tartozik, míg a `bind-params` attribútum az `<xsql:query>` elemhez kapcsolódik.

Egy ún. *lexikális helyettesítési paraméter* segítségével bármely XSQL-akcióelembe behelyettesíthető bármely (lap vagy munkamenet szintű) paraméter értéke. Ez lehetővé teszi az akciók viselkedésének parametrizálását, valamint az általuk végrehajtott SQL-utasítások bizonyos részeinek behelyettesítését. A lexikális helyettesítési paraméterek megadási módja `{@`*paraméternév*`}`.

A következő példa két lexikális helyettesítési paraméter használatát mutatja be. Az egyik lehetővé teszi, hogy a maximálisan lekérdezendő sorok számát paraméterként adjuk meg, míg a másik az ORDER BY utasításrészben szereplő oszlopnevek listáját szabályozza. Ha `max` és `rendezesi_szempont` lap vagy munkamenet szintű paraméter, akkor értékük behelyettesítődik a lexikális helyettesítési paraméter hivatkozásának helyére. Ha például a `max` és `rendezesi_szempont` lap szintű paraméterek értéke rendre 3 és `nev`, akkor a

```
<xsql:query max-rows="{@max}" bind-params="Reszleg">
  SELECT nev, projektnev, oraszam
    FROM alkalmazott a, projekt p, projekten_dolgozik d
   WHERE a.azon = d.alkalmazott_azon AND p.projektkod = d.projekt_azon
     AND a.reszleg = ?
     AND oraszam > 30
  ORDER BY {@rendezesi_szempont}
</xsql:query>
```

akcióelem megadja, hogy kik azok az alkalmazottak, akik egy projekten 30 óránál többet dolgoznak, és melyik az a projekt. Mindezt úgy adja meg, hogy legfeljebb 3 sor kerül lekérésre, és azok az alkalmazottak nevei alapján lesznek rendezve.

A lexikális helyettesítési paraméterek segítségével lehetőségünk nyílik az adatbázis-kapcsolat és az alkalmazandó stíluslap paraméterezésére is:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="{@stilus}.xsl"?>
<xsql:query connection="{@kapcsolat}" xmlns:xsql="urn:oracle-xsql" max-
rows="{@max}" bind-params="Reszleg">
  SELECT nev, projektnev, oraszam
    FROM alkalmazott a, projekt p, projekten_dolgozik d
   WHERE a.azon = d.alkalmazott_azon AND p.projektkod = d.projekt_azon
     AND a.reszleg = ?
     AND oraszam > 30
  ORDER BY {@rendezesi_szempont}
</xsql:query>
```

Gyakran szükségessé válhat, hogy a paraméterváltozóknak és a helyettesítési paramétereknek közvetlenül a lapon alapértelmezett értéket tudjuk adni, mivel így híváskor esetleg kevesebb paramétert kell explicit módon megadni.

Egy paraméterhez nagyon egyszerűen alapértelmezett értéket tudunk rendelni. Nem kell mást tennünk, mint egy, a paraméterrel megegyező nevű XML-attribútumot megadnunk magában az akcióelemben vagy annak valamely ősében. Amennyiben egy kérésben valamely paraméternek nem adunk értéket, az XSQL-feldolgozó megnézi, hogy az aktuális akcióelem rendelkezik-e ugyanilyen nevű attribútummal. Ha igen, akkor az itt megadott érték lesz a paraméter értéke, ellenkező esetben pedig a feldolgozó megnézi, hogy a tartalmazó (ős-) elemnek van-e ilyen nevű attribútuma, és így tovább, egészen addig, amíg el nem ér a lap dokumentumeleméig. Ha itt sem talál azonos nevű attribútumot, akkor a lekérdezés nem ad vissza egyetlen sort sem, így az eredményül kapott XML-dokumentum csupán egy (alapértelmezés szerint `<ROWSET/>` nevű) üres elemet tartalmaz. Lényeges, hogy amennyiben a kérésben értéket adunk a paraméternek, az minden alapértelmezést felüldefiniál.

Paraméterfajták

Az XSQL-lapok mind a kérésben megadott, mind pedig a lap akciói által meghatározott lap szintű paramétereket használhatják. Ha egy akció egy `param` nevű paraméterre történő hivatkozással találkozik, a paraméter értéke a következő lesz:

Ha létezik `param` nevű lap szintű paraméter, akkor annak értéke, különben:
ha a kérésben meg van adva a `param` nevű paraméter, akkor az ott megadott érték, különben:
ha az aktuális akcióelemben létezik alapértelmezett értéke a paraméternek, akkor ez az érték, ha nem, akkor az aktuális elem legközelebbi olyan őse `param` nevű paraméterének értéke, ahol ez meg van adva, különben:
a lexikális helyettesítési paraméterek értéke üres sztring, minden más paraméteré pedig NULL-érték.

A HTTP feletti XSQL-szervlet által feldolgozott XSQL-lapok további két, HTTP-specifikus paraméterrel rendelkeznek, a munkamenet szintű változókkal és a cookie-val. Az XSQL-szervlet által feldolgozott XSQL-lapok esetén a paraméterérték-meghatározás folyamata a következőképpen bővül ki:

Ha létezik `param` nevű lap szintű paraméter, akkor annak értéke, különben:
ha be van állítva a `param` nevű cookie, akkor annak értéke, különben:
ha meg van adva a `param` nevű munkamenet szintű változó értéke, akkor ez az érték, különben:
ha a kérésben meg van adva a `param` nevű paraméter, akkor az ott megadott érték, különben:
ha az aktuális akcióelemben létezik alapértelmezett értéke a paraméternek, akkor ez az érték, ha nem, akkor az aktuális elem legközelebbi olyan őse `param` nevű paraméterének értéke, ahol ez meg van adva, különben:
a lexikális helyettesítési paraméterek értéke üres sztring, minden más paraméteré pedig NULL-érték.

Az értékmeghatározás ezen sorrendje biztosítja, hogy a felhasználók nem definiálhatják felül sem a webszerver által kezelt HTTP munkamenet-változók értékét – amelyek addig élnek, amíg a HTTP-munkamenet tart –, sem pedig a cookie-kban tárolt értékeket, amelyek a böngésző munkamenetei között is élnek.

3.3.2. Az XML-adatcsomagok transzformációja

Ha a 3.3. ábrán látható – `<ROWSET>` és `<ROW>` elemek által közrefogott – kanonikus XML-kimenet nem megfelelő számunkra, az XSQL-lapsablonhoz egy XSLT-stíluslaptranszformációt kapcsolhatunk, amely az előállított XML-adatcsomagot még a szerver oldalon a kívánt formára alakítja.

Az alkalmazások közötti adatcsere során elsőként az üzletfelek között mozgó adatok szerkezetét leíró dokumentumsémát kell megadni, amely definiálja, hogy a dokumentum milyen elemeket és attribútumokat tartalmazhat. A példánkban szereplő repülőtéri rendszer adatainak dokumentumtípus-definícióját egy `alkalmazottlista.dtd` nevű állomány tartalmazza. A DTD felépítése a következő: az `<ALKALMAZOTTLISTA>` dokumentumelem legalább egy `<ALKALMAZOTT>` elemet tartalmaz, amelynek két attribútuma `AZON` és `fizetes`. Az `<ALKALMAZOTT>` elemek egy-egy `<NEV>`, illetve `<BEOSZTAS>` elemet tartalmaznak.

Az alábbi `alkalmazott.xsl` stíluslaptranszformáció XSQL-laphoz kapcsolásával az alkalmazottakat leíró dokumentum alapértelmezett `<ROWSET>` és `<ROW>` elemeit a kommunikáció közös nyelvét leíró DTD-nek megfelelő formátumra alakíthatjuk.

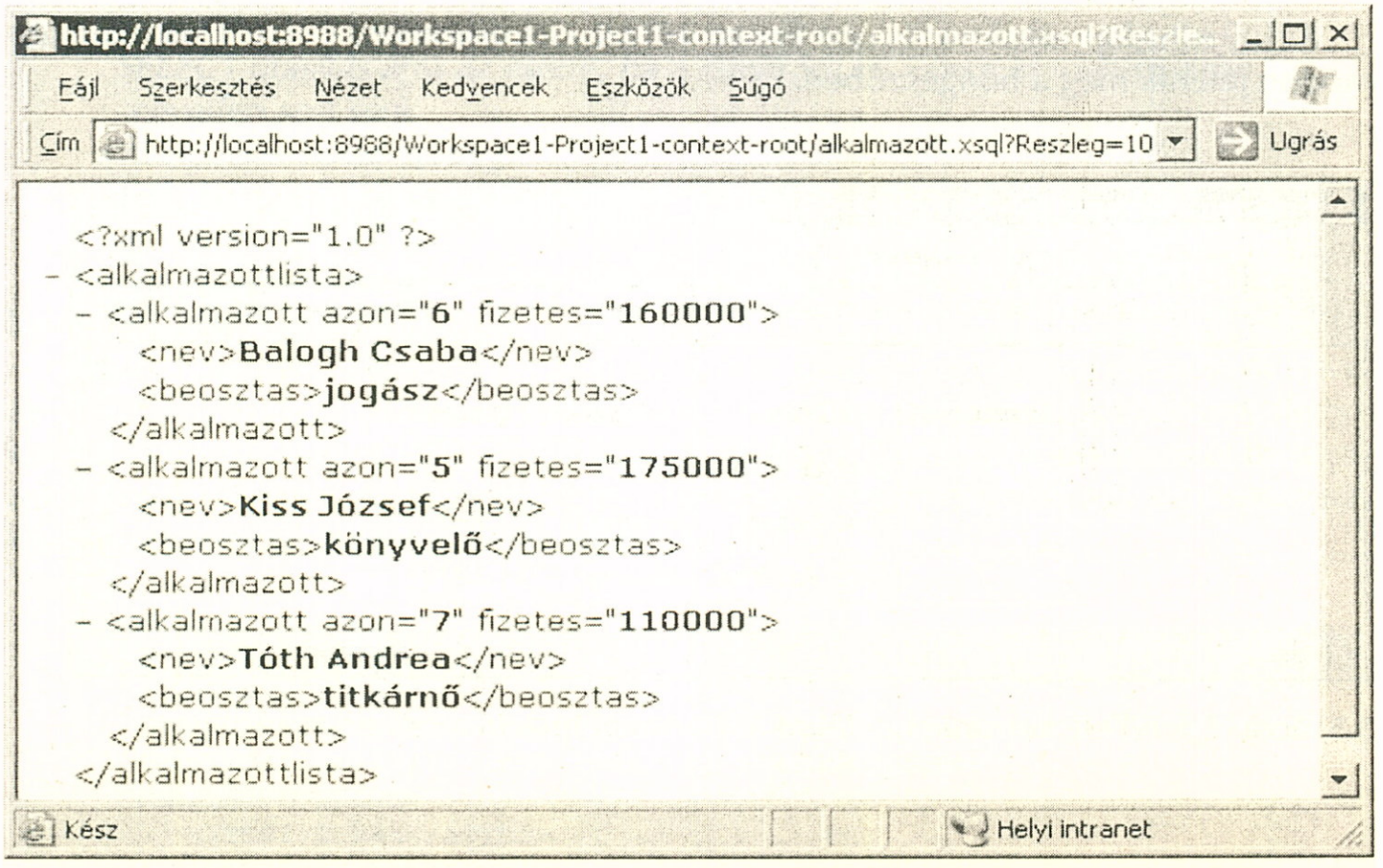
```
<!-- A ROWSET/ROW formájú eredményeket alkalmazottlista formátumúvá
alakító XSLT -->
```

```
<alkalmazottlista xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">
  <xsl:for-each select="ROWSET/ROW">
    <alkalmazott azon="{AZON}" fizetes="{FIZETES}">
      <nev><xsl:value-of select="NEV"/></nev>
      <beosztas><xsl:value-of select="BEOSZTAS"/></beosztas>
    </alkalmazott>
  </xsl:for-each>
</alkalmazottlista>
```

A stíluslaptranszformációt egy `<?xml-stylesheet?>` feldolgozási utasítás segítségével kapcsolhatjuk az XSQL-laphoz:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="alkalmazott.xsl"?>
<xsql:query connection="pelda" bind-params="Reszleg"
xmlns:xsql="urn:oracle-xsql">
    SELECT * FROM alkalmazott
     WHERE reszleg = ?
     ORDER BY nev
</xsql:query>
```

A stíluslaptranszformációt az XSQL-lap által előállított XML-dokumentumra alkalmazva a 3.4. ábrán látható eredményhez jutunk. Ez a dokumentum megfelel az `alkalmazottlista.dtd`-nek.

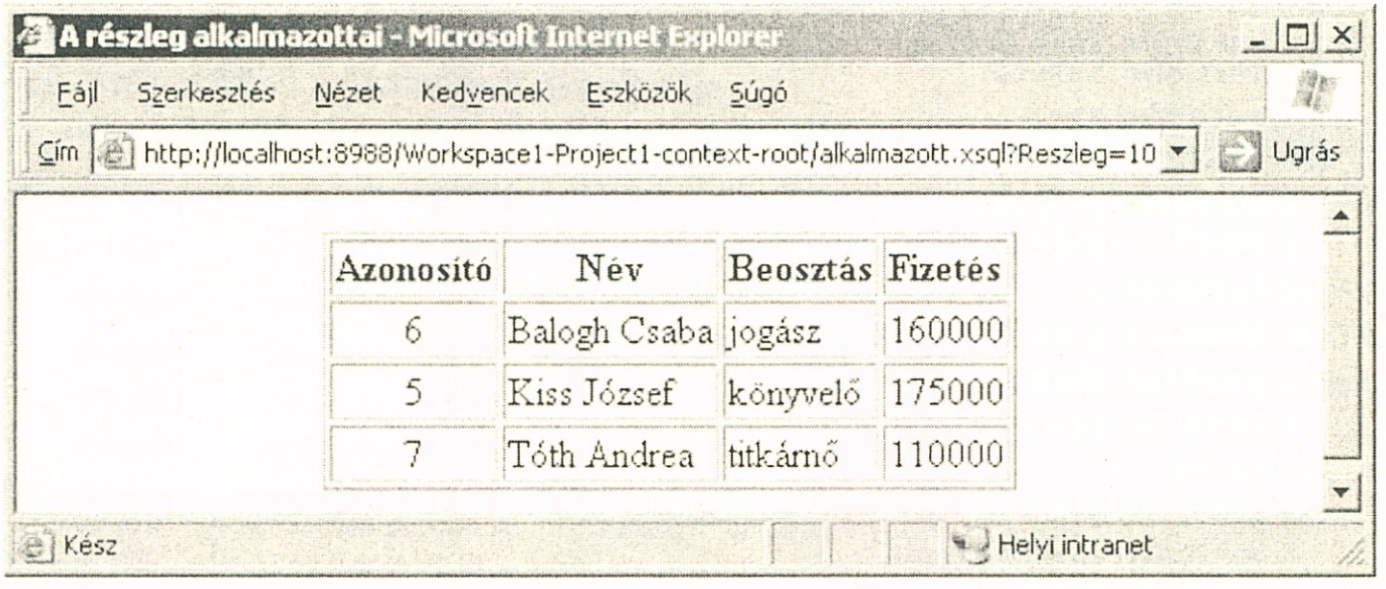


3.4. ábra. *Az XSQL-lap eredménye az „ipari szabványt" leíró formátumban*

Ugyanazon XML-információtartalmat nemcsak XML, hanem HTML formában is megadhatjuk. Ehhez egy újabb stíluslaptranszformációt kell definiálnunk, amely az `alkalmazottlista.dtd`-ben deklarált elemek (például `<ALKALMAZOTT>`, `<BEOSZTAS>`) helyett HTML-elemeket (például `<TABLE>`, `<TR>`, `<TD>`) állít elő.

```
<!-- A ROWSET/ROW formájú eredmények HTML-be alakítása -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">
  <head><title>A részleg alkalmazottai</title></head>
  <body>
    <center>
      <table border="1">
        <tr>
<th>Azonosító</th><th>Név</th><th>Beosztás</th><th>Fizetés</th>
        </tr>
      <xsl:for-each select="ROWSET/ROW">
        <tr>
          <td align="center"><xsl:value-of select="AZON"/></td>
          <td align="left"><xsl:value-of select="NEV"/></td>
          <td align="left"><xsl:value-of select="BEOSZTAS"/></td>
          <td align="right"><xsl:value-of select="FIZETES"/></td>
        </tr>
      </xsl:for-each>
    </table></center>
  </body>
</html>
```

Az új stíluslap alkalmazásával kapott HTML-dokumentum a 3.5. ábrán látható módon jelenik meg a böngészőben.



3.5. ábra. *Egy HTML formátumra alakító XSL-transzformáció eredménye*

3.3.3. Az XSQL-lapok felépítése

Ebben az alfejezetben a lapok készítésekor használható beépített akciókat mutatjuk be.

Az <xsql:query> akció

Az `<xsql:query>` akcióelem végrehajt egy SQL SELECT utasítást, és a lekérdezés eredményhalmazának kanonikus XML formájú reprezentációját beilleszti a lapba. Az akció a lekérdezést az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre. Az akció szintaktikája:

```
<xsql:query>
  SELECT utasítás
</xsql:query>
```

Az `</xsql:query>` elem tartalma tetszőleges szabályos SELECT utasítás lehet. Egy `<xsql:no-rows-query>` elemben megadhatunk egy alternatív lekérdezést, arra az esetre, ha az eredeti lekérdezés nem eredményezne egyetlen sort sem.

```
<xsql:query>
  SELECT utasítás
  <xsql:no-rows-query>
    SELECT utasítás arra az esetre, ha a külső SELECT nem eredményezne
     egyetlen sort sem
  </xsql:no-rows-query>
</xsql:query>
```

Az `<xsql:no-rows-query>` elembe újabb `<xsql:no-rows-query>` elemek ágyazhatók be, tetszőleges mélységig.

Alaphelyzetben, a lekérdezés által előállított XML-dokumentum elemeinek nevei az eredményhalmaz oszlopait tükrözik. Az összetett szerkezetű (objektum, illetve kollekció típusú, vagy CURSOR-kifejezést tartalmazó) oszlopok szerkezetét beágyazott elemek tükrözik. A különféle típusú oszlopokat eredményező, egy sort visszaadó lekérdezés sematikusan a következőképpen néz ki:

```
<ROWSET>
  <ROW id="1">
    <VARCHAROSZLOP>Érték</VARCHAROSZLOP>
    <NUMBEROSZLOP>12345</NUMBEROSZLOP>
    <DATUMOSZLOP>12/10/2001 10:13:22</DATUMOSZLOP>
    <OBJEKTUMOSZLOP>
       <ATTR1>Érték</ATTR1>
       <ATTR2>Érték</ATTR2>
```

```
      </OBJEKTUMOSZLOP>
      <KOLLEKCIO>
         <KOLLEKCIO_ITEM>
           <ATTR1>Érték</ATTR1>
           <ATTR2>Érték</ATTR2>
         </KOLLEKCIO_ITEM>
         <KOLLEKCIO_ITEM>
           <ATTR1>Érték</ATTR1>
           <ATTR2>Érték</ATTR2>
         </KOLLEKCIO_ITEM>
      </KOLLEKCIO>
      <KURZOR>
        <KURZOR_ROW>
          <OSZLOP1>Érték1</OSZLOP1>
          <OSZLOP2>Érték2</OSZLOP2>
        </KURZOR_ROW>
      </KURZOR>
    </ROW>
  </ROWSET>
```

Az eredményhalmaz minden sora egy `<ROW>` elem tartalmaként jelenik meg. A lekérdezésekben másodlagos neveket megadva a létrejövő XML-elemek nevei megváltoztathatók. A másodlagos nevek megadása kötelező abban az esetben, ha az oszlop eredeti neve nem lehet egy XML-elem neve. Például az

```
<xsql:query>SELECT 2 * fizetes FROM alkalmazott</xsql:query>
```

akció hibát eredményez, mert a kifejezés alapértelmezett oszlopneve (`2 * fizetes`) nem lehet egy XML-elem neve, mivel a `*` karakter nem szerepelhet XML-elemnévben. Ezt a problémát egy másodlagos név megadásával küszöbölhetjük ki:

```
<xsql:query>SELECT 2 * fizetes dupla_fizetes FROM alkalmazott</xsql:query>
```

Az `<xsql:query>` akció által generált XML-dokumentumra vonatkozó beállításokra szolgáló attribútumokat a 3.6. táblázat foglalja össze.

3.6. táblázat. *Az* `<xsql:query>` *akcióelem attribútumai*

Attribútumnév	*Leírás*
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`date-format = "sztring"`	A dátum típusú oszlopok, illetve objektumattribútumok lekérdezésekor ezen attribútummal adhatjuk meg, hogy milyen formátumban kerüljenek az adatok az XML-dokumentumba. Az érvényes formátummaszkok leírása a `java.text.SimpleDateFormat` osztály dokumentációjában található meg.

3.6. táblázat. *Az* `<xsql:query>` *akcióelem attribútumai (folytatás)*

Attribútumnév	*Leírás*
`error-statement = "logikai"`	A paraméter lehetséges értékei `yes` és `no`. Az alapértelmezés szerinti `yes` érték esetén, ha az SQL-utasítás hibás, a hibát leíró `<xsql-error>` elemben nemcsak a hibaüzenet, hanem a hibát kiváltó SQL-utasítás szövege is megjelenik. Ez utóbbi a paraméter értékének `no`-ra állításával tiltható le.
`fetch-size = "egész"`	Az egy adatbázishoz fordulás során lehívandó rekordok száma. Ha nem adjuk meg, akkor az `XSQLConfig.xml` konfigurációs állomány `/XSQLConfig/processor/default-fetch-size` beállítása jelenti az alapértelmezett értéket.
`id-attribute = "sztring"`	Alapértelmezés szerint az eredményhalmaz sorai egy `num` nevű attribútum segítségével azonosíthatók. Ennek az attribútumnak a segítségével ezt felülírhatjuk. Amennyiben a paraméter értékét üres sztringre állítjuk, az attribútum nem jelenik meg a kimeneti XML-dokumentumban.
`id-attribute-column = "sztring"`	Annak az oszlopnak a (a kis- és nagybetűkre érzékeny) nevét adhatjuk meg itt, amelynek értékét a lekérdezés egy sorát tartalmazó (alaphelyzetben `<ROW>` nevű) elem azonosító típusú attribútuma felveszi. Az alapértelmezés szerint ez az érték a `rownum` pszeudooszlop értéke.
`include-schema = "logikai"`	A paraméter segítségével előírhatjuk, hogy a kimeneti XML-dokumentum tartalmazzon-e egy, az eredményhalmaz szerkezetét leíró beépített XML Schema definíciót. A paraméter lehetséges értékei `yes` és `no`. Az utóbbi az alapértelmezés.
`max-rows = "egész"`	A maximálisan lehívandó sorok száma, miután az esetlegesen a `skip-rows` attribútumban megadott számú sor átugrásra került. Ha nincs megadva, akkor az eredményhalmaz összes sora lehívásra kerül.
`null-indicator = "logikai"`	Azt jelzi, hogy egy oszlop esetleges NULL értéke legyen-e jelölve az oszlopnak megfelelő elem `NULL="Y"` attribútumaként. A paraméter lehetséges értékei `yes` és `no`. Az alapértelmezés a `no`, ami azt jelenti, hogy a NULL értéket tartalmazó oszlopoknak megfelelő elemek meg sem jelennek az XML-dokumentumban.
`row-element = "sztring"`	A lekérdezés egy sorát tartalmazó – alapértelmezés szerint `<ROW>` nevű – elem neve írható felül az atrribútum segítségével. Ha az elemet egyáltalán nem szeretnénk megjeleníteni, akkor a paraméter értékét állítsuk üres sztringre.

3.6. táblázat. *Az* `<xsql:query>` *akcióelem attribútumai (folytatás)*

Attribútumnév	*Leírás*
`rowset-element = "sztring"`	A paraméter segítségével beállítható, hogy mi legyen a teljes eredményhalmazt tartalmazó – alapértelmezés szerint `<ROWSET>` nevű – elem neve. Ha a paraméter értéke üres sztring, akkor ez az elem nem jelenik meg.
`skip-rows = "egész"`	Megadja, hogy hány sort kell a sorok eredményhalmazból történő lehívása előtt átugrani. Ez – a `max-rows` paraméterrel kombinálva – igen hasznos állapot nélküli alkalmazásokban a lekérdezés eredményeinek lapozására.
`tag-case = "sztring"`	Megadja, hogy az XML-elemek nevei kis- vagy nagybetűkkel szerepeljenek. Lehetséges értékei `lower` (csupa kisbetű) és `upper` (csupa nagybetű). Ha nincs megadva, akkor az elemnevek alapértelmezés szerint csupa nagybetűvel kerülnek kiírásra.

Az <xsql:dml> akció

Az `<xsql:dml>` akció segítségével tetszőleges DML- és DDL-utasítást, illetve PL/SQL-blokkot végrehajthatunk. Az akció az utasítást, illetve blokkot az XSQL-lap dokumentelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre. Az akció szintaktikája a következő:

```
<xsql:dml>
  DML-utasítás, DDL-utasítás vagy PL/SQL-blokk
</xsql:dml>
```

A 3.7. táblázat az `<xsql:dml>` akcióelemhez megadható attribútumokat mutatja be.

3.7. táblázat. *Az* `<xsql:dml>` *akcióelem attribútumai*

Attribútumnév	*Leírás*
`commit = "logikai"`	Ezzel a paraméterrel állítható be, hogy a DML-utasítás sikeres végrehajtását követően legyen-e véglegesítés. Lehetséges értékei `yes` és `no`. Az utóbbi az alapértelmezés.
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`error-statement = "logikai"`	A paraméter lehetséges értékei `yes` és `no`. Az alapértelmezés szerinti `yes` érték esetén, ha az SQL-utasítás hibás, a hibát leíró `<xsql-error>` elemben nemcsak a hibaüzenet, hanem a hibát kiváltó SQL-utasítás szövege is megjelenik. Ez utóbbi tiltható le a paraméter értékének `no`-ra állításával.

Az <xsql:ref-cursor-function> akció

Az `<xsql:ref-cursor-function>` akció egy lekérdezés eredményhalmazának egy PL/SQL tárolt függvény végrehajtásával kapott XML formátumú eredményeinek dokumentumba illesztésére szolgál. Az akció a lekérdezést az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A függvény a PL/SQL dinamikus SQL-képességeit kihasználva, dinamikusan, bizonyos feltételektől függően állíthatja össze a lekérdezést. Az XSQL-lapot feldolgozó motor megkapja a lekérdezés eredményhalmazának kurzorkezelőjét. Mint ahogyan arra az akció neve is utal, a függvény visszatérési értéke típusának `REF CURSOR`-nak kell lennie. Az akció szintaktikája:

```
<xsql:ref-cursor-function>
   [sémanév.][csomagnév.]függvénynév(paraméterek);
</xsql:ref-cursor-function>
```

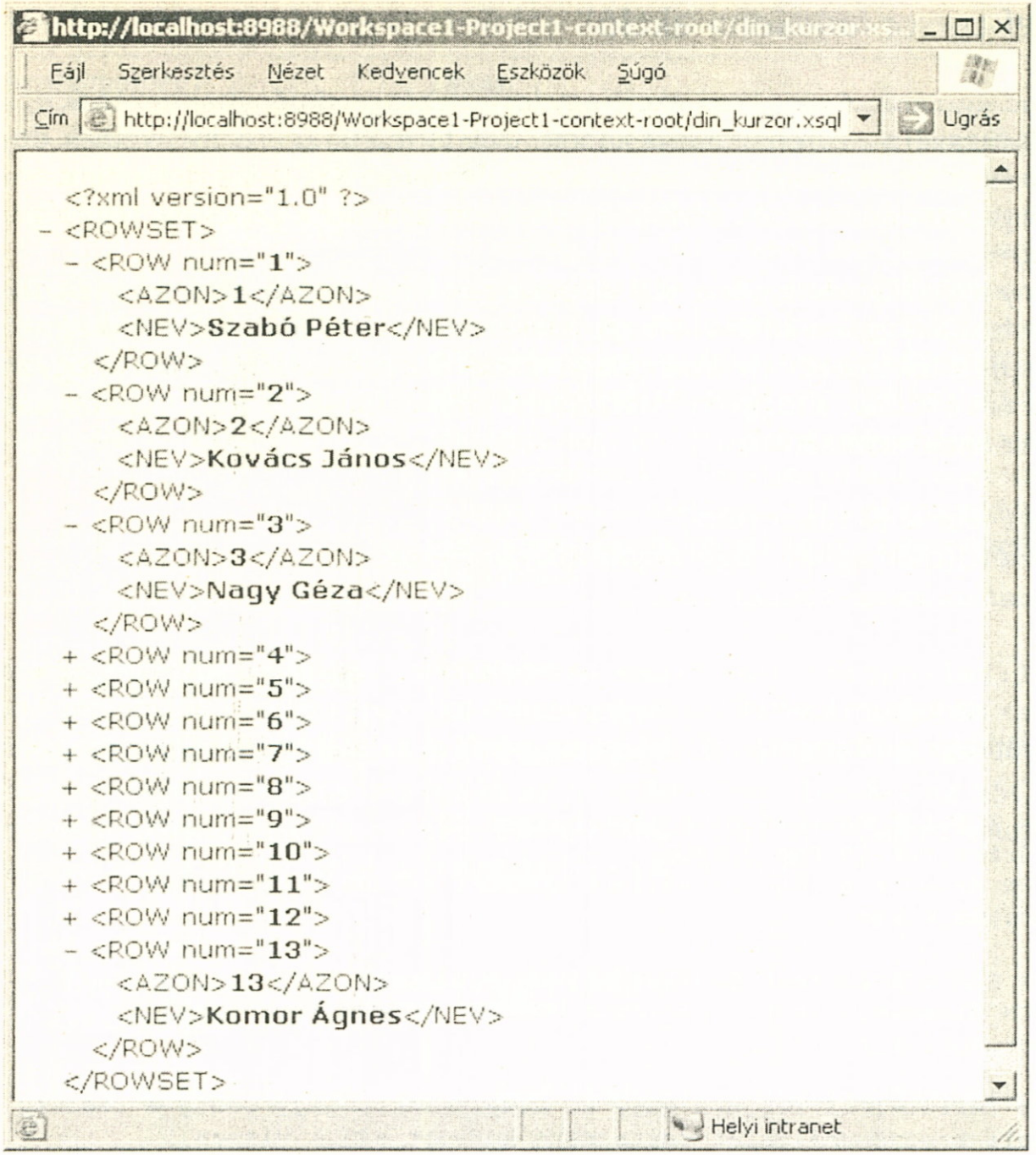
Az `<xsql:ref-cursor-function>` akcióelemhez társítható attribútumok a `<fetch-size>` attribútum kivételével megegyeznek az `<xsql:query>` akcióelem 3.6. táblázatban látható attribútumaival. Tekintsük például az alábbi PL/SQL-csomagot:

```
CREATE OR REPLACE PACKAGE dinamikus_kurzor IS
  TYPE kurzor_ref IS REF CURSOR;
  FUNCTION dinamikus_lekerdezes(azon NUMBER) RETURN kurzor_ref;
END dinamikus_kurzor;
/
CREATE OR REPLACE PACKAGE BODY dinamikus_kurzor IS
  FUNCTION dinamikus_lekerdezes(azon NUMBER) RETURN kurzor_ref IS
    kurzor kurzor_ref;
  BEGIN
   -- Egy feltételtől függően vagy az alkalmazott, vagy pedig a részleg
   -- tábla eredményhalmazát tartalmazó kurzort adjuk vissza.
   IF azon = 1 THEN
     OPEN kurzor FOR 'SELECT azon, nev FROM alkalmazott';
   ELSE
     OPEN kurzor FOR 'SELECT nev, kod FROM reszleg';
   END IF;
   RETURN kurzor;
  END dinamikus_lekerdezes;
END dinamikus_kurzor;
```

Egy `<xsql:ref-cursor-function>` akcióval beilleszthetjük a lapba a függvény által visszaadott `REF CURSOR` dinamikus eredményeit:

```
<xsql:ref-cursor-function>
  dinamikus_kurzor.dinamikus_lekerdezes(1);
</xsql:ref-cursor-function>
```

A 3.6. ábrán a függvény által létrehozott XML-dokumentum látható.



3.6. ábra. *A* `dinamikus_lekerdezes` *függvény által létrehozott XML-dokumentum*

Az <xsql:include-owa> akció

Az `<xsql:include-owa>` akció segítségével egy adatbázisbeli tárolt eljárás által generált XML-tartalom illeszthető be a lapba. Az akció az utasítást, illetve blokkot az XSQL-lap dokumentelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A tárolt eljárás az XML-tag-eket az Oracle Web Agent (OWA) `HTP` and `HTF` csomagjai [23] segítségével egy szerver oldali pufferbe írja, ahonnan majd az XSQL-feldolgozó kiovassa, elemzi és a dinamikusan létrehozott XML-tartalmat a kimenetet jelentő lapra beilleszti. A tárolt eljárásnak egy jólformált XML-lapot kell generálnia, különben hibaüzenetet kapunk.

Az akció szintaktikája a következő:

```
<xsql:include-owa>
    A HTP és/vagy HTF csomagokat használó eljárást hívó PL/SQL-blokk
</xsql:include-owa>
```

Az `<xsql:include-owa>` akcióelemben megadható attribútumokat a 3.8. táblázat foglalja össze.

3.8. táblázat. *Az* `<xsql:include-owa>` *akcióelem attribútumai*

Attribútumnév	*Leírás*
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`error-statement = "logikai"`	A paraméter lehetséges értékei `yes` és `no`. Az alapértelmezés szerinti `yes` érték esetén, ha az SQL-utasítás hibás, a hibát leíró `<xsql-error>` elemben nemcsak a hibaüzenet, hanem a hibát kiváltó SQL-utasítás szövege is megjelenik. Ez utóbbi tiltható le a paraméter értékének `no`-ra állításával.

A 3.5. ábrán látható HTML-dokumentum forráskódját az alábbi tárolt eljárás segítségével hozhatjuk létre:

```
CREATE OR REPLACE PROCEDURE htp_pelda(reszlegkod IN reszleg.kod%TYPE) IS
BEGIN
  htp.print('<html><head><title>A részleg alkalmazottai</title></head>');
  htp.print('<body><center><table border="1">');
  htp.print('<tr><th>Azonosító</th><th>Név</th>');
  htp.print('<th>Beosztás</th><th>Fizetés</th></tr>');
  FOR i IN (SELECT * FROM alkalmazott WHERE reszleg = reszlegkod) LOOP
    htp.tablerowopen;
    htp.tabledata(to_char(i.azon), 'center');
    htp.tabledata(i.nev, 'left');
    htp.tabledata(i.beosztas, 'left');
    htp.tabledata(to_char(i.fizetes), 'right');
    htp.tablerowclose;
  END LOOP;
  htp.print('</table></center></body></html>');
END htp_pelda;
/
```

Ezt az eljárást az XSQL-lapon egy `<xsql:include-owa>` akcióban történő meghívását követően az eljárás által kiírt HTML-dokumentum beillesztésre kerül a kimenő dokumentumba.

```
<xsql:include-owa bind-params="Reszleg" xmlns:xsql="urn:oracle-xsql"
connection="pelda">
  BEGIN htp_pelda(?); END;
</xsql:include-owa>
```

Az <xsql:include-request-params> akció

Az `<xsql:include-request-params>` akció az adatcsomagba illeszti a kérésben megadott paraméterek XML-reprezentációját, ami különösen akkor lehet hasznos, ha a dokumentumhoz kapcsolt stíluslapban egy XPath-kifejezés segítségével hozzá szeretnénk férni a kérés paramétereihez. Az akció szintaktikája:

```
<xsql:include-request-params/>
```

Az XML-adat a következő formában kerül beillesztésre:

```
<request>
  <parameters>
    <paraméternév1>érték1</paraméternév1>
    <paraméternév2>érték2</paraméternév2>
      ...
  </parameters>
</request>
```

vagy

```
<request>
  <parameters>
    <paraméternév1>érték1</paraméternév1>
    <paraméternév2>érték2</paraméternév2>
      ...
  </parameters>
  <session>
    <munkamenetváltozó-név>érték1</munkamenetváltozó-név>
      ...
  </session>
  <cookies>
    <cookie-név>érték1</cookie-név>
      ...
  </cookies>
</request>
```

abban az esetben, ha a lap feldolgozása XSQL-szervlet segítségével történik. Az akciónak nincsenek attribútumai.

Az <xsql:include-param> akció

Az `<xsql:include-param>` akció egyetlen paraméter XML-reprezentációját az adatcsomagba illeszti, ami különösen akkor lehet hasznos, ha a dokumentumhoz kapcsolt stíluslapban egy XPath-kifejezés segítségével hozzá szeretnénk férni az adott paraméter értékéhez. Az akció szintaktikája:

```
<xsql:include-param name="paraméternév"/>
```

A beillesztett XML-kód a következő alakú lesz:

```
<paraméternév>érték</paraméternév>
```

Az <xsql:include-xml> akció

Az `<xsql:include-xml>` akció egy helyi vagy távoli XML-tartalom adatcsomagba illesztésére szolgál. Az XML formátumú információt tartalmazó erőforrás egy URL segítségével adható meg. Az akció szintaktikája:

```
<xsql:include-xml href="URL"/>
```

Az URL akár abszolút, akár pedig relatív is lehet, így egy másik webszerveren lévő XML-tartalmat ugyanúgy beilleszthetünk, mint egy, az adott gép állományrendszerében elhelyezkedőt. A `href` attribútum megadása kötelező, az akció semmilyen más paraméterrel nem rendelkezik.

Az <xsql:set-page-param> akció

Az `<xsql:set-page-param>` akció egy lap szintű paraméter értékének beállítására szolgál. A paraméter értéke statikus szöveg és egyéb paraméterértékek (például egy SELECT utasítás eredményének) kombinációjaként is előállhat.

Az akció szintaktikája:

```
<xsql:set-page-param name="paraméternév" value="érték"/>
```

vagy

```
<xsql:set-page-param name="paraméternév">
  SELECT utasítás
</xsql:set-page-param>
```

Ha a második (a SELECT-et tartalmazó) megadási módot használjuk, akkor az eredményhalmaz legelső sorának első oszlopában található érték rendelődik a változó-

hoz. Az akció az utasítást, illetve blokkot az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A `name` attribútum megadása kötelező, a `value` attribútum és a SELECT utasítás pedig kölcsönösen kizárja egymást, azaz, ha az egyiket megadtuk, a másik nem szerepelhet.

Az akció attribútumait a 3.9. táblázat foglalja össze.

3.9. táblázat. *Az* `<xsql:set-page-param>` *akció attribútumai*

Attribútumnév	*Leírás*
`name = "sztring"`	A beállítandó lap szintű változó neve.
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`ignore-empty-value = "logikai"`	Beállíthatjuk, hogy figyelmen kívül kell-e hagyni a lap szintű paraméterhozzárendelést abban az esetben, ha a hozzárendelt érték üres sztring. A lehetséges értékek `yes` és `no`. Az alapértelmezés `no`.

Az <xsql:set-session-param> akció

Az `<xsql:set-session-param>` akció egy HTTP-munkamenet szintű paraméter értékének beállítására szolgál. A munkamenet szintű paraméter értéke mindaddig megmarad, míg a böngésző HTTP-munkamenete tart. Ezt a webszerver felügyeli. A paraméter értéke statikus szöveg és egyéb paraméterértékek (például egy SELECT utasítás eredményének) kombinációjaként is előállhat.

Mivel ez egy HTTP-protokoll specifikus tulajdonság, ezt az akciót csak akkor van értelme használni, ha azt az XSQL-lapot, amelyen megjelenik, az XSQL-szervlet dolgozza fel. Ha ez az akció egy XSQL-parancssor vagy az `XSQLRequest` API segítségével feldolgozandó XSQL-lapon jelenik meg, akkor ez az akció gyakorlatilag üres utasításként fogható fel. Az akció szintaktikája:

```
<xsql:set-session-param name="paraméternév" value="érték"/>
```

vagy

```
<xsql:set-session-param name="paraméternév">
  SELECT utasítás
</xsql:set-session-param>
```

Ha a második (a SELECT-et tartalmazó) megadási módot használjuk, akkor az eredményhalmaz legelső sorának első oszlopában található érték rendelődik a változóhoz. Az akció az utasítást, illetve blokkot az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A `name` attribútum megadása kötelező, a `value` attribútum és a SELECT utasítás pedig kölcsönösen kizárja egymást, azaz, ha az egyiket megadtuk, a másik nem szerepelhet.

Az akció attribútumait a 3.10. táblázat foglalja össze.

3.10. táblázat. *Az* `<xsql:set-session-param>` *akció attribútumai*

Attribútumnév	*Leírás*
`name = "sztring"`	A beállítandó munkamenet szintű változó neve.
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`ignore-empty-value = "logikai"`	Beállíthatjuk, hogy figyelmen kívül kell-e hagyni a lap szintű paraméterhozzárendelést abban az esetben, ha a hozzárendelt érték üres sztring. A lehetséges értékek `yes` és `no`. Az alapértelmezés `no`.
`only-if-unset = "logikai"`	Ezzel a paraméterrel megadhatjuk, hogy a munkamenet szintű változó értéke csak akkor kerüljön beállításra, ha az még nem létezik. A lehetséges értékek `yes` és `no`. Az utóbbi az alapértelmezés.

Az <xsql:set-cookie> akció

Az `<xsql:set-cookie>` akció egy HTTP-cookie értékének beállítására szolgál. Alapértelmezés szerint a cookie értéke mindaddig megmarad, míg a böngészőablakot be nem zárjuk, de a cookie élettartamát az akcióelem `max-age` attribútumával szabályozhatjuk. A cookie értéke statikus szöveg és egyéb paraméterértékek (például egy SELECT utasítás eredményének) kombinációjaként állhat elő.

Mivel ez egy HTTP-protokoll specifikus tulajdonság, ezt az akciót csak akkor van értelme használni, ha azt az XSQL-lapot, amelyen megjelenik, az XSQL-szervlet dolgozza fel. Ha ez az akció egy XSQL-parancssor vagy az `XSQLRequest` API segítségével feldolgozandó XSQL-lapon jelenik meg, akkor ez az akció gyakorlatilag üres utasításként fogható fel.

Az akció szintaktikája a következő:

```
<xsql:set-cookie name="paraméternév" value="érték"/>
```

vagy

```
<xsql:set-cookie name="paraméternév">
  SELECT utasítás
</xsql:set-cookie>
```

Ha a második (a SELECT-et tartalmazó) megadási módot használjuk, akkor az eredményhalmaz legelső sorának első oszlopában található érték lesz a cookie értéke. Az akció az utasítást, illetve a blokkot az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A `name` attribútum megadása kötelező, a `value` attribútum és a SELECT utasítás pedig kölcsönösen kizárja egymást, azaz ha az egyiket megadtuk, a másik nem szerepelhet.

Az akció attribútumait a 3.11. táblázat sorolja fel.

3.11. táblázat. *Az* `<xsql:set-cookie>` *akció attribútumai*

Attribútumnév	*Leírás*
`name = "sztring"`	A beállítandó cookie neve.
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`domain = "sztring"`	Az a tartomány, amelyben a cookie értéke érvényes és olvasható. Ha nem adjuk meg explicit módon, akkor alapértelmezés szerint a cookie-t létrehozó dokumentumot tartalmazó gép teljes neve (pl.: `szerver.tartomany.hu`).
`ignore-empty-value = "logikai"`	Beállíthatjuk, hogy figyelmen kívül kell-e hagyni a lap szintű paraméterhozzárendelést abban az esetben, ha a hozzárendelt érték üres sztring. A lehetséges értékek `yes` és `no`. Az alapértelmezés `no`.
`max-age = "egész"`	A cookie élettartamának másodpercekben történő megadására szolgál. Alapértelmezés szerint a cookie akkor jár le, amikor a felhasználó bezárja a böngészőablakot.
`only-if-unset = "logikai"`	Ezzel a paraméterrel megadhatjuk, hogy a cookie értéke csak akkor kerüljön beállításra, ha az még nem létezik. A lehetséges értékek `yes` és `no`. Az utóbbi az alapértelmezés.
`path = "sztring"`	Azon tartományon belüli relatív URL elérési út, amelyben a cookie értéke érvényes és olvasható. Ha nem adjuk meg explicit módon, akkor alapértelmezés szerint a cookie-t létrehozó dokumentum URL elérési útja.

Az <xsql:set-stylesheet-param> akció

Az `<xsql:set-stylesheet-param>` akció segítségével egy csúcs szintű XSLT stíluslap paraméter beállítása végezhető. Ez az érték akár kombináltan, azaz statikus szöveg és más paraméterek segítségével, akár pedig egy SELECT utasítás által visszaadott érték alapján adható meg. Az akció szintaktikája:

```
<xsql:set-stylesheet-param name="paraméternév" value="érték"/>
```

vagy

```
<xsql:set-stylesheet-param name="paraméternév">
  SELECT utasítás
</xsql:set-stylesheet-param>
```

Ha a lekérdezéssel dolgozunk, akkor az eredményhalmaz első sorának első oszlopában található érték rendelődik a paraméterhez. Az akció az utasítást az XSQL-lap dokumentumelemében megadott `connection` attribútum által leírt adatbázis-kapcsolat segítségével hajtja végre.

A `name` attribútum megadása kötelező. A `value` attribútum és a tartalmazott SQL-utasítás kölcsönösen kizárja egymást, azaz, ha az egyik meg van adva, a másik nem lehet.

A 3.12. táblázat bemutatja az akció attribútumait.

3.12. táblázat. *Az* `<xsql:set-stylesheet-param>` *akció attribútumai*

Attribútumnév	*Leírás*
`name = "sztring"`	A beállítandó csúcs szintű stíluslap paraméter neve.
`bind-params = "sztring"`	XSQL-paraméternevek szóközökkel elválasztott, rendezett listája. Ezen paraméternevek sorrendi kötéssel kapcsolódnak az SQL-utasításban kérdőjelekkel jelölt paraméterekhez.
`ignore-empty-value = "logikai"`	Beállíthatjuk, hogy figyelmen kívül kell-e hagyni a lap szintű paraméterhozzárendelést abban az esetben, ha a hozzárendelt érték üres sztring. A lehetséges értékek `yes` és `no`. Az alapértelmezés `no`.

Az <xsql:include-xsql> akció

Az `<xsql:include-xsql>` akció segítségével egy XSQL-lap eredményeit könnyen egy másik XSQL-lapba ágyazhatjuk, így megírt lapjaink a későbbiekben újrafelhasználhatóvá válnak.

Példaként tekintsünk egy internetes fórum kategóriáit kilistázó XSQL-lapot:

```
<!-- Kategoriak.xsql -->
<xsql:query connection="forum" xmlns:xsql="urn:oracle-xsql">
  SELECT nev
    FROM kategoria
    ORDER BY nev
</xsql:query>
```

Ezen lap eredményeit beilleszthetjük egy olyan lapba, amely azt a tíz témát listázza ki, amelyhez legutóbb hozzászóltak:

```
<!-- LegutobbiTizTema.xsql -->
<utolso-temak connection="forum" xmlns:xsql="urn:oracle-xsql">
  <temak>
    <xsql:query max-rows="10">
      SELECT cim FROM tema ORDER BY utolso_modositas DESC
    </xsql:query>
  </temak>
  <kategoriak>
    <xsql:include-xsql href="Kategoriak.xsql"/>
  </kategoriak>
</utolso-temak>
```

Az `<xsql:include-xsql>` akció az adatok és a megjelenítés különválasztásának egyik fontos eszköze, segítségével egy létező XSQL-lapot több olyan lapba illeszthetünk be, amelyek csak az alkalmazandó stíluslaptranszformációban különböznek egymástól. Készítsünk két stíluslapot, ezek:

- `lista-html.xsl`, amely a témákat HTML-ben, és
- `lista-wml.xsl`, amely a témákat WML-ben

jeleníti meg.

Így a `Kategoriak.xsql` lap tartalmát (azaz a többféleképpen megjelenítendő adatokat) két új XSQL-lapba illesztve, egyszer a HTML-re, másodszor pedig a WML-re transzformáló stíluslap megadásával különféle eszközökön válnak megjeleníthetővé a kategóriák.

```
<?xml version="1.0"?>
<!-- HTMLKategoriak.xsql -->
<?xml-stylesheet type="text/xsl" href="lista-html.xsl"?>
<xsql:include-xsql href="Kategoriak.xsql" xmlns:xsql="urn:oracle-xsql"/>

<?xml version="1.0"?>
<!-- WMLKategoriak.xsql -->
<?xml-stylesheet type="text/xsl" href="lista-wml.xsl"?>
<xsql:include-xsql href="Kategoriak.xsql" xmlns:xsql="urn:oracle-xsql"/>
```

Ha a beillesztett lap tartalmaz `<?xml-stylesheet?>` feldolgozási utasítást, akkor a stíluslap alkalmazása még az eredmény új lapba történő beillesztése *előtt* megtörténik, így ezek a lapok is újabb lapokba illeszthetők.

Ha egy XSQL-lap tartalmát beillesztjük egy másik lapba, a kéréskor megadott paraméterek a beágyazott lapból is láthatóak. Az XSQL-szervlet által feldolgozott lapok esetén ugyanez a helyzet a munkamenet szintű paraméterekkel és a cookie-kkal is. Nem látja azonban a beágyazott lap a lap szintű paramétereket.

A 3.13. táblázat bemutatja az akció lehetséges attribútumait.

3.13. táblázat. *Az* `<xsql:include-xsql>` *akció attribútumai*

Attribútumnév	*Description*
`href = "sztring"`	A beillesztendő XSQL-lap relatív vagy abszolút URL-je.
`reparse = "logikai"`	Azt jelzi, hogy a beillesztendő XSQL-lap által generált kimenetet szükséges-e beillesztés előtt újra elemezni. Ez abban az esetben lehet hasznos, ha a beágyazott lap által generált kimenet egy XML-dokumentumtöredék szövege, amelyet a beágyazó lap elemekként szeretne tekinteni. Az attribútum lehetséges értékei `yes` és `no`, az alapértelmezés a `no`.

Az elpostázott információk kezelése

Az XSQL-lapok az XML-tartalmak összeállításán és transzformációján túl az elpostázott XML-tartalmak kezelését is megkönnyíti. Az XML-dokumentumokból és HTML-űrlapokból elpostázott információk kezelését beépített akciók segítik, és így – az XSU lehetőségeit kihasználva – lehetővé válik az információ közvetlenül egy adatbázis-táblába történő postázása.

Az XSU lehetővé teszi egy kanonikus XML-dokumentum tartalmának adatbázisba történő beszúrását és frissítését, illetve lehetőséget biztosít a dokumentum tartalma alapján az adatbázisból történő törlésre.

Az XSQL-lapokban erre a következő beépített akciók segítségével van lehetőség:

- `<xsql:insert-request>`
  A kérés során elpostázott (esetleg transzformált) XML-dokumentum alapján beszúr egy táblába vagy nézetbe. Az akció attribútumait a 3.14. táblázat írja le.
- `<xsql:update-request>`
  A kérés során elpostázott (esetleg transzformált) XML-dokumentum alapján frissíti egy tábla vagy nézet adatait. Az akció attribútumait a 3.15. táblázat írja le.
- `<xsql:delete-request>`
  A kérés során elpostázott (esetleg transzformált) XML-dokumentum alapján töröl egy táblából vagy nézetből. Az akció attribútumait a 3.16. táblázat írja le.
- `<xsql:insert-param>`
  A kérés egyik paramétereként elpostázott (esetleg transzformált) XML-dokumentum alapján beszúr egy táblába vagy nézetbe. Az akció attribútumait a 3.17. táblázat írja le.

Ha a beszúrást egy adatbázisnézetbe végezzük, az elpostázott információ kezelésének további automatizálása érdekében létrehozhatunk egy INSTEAD OF INSERT triggert. Például egy nézetre megadott INSTEAD OF INSERT trigger PL/SQL-eszközök segítségével ellenőrizheti, hogy egy bizonyos rekord létezik-e, és ez alapján intelligens módon dönthet, hogy INSERT vagy UPDATE utasítást hajtson-e végre.

3.14. táblázat. *Az* `<xsql:insert-request>` *akció attribútumai*

Attribútumnév	*Leírás*
`table = "sztring"`	A beszúráshoz használandó tábla, nézet vagy szinonima neve.
`transform = "URL"`	A beszúrás alapjául szolgáló XML-dokumentumot kanonikus `ROWSET/ROW` formába alakító stíluslaptranszformáció relatív vagy abszolút URL-je.
`columns = "sztring"`	Azon oszlopnevek szóközökkel vagy vesszőkkel elválasztott listája, amelyekbe be kell szúrni. Ha az attribútum meg van adva, akkor csak ezekbe az oszlopokba történik meg a beszúrás, ellenkező esetben minden oszlopba; és azon oszlopok értékei, amelyek az XML-dokumentumban nincsenek jelen, NULL-értékek lesznek.
`commit-batch-size = "egész"`	A pozitív N attribútumérték azt jelenti, hogy minden N beszúrt rekord után végrehajtódik egy véglegesítés. Az alapértelmezés 0, így ha nem adjuk meg az attribútumot, az azt jelenti, hogy a kötegelt végrehajtás során nem történik véglegesítés.
`date-format = "sztring"`	Az attribútum értéke egy adatformátum-maszk, amely megadja, hogy hogyan kell az XML-dokumentum dátumértéket leíró mezőit értelmezni. Az érvényes értékek a `java.text.SimpleDateFormat` osztály dokumentációjában olvashatók.

3.15. táblázat. *Az* `<xsql:update-request>` *akció attribútumai*

Attribútumnév	*Leírás*
`table = "sztring"`	A frissítéshez használandó tábla, nézet vagy szinonima neve.
`key-columns = "sztring"`	Azon oszlopnevek szóközökkel vagy vesszőkkel elválasztott listája, amelyek értékei az elpostázott XML-dokumentumban a frissítésre szoruló létező sorok azonosítására szolgálnak.
`transform = "URL"`	A frissítés alapjául szolgáló XML-dokumentumot kanonikus `ROWSET/ROW` formába alakító stíluslaptranszformáció relatív vagy abszolút URL-je.
`columns = "sztring"`	Azon oszlopnevek szóközökkel vagy vesszőkkel elválasztott listája, amelyeket frissíteni kell. Ha az attribútum meg van adva, akkor csak ezek az oszlopok frissülnek, ellenkező esetben az összes oszlop frissül, és azon oszlopok értékei, amelyek az XML-dokumentumban nincsenek jelen, NULL értékek lesznek.
`commit-batch-size = "egész"`	A pozitív N attribútumérték azt jelenti, hogy minden N frissített rekord után végrehajtódik egy véglegesítés. Az alapértelmezés 0, így ha nem adjuk meg az attribútumot, az azt jelenti, hogy a kötegelt végrehajtás során nem történik véglegesítés.

3.15. táblázat. *Az* `<xsql:update-request>` *akció attribútumai (folytatás)*

Attribútumnév	*Leírás*
`date-format = "sztring"`	Az attribútum értéke egy adatformátum-maszk, amely megadja, hogy hogyan kell az XML-dokumentum dátumértéket leíró mezőit értelmezni. Az érvényes értékek a `java.text.SimpleDateFormat` osztály dokumentációjában olvashatók.

3.16. táblázat. *Az* `<xsql:delete-request>` *akció attribútumai*

Attribútumnév	*Leírás*
`table = "sztring"`	A törléshez használandó tábla, nézet vagy szinonima neve.
`key-columns = "sztring"`	Azon oszlopnevek szóközökkel vagy vesszőkkel elválasztott listája, amelyek értékei az elpostázott XML-dokumentumban a törlésre szoruló létező sorok azonosítására szolgálnak.
`transform = "URL"`	A törlés alapjául szolgáló XML-dokumentumot kanonikus `ROWSET/ROW` formába alakító stíluslap-transzformáció relatív vagy abszolút URL-je.
`commit-batch-size = "egész"`	A pozitív N attribútumérték azt jelenti, hogy minden N törölt rekord után végrehajtódik egy véglegesítés. Az alapértelmezés 0, így ha nem adjuk meg az attribútumot, az azt jelenti, hogy a kötegelt végrehajtás során nem történik véglegesítés.

3.17. táblázat. *Az* `<xsql:insert-param>` *akció attribútumai*

Attribútumnév	*Leírás*
`name = "sztring"`	A beszúrandó XML-adatot tartalmazó paraméter neve.
`table = "sztring"`	A beszúráshoz használandó tábla, nézet vagy szinonima neve.
`transform = "URL"`	A beszúrás alapjául szolgáló XML-dokumentumot kanonikus `ROWSET/ROW` formába alakító stíluslap-transzformáció relatív vagy abszolút URL-je.
`columns = "sztring"`	Azon oszlopnevek szóközökkel vagy vesszőkkel elválasztott listája, amelyekbe a beszúrást el kell végezni. Ha az attribútum meg van adva, akkor csak ezekbe az oszlopokba történik meg a beszúrás, ellenkező esetben minden oszlopba; és azon oszlopok értékei, amelyek az XML-dokumentumban nincsenek jelen, NULL értékek lesznek.
`commit-batch-size = "egész"`	A pozitív N attribútumérték azt jelenti, hogy minden N beszúrt rekord után végrehajtódik egy véglegesítés. Az alapértelmezés 0, így ha nem adjuk meg az attribútumot, az azt jelenti, hogy a kötegelt végrehajtás során nem történik véglegesítés.

3.17. táblázat. *Az* `<xsql:insert-param>` *akció attribútumai (folytatás)*

Attribútumnév	*Leírás*
`date-format = "sztring"`	Az attribútum értéke egy adatformátum-maszk, amely megadja, hogy hogyan kell az XML-dokumentum dátumértéket leíró mezőit értelmezni. Az érvényes értékek a `java.text.SimpleDateFormat` osztály dokumentációjában olvashatók.

Az XSQL-lapok az elpostázott információkat háromféleképpen tudják kezelni:

1. Egy kliens egy HTTP POST üzenetet küldhet egy XSQL-lapnak, amely kérés törzse egy XML-dokumentumot tartalmaz, a HTTP-fejlécben megadott tartalomtípus pedig `text/xml`.

   Ez esetben az `<xsql:insert-request>`, `<xsql:update-request>` és `<xsql:delete-request>` akciók használatakor az elpostázott XML-dokumentum tartalma a megadott táblába vagy nézetbe beszúrásra, frissítésre, illetve törlésre kerül.
2. Egy kliens egy olyan HTTP GET kérést küld az XSQL-lapnak, amelynek egyik paramétere egy XML-dokumentumot tartalmaz.

   Ekkor az `<xsql:insert-param>` akció segítségével az elküldött XML-dokumentum tartalma beszúrásra kerül a megadott táblába, illetve nézetbe.
3. A böngésző egy POST kérési metódussal kommunikáló HTML-űrlap tartalmát küldi el, melynek célja egy XSQL-lap. Ekkor a böngésző egy olyan HTTP POST üzenetet küld, amelynek törzse az űrlap összes mezőjének kódolt, `application/x-www-form-urlencoded` tartalom típusú értékét tartalmazza.

   Ebben az esetben a kérés nem tartalmaz XML-dokumentumot, csak az űrlap kódolt paramétereit. A három eset egységes kezelése érdekében azonban az XSQL-feldolgozó az űrlap paraméterei, és a kérésben szereplő munkamenet szintű változók és cookie-k segítségével felépít egy XML-dokumentumot, amelyet egy stíluslaptranszformáció segítségével kanonikus formába alakíthatunk, így az `<xsql:insert-request>`, `<xsql:update-request>` és `<xsql:delete-request>` akciók segítségével rendre beszúrhatjuk, frissíthetjük és törölhetjük az adatbázisban tárolt adatokat az XML-dokumentum alapján.

HTML-űrlapok elpostázása esetén a dinamikusan létrehozott XML-dokumentum a következő alakú:

```
<request>
  <parameters>
    <paraméternév1>érték1</paraméternév1>
    <paraméternév2>érték2</paraméternév2>
      ...
  </parameters>
  <session>
```

```
      <munkamenetváltozó-név>érték1</munkamenetváltozó-név>
        ...
    </session>
    <cookies>
      <cookie-név>érték1</cookie-név>
        ...
    </cookies>
  </request>
```

Ha több elpostázott paraméternek is ugyanaz a neve, akkor a további feldolgozást megkönnyítendő, automatikusan sorokká alakíthatók. Például tegyük fel, hogy a kérés a következő paramétereket tartalmazza:

- `azon` = 101
- `nev` = Kovács
- `azon` = 102
- `nev` = Szabó
- `muvelet` = update

Ezeket a paramétereket könnyen sorokká alakíthatjuk:

```
<request>
  <parameters>
    <row>
      <id>101</id>
      <name>Kovács</name>
    </row>
    <row>
      <id>102</id>
      <name>Szabó</name>
    </row>
    <operation>update</operation>
  </parameters>
      :
</request>
```

Ezt a dokumentumot kell egy XSL-transzformáció segítségével a beszúrandó táblának vagy nézetnek megfelelő kanonikus formára hozni.

Egyéni XSQL-akciókezelők használata

Az XSQL-keretrendszer lehetőséget biztosít egyéni akciók megadására is arra az esetre, ha olyan tevékenységet kell végrehajtanunk, amelyet a beépített akciókezelők nem kezelnek. Az egyéni akciók tetszőleges XML-tartalom adatcsomagba illesztését teszik lehetővé, és tetszőleges feldolgozási lépések végrehajtására alkalmasak. Az egyéni akciókezelő Java nyelven történő megvalósításának részleteiről az Olvasó bővebben az XSQL-keretrendszer dokumentációjából [19] tájékozódhat. Itt mi csak arra térünk ki, hogyan használhatunk egy már létrehozott egyéni akciókezelőt.

Egy egyéni akciókezelő használatához az `<xsql:action>` beépített akcióelemre van szükség. Ez egyetlen, `handler` nevű paraméterrel rendelkezik, amelyet kötelező megadni. A paraméter értéke annak a Java-osztálynak a teljes neve, amelyet meg akarunk hívni. Ennek az osztálynak implementálnia kell az `oracle.xml.xsql.XSQLActionHandler` interfészt. Például:

```
<xsql:action handler="sajatcsomag.EgyeniKezelo"/>
```

A kezelőnek tetszőleges számú további attribútumot átadhatunk. Ha például a `sajatcsomag.EgyeniKezelo` két, `param1` és `param2` nevű paramétert vár, akkor azt a következőképpen adhatjuk meg:

```
<xsql:action handler="sajatcsomag.EgyeniKezelo" param1="xxx"
param2="yyy"/>
```

Néhány akciókezelő az attribútumokon túlmenően az `<xsql:action>` elemen belül szöveges vagy elemtartalom megadását is igényelheti. Ezeket a következő módon adhatjuk meg:

```
<xsql:action handler="sajatcsomag.EgyeniKezelo" param1="xxx"
param2="yyy">
   Ide jön a szöveges tartalom.
</xsql:action>
```

illetve:

```
<xsql:action handler="sajatcsomag.EgyeniKezelo" param1="xxx"
param2="yyy">
  <elemek>
    <szerepelhetnek/>
    <itt/>
    <is/>
  </elemek>
</xsql:action>
```

4. FEJEZET

Oracle9*i*AS Portal

Az Oracle9*i*AS Portal egy keretrendszer, amely lehetőséget nyújt nagyméretű, integrált, biztonságos vállalati portálok tervezéséhez és készítéséhez. Az Oracle Portal az Oracle WebDB továbbfejlesztéseként jött létre az Oracle9*i* Application Server egyik komponenseként.

Ebben a fejezetben az Olvasó megismerheti az Oracle Portal céljait, szerkezetét, fogalmait. Egyszerű portáloldalak felépítésén keresztül elsajátíthatja a legalapvetőbb szerkesztő- és adminisztrációs eszközök használatát. Példát adunk arra is, hogy a Portal Development Kit keretrendszer segítségével hogyan készíthetjük el saját Java-portletjeinket, amelyekkel integrálhatjuk az általunk létrehozott más webalkalmazásokat az Oracle Portal szolgáltatásaiba. Ezzel szemben nem foglalkozunk a Portal telepítésének kérdéskörével, feltételezzük, hogy az Olvasó számára rendelkezésre áll egy telepített Oracle Portal példány, amelyhez megfelelő jogosultságokkal rendelkezik.

Az általunk használt Oracle Portal verziója 3.0.9.8.0, amelyet az Oracle9*i* Application Server Release 1 (v1.0.2.2.1) verziójával telepítettünk RedHat 7.3 Linux operációs rendszer környezetben. A szoftverben az alapinstallációnak megfelelően az angol nyelvi támogatást használtuk, azonban megjegyezzük, hogy lehetőség van emellett (és nem ehelyett) a magyar nyelvi támogatás utólagos installációjára is. Ennek leírását [15] tartalmazza.

4.1. Az Oracle Portal felépítése

Ebben a részben röviden áttekintjük az Oracle Portal rendszer felépítését, a portálokat alkotó tartalmi elemek típusait: az oldalakat, az adatterületeket és az alkalmazásokat. Szó lesz a rendszer biztonsági elveiről is. Ezután megismerjük az Oracle Portal Home Page oldalt, a súgó és a navigátor használatát.

4.1.1. Az Oracle Portal részei

Az Oracle Portal lehetővé teszi nagy mennyiségű információ hatékony kezelését, elérését, információ interaktív bevitelét weben keresztül. A rendszer telepítésével egy előre konfigurált induló portált kapunk. Elsősorban a fejlesztők és az adminisztrátorok feladata, hogy ezt a rendszert a benne levő eszközrendszer felhasználásával feltöltsék tartalommal, kialakítsák a kívánt felépítést és megjelenést, meghatározzák a rendszer elemeinek hozzáférhetőségét. Azt is meg kell határozniuk, hogy az egyes felhasználók, illetve felhasználói csoportok milyen mértékben szabhatják testre a saját ízlésüknek megfelelően a meglevő tartalmat, szerkezeteket és megjelenést.

Egy portál fejlesztése elsősorban tehát a szerkezet és a tartalom felépítését jelenti különböző tartalmi elemek létrehozásával. A Portal rendszerben ezek az elemek mindig három konstrukciótípus egyikéhez kötődnek. Mindhárom konstrukció tartalmi elemek bizonyos fajtáit fogja össze.

- *Adatterületek* (Content Areas) – Tartalmi elemek rendezett – alapvetően hierarchikus – tárolására szolgálnak. A tartalmazott elemeket több saját szempont szerint is osztályozhatjuk, valamint kereszthivatkozásokat adhatunk meg közöttük.
- *Alkalmazások* (Applications) – Oracle-adatbázisban tárolt adatok dinamikus megjelenítését, bevitelét és kezelését teszik lehetővé.
- *Oldalak* (Pages) – Az információ megjelenítői. Az adatterületek elemeit és más különböző forrásokból (adatbázisból, illetve külső forrásokból) származó adatokat egyazon oldalon összefogva jeleníthetünk meg.

A három típus közül központi szerepet töltenek be az oldalak. Noha ezek nélkül is létrehozhatunk adatterületeket és alkalmazásokat, azonban az Oracle Portal alapvetően *portálok* építésére szolgál. Portál alatt egy olyan web alapú alkalmazást értünk, amelynek segítségével a felhasználók adatterületeken tárolt elemeket, az Oracle Portalon kívüli webhelyek tartalmát, külső alkalmazásokat, hírszolgáltatókat és további hasznos információkat érhetnek el. Az információ prezentálására oldalakat hozunk létre, ahol a konkrét célnak megfelelően válogatjuk össze a megjelenítendő adatokat, alkalmazásokat. A felhasználó szemszögéből egy portált a felhasználó által elérhető oldalak összességének tekinthetünk. Fejlesztői szemszögből egy Oracle Portal példányt tekinthetünk egy portálnak, ideértve az oldalakon kívül az összes többi tartalmi és konfigurációs elemet is.

Az Oracle Portal az adatokat Oracle8*i* vagy 9*i* adatbázisban tárolja és az Oracle HTTP Servert használja az adatok szolgáltatására. Mindkét összetevőre jellemző a skálázhatóság és a nagyfokú biztonság. Ezeket a jellemzőket örökli a Portal rendszer is, természetesen megfelelő konfiguráció mellett.

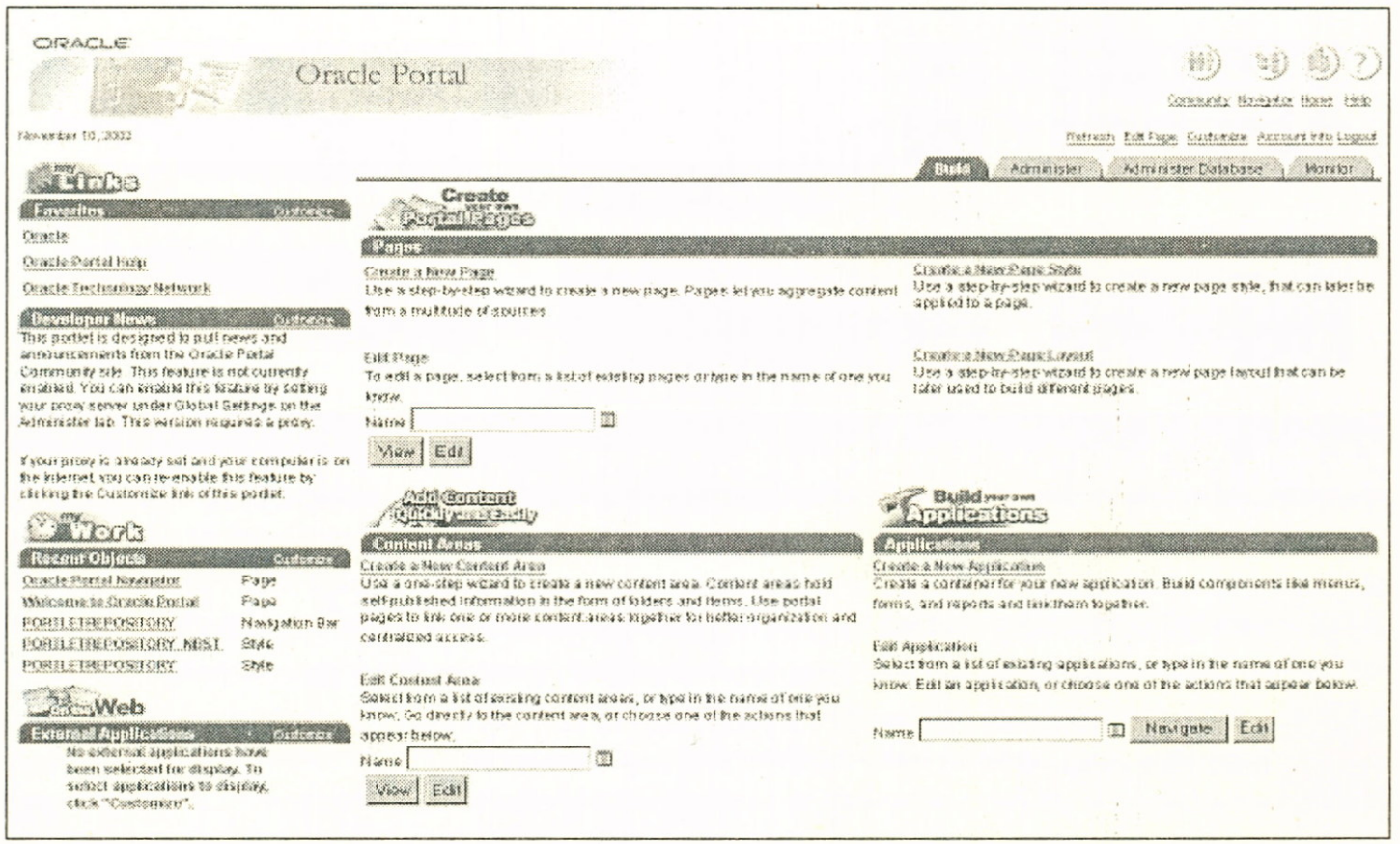
A Portal megvalósítása az Oracle HTTP Server *mod_plsql* modulján alapul, ahogyan az elődje, a WebDB is. A *mod_plsql* több PL/SQL-alkalmazás használatát is

lehetővé teszi, ezek közül egy a Portal, melyet az alkalmazások közt egy *Database Access Descriptor* (DAD) azonosít. A rendszer a mod_plsql modulon kívül szervleteket is használ.

4.1.2. Oldalak

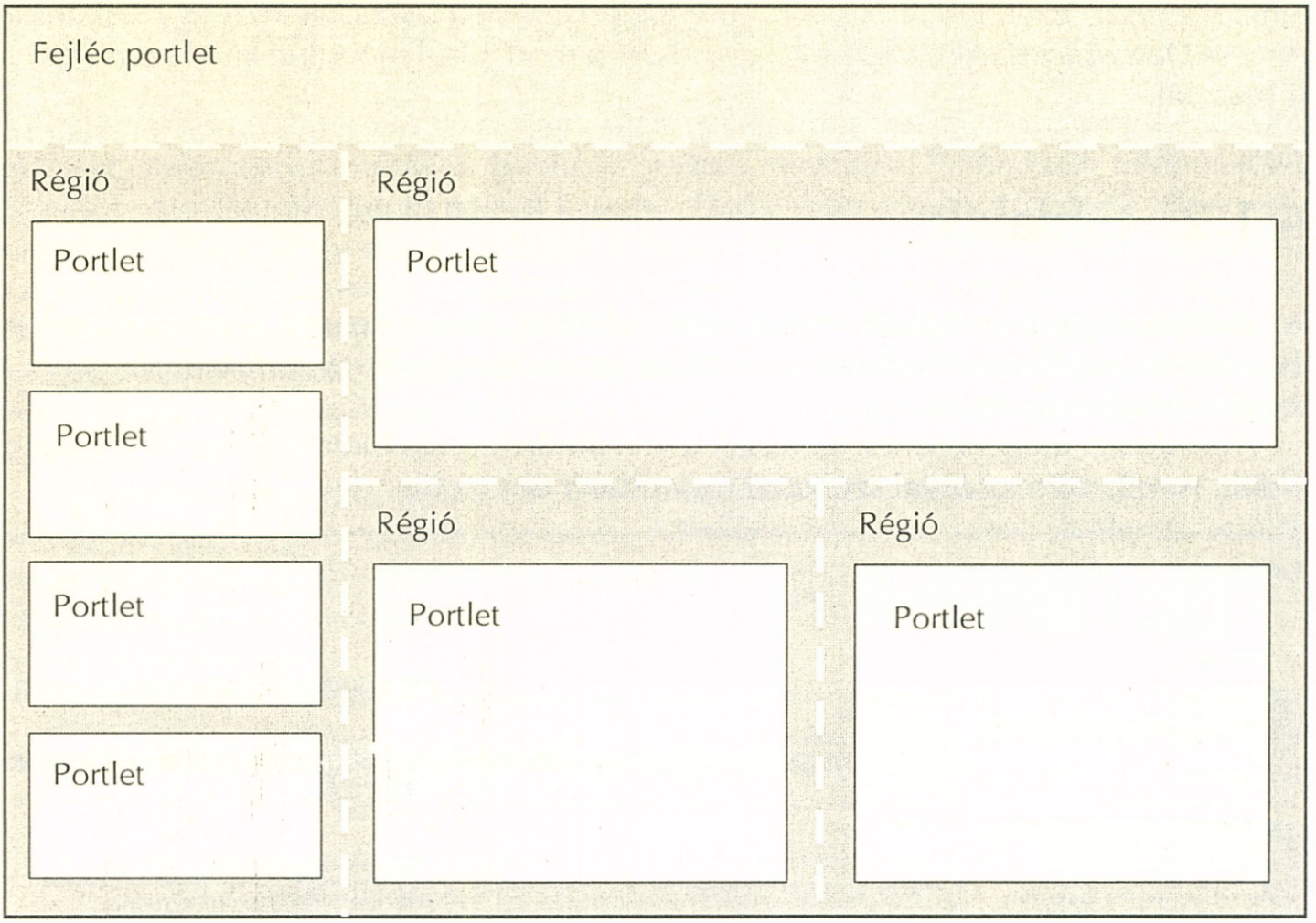
Az oldalak adják a portálunk felhasználói felületét. Rendelkeznek szerkezeti, stílus- és hozzáférési jellemzőkkel. Minden oldalhoz rendelni kell egy *oldalstílust*, amely a megjelenést befolyásolja.

A szerkezet megértéséhez tekintsük a 4.1. ábrán látható Oracle Portal Home Page oldalt. A 4.2. ábra az oldal szerkezeti felépítését ábrázolja.



4.1 ábra. *Az Oracle Portal Home Page oldal*

A tartalmi elemek minden oldalon téglalapokba – *régiókba* – rendezve jelennek meg. Az ábrán ezeket szaggatott vonal választja el. A régiók elrendezését a rendszer HTML-táblákkal valósítja meg. A régiók speciális konténerek, amelyek egy vagy több *portletet* tartalmazhatnak. A portlet egy újrafelhasználható információdarabka, amelyen keresztül valamilyen információforrást, például egy webhelyet vagy egy hírszolgáltatót érhetünk el. A portletek az Oracle Portal oldalak atomi építőkockái. Egy oldal általában több portletből épül fel, így a felhasználók a gyakran használt információkat egy helyen, egy oldalon összegyűjtve érhetik el. Az egyes portletek különböző funkcionalitást megvalósítva, szinte bármilyen forrásból származó adat prezentációjára alkalmasak.



4.2. ábra. *Az Oracle Portal Home Page oldal szerkezete*

Tekintsük át a portletek néhány tipikus felhasználási módját.

- *Információ központosított elérése* – Olyan oldalakat hozhatunk létre, melyek több, a weben létező információ elérését valósítják meg egy oldalon. Az Oracle Portal segítségével könnyedén összegyűjthetjük egy oldalra azokat a linkeket, amelyekkel a felhasználók elérhetik ezeket a számukra fontos, gyakran használt szolgáltatásokat.
- *Dokumentumok megosztása, publikálása* – A meglevő eszközkészlettel állományokat és egyéb tartalmat tölthetünk a böngészőnkön keresztül az adatterület mappáiba. A feltöltött adatokat rendezhetjük és a mappák tartalmát portletekként bármely portáloldalon közzétehetjük.
- *Dinamikus adatszolgáltatások integrálása* – Külső információk, például időjárás-jelentés, hírek stb. megjelenítése a vállalati információk mellett.
- *Webalkalmazások megvalósítása, elérése* – Gyakran van szükségünk egy alkalmazás valamely speciális részére, ilyenkor általában az alkalmazásba bejelentkezve elnavigálunk a keresett részhez. Erre a célra használhatunk olyan portleteket, amelyek bejelentkeznek helyettünk az alkalmazásba (gyakrabban annak adatforrásába), s a keresett információt egy portáloldal részeként jelenítik meg.

- *Vállalati rendszerek felületének egységesítése* – Az egy vállalaton belül használt különböző célú rendszerek fajtája nagyon sokféle lehet. Portletek segítségével egységes felület mögé rejthetjük a különböző rendszereket.

A portletek funkcionális, újrafelhasználható egységek, melyeket megfelelően paraméterezve a kívánt információt szolgáltatják. Az oldalakon megjelenő portletek már megfelelően paraméterezett példányai az adott típusú portleteknek. Az egyes portlettípusok mindig pontosan egy *portletszolgáltatóhoz* tartoznak. Egy portletszolgáltató több portletet is nyújthat. Portletszolgáltatók lehetnek az adatterületek, az alkalmazások, sőt akár a már létező oldalaink is. Lehetőség van ezektől eltérő, egyedi portletszolgáltatók definiálására is a Portal Development Kit használatával. Így bármely webalkalmazásunkat is felhasználhatjuk portletként az oldalainkon. Erről bővebben lásd a 4.6. fejezetet.

Ha az Oracle Portalt elosztottan telepítjük, akkor az egymással kapcsolatban levő Portal csomópontok felhasználhatják egymás portletjeit, amennyiben ezt a portletet tartalmazó csomópontban engedélyezzük. Az elosztott telepítésről információt a [15] tartalmaz.

A 4.2. ábrán szerepel egy fejléc is, ami nem kapcsolódik egyetlen régióhoz sem. Ennek ellenére a fejlécet is egy portlet (fejlécportlet) állítja elő.

4.1.3. Adatterületek

Minden adatterület mappákból épül fel. A mappák adatelemeket tárolnak ugyanúgy, ahogy az állományrendszer mappái állományokat. Egy multimédiás adatokat tároló adatterület esetén például Kazetták, CD-k stb. nevű mappáink lehetnének. A mappák adatelemei URL-ek, szöveges információk, képek, dokumentumok, kapcsolódó mappákra hivatkozó linkek stb. lehetnek. Így például a Kazetták mappában konkrét kazetták információit, kazettaborítókat, együttesek weboldalaira hivatkozó URL-eket stb. tárolhatnánk.

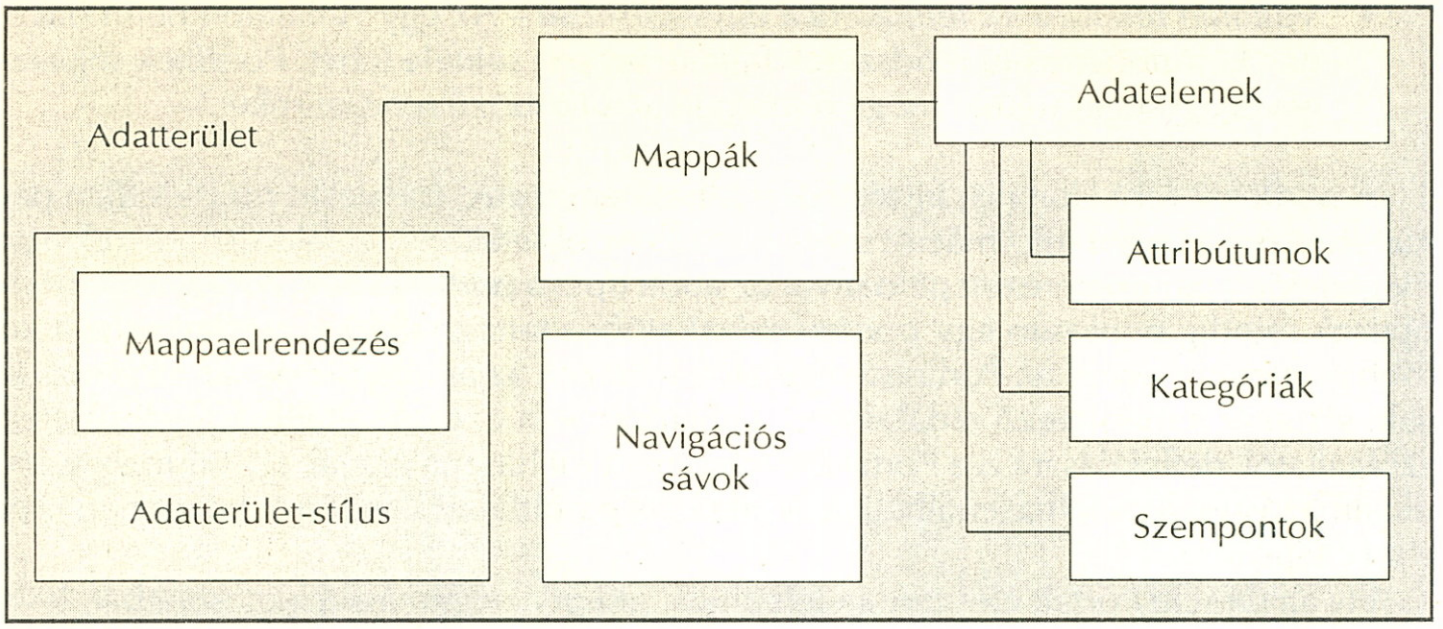
A 4.3. ábra az adatterületek szerkezeti felépítését mutatja.

Az adatterületek megjelenését a hozzárendelt *adatterület-stílus* határozza meg, beleértve a *mappaelrendezéseket* is, amelyek mappánként eltérőek is lehetnek.

Az adatterületeken elhelyezett *navigációs sávok* segítségével az információk elérése tovább gyorsítható, ésszerűsíthető.

Az adatterületek adatelemek rendezett tárolására szolgálnak. Minden adatelem rendelkezik egy típussal, amely meghatározza azt, hogy az adott típusú elemek milyen leíró *attribútumokkal* kell rendelkezzenek. Vannak beépített elemtípusok, de mi magunk is hozhatunk létre típust, megadva a szükséges leíró attribútumokat. Vannak rögzített attribútumok, amivel minden elem rendelkezik, például a megnevezése, amivel az adatterületen megjelenik.

Minden adatelemet kötelezően be kell sorolni pontosan egy *kategóriába*. A kategóriák a „Milyen témához kapcsolódnak az elemek?" kérdésre adott válaszok. A kategóriák nem beépített, hanem az *adatterület-adminisztrátor* által létrehozott diszjunkt



4.3. ábra. *Adatterület elemeinek kapcsolata*

osztályok. Minden elemet pontosan egy osztályba kell tehát besorolni. Egy multimédiás adatterület esetén például a következő kategóriák létezhetnek: Előadó/Szerző, Kritika, Poszter stb. Kategóriák használatával lehetőségünk van az azonos típusú objektumok együttes megjelenítésére, és egy kategória elemei között keresni is tudunk.

Az adatelemek célszerű elérését tovább könnyítendő *szempontokat* használhatunk. A szempontok tetszőleges csoportosítások, lehetnek átfedők is. Egy adatterületen tetszőleges számú független szempontunk lehet. Az adatelemeket nem kötelező szempontokhoz rendelni, de akár több szemponthoz is rendelhetjük. A multimédiás adatterületünkön az elemek rendezésére így újabb csoportosításokat hozhatnánk létre, például a műfajok szerint (Pop, Rock, Komédia, Dráma stb.), korok szerint (70-es évek, 80-as évek stb.). Az azonos szempontokhoz kapcsolt elemek szintén kezelhetők együtt, megjeleníthetők együtt, a megadott szempontok keresési feltételként használhatók.

Az adatterületeken tárolt adatok a mappák által adott hierarchiába rendezetten hozhatók létre, s ezenfelül saját navigációs eszközökkel, a navigációt megkönnyítő kategóriákkal és szempontokkal rendelkeznek. Az egy adatterületen tárolt adatok viszonylag szorosan kapcsolódnak egymáshoz. Ezzel szemben egy portáloldalon egymástól független, lazán, a felhasználás szempontjai szerint kapcsolódó információk találhatók. Aki egy konkrét adatterület sok elemére kíváncsi, jól ismeri az uralkodó rendező elveket, szívesen használja magát az adatterületet a számára szükséges adatok elérésére. Más felhasználóknak azonban nehézkes lehet egy konkrét, ezen az adatterületen tárolt információ elérése, mivel ők nem ismerik olyan jól az adatterület szerkezetét, rendező elveit. Ezt elkerülendő, az adatterületet portletszolgáltatóként használhatjuk fel, a keresett információkat tároló mappákat pedig portletekként integrálhatjuk más információk mellé a portáloldalakra.

Vegyünk egy példát: A személyzeti osztály a vállalati intraneten információkat tárol a *Személyzeti osztály információi* adatterületen. A következő évi szabadságolások megtervezéséhez készítenek egy dokumentumot, és szeretnék, ha ezt mindenki

letöltené, kinyomtatná és kitöltve elküldené. A dokumentumot feltöltik az adatterületre egy mappába, ahova további dokumentumok, kitöltési útmutató stb. is felkerülnek. Ezután szólnak az intranet portáladminisztrátorának, hogy ez az információ jelenjen meg az intranet közös nyitóoldalán. A portáladminisztrátor ezután a mappát portletként kihelyezi a nyitóoldalra.

4.1.4. Alkalmazások

Egy alkalmazás *komponensek* összességéből áll. Különböző funkciójú komponensek vannak: menük, diagramok, űrlapok, naptárak stb. Ezek használatára később a 4.4. alfejezetben még visszatérünk. Egy alkalmazáson belül több azonos típusú komponensünk is lehet. A különböző komponenseket összekapcsolva egy teljes web alapú adatbázis-alkalmazást hozhatunk létre. Például egy összesítő diagramon keresztül elérhetjük az oszlopokhoz tartozó egyes jelentéseket. Egy alkalmazás komponenseit portáloldalakon is használhatjuk, ha az alkalmazást portletszolgáltatóként rendelkezésre bocsátjuk.

Minden alkalmazás alapja egy adatbázisséma. Ebben a sémában a komponensek a létező, illetve az itt létrehozott objektumokkal dolgoznak. Egy űrlapkomponens segítségével egy vagy több adatbázisbeli tábla tartalmát kezelhetjük, az adatbázison végrehajtott lekérdezések eredményeit megjeleníthetjük diagramokon stb.

4.1.5. Biztonság

Az Oracle Portal felhasználókat és felhasználói csoportokat kezel. Minden kliens, aki a portált használja, valamilyen felhasználó kell legyen. Ez akkor is igaz, ha nem történt bejelentkezés (ez nem ekvivalens azzal, ha a felhasználót a rendszer automatikusan belépteti), ilyenkor egy speciális felhasználói szerepkörben érjük el a portált, amit `PUBLIC`-nak hívnak.

Az Oracle Portal minden építőelemére külön szabályozható egyéni vagy csoport szinten, hogy ki milyen jogosultságokkal érheti el az oldalt. Ezt a rendszer *hozzáférési listák* alkalmazásával valósítja meg, angolul *access control list*, továbbiakban ACL. A konténer típusú szerkezeti elemek, mint például oldalak, adatterületek, mappák, alkalmazások stb. hozzáférési tulajdonságait örökölhetik a tartalmazott objektumok. Ezzel egyszerűsödik a rendszer adminisztrációja.

Emellett lehetőség van arra is, hogy azonos típusú objektumokra együttesen adjunk jogosultságokat a felhasználóknak és csoportoknak. Például adhatunk `All Pages : Create` jogosultságot, aminek birtokában a felhasználó bárhol létrehozhat oldalakat. A privilégiumok szabályozása megengedő, vagyis azt kell megmondanunk, hogy egy adott felhasználó mit tehet, azt nem mondhatjuk meg, hogy mit nem tehet, nincs tiltás. Egy felhasználó egy objektumra több forrásból is rendelkezhet jogosultságokkal.

4.1.6. Az Oracle Portal Home Page

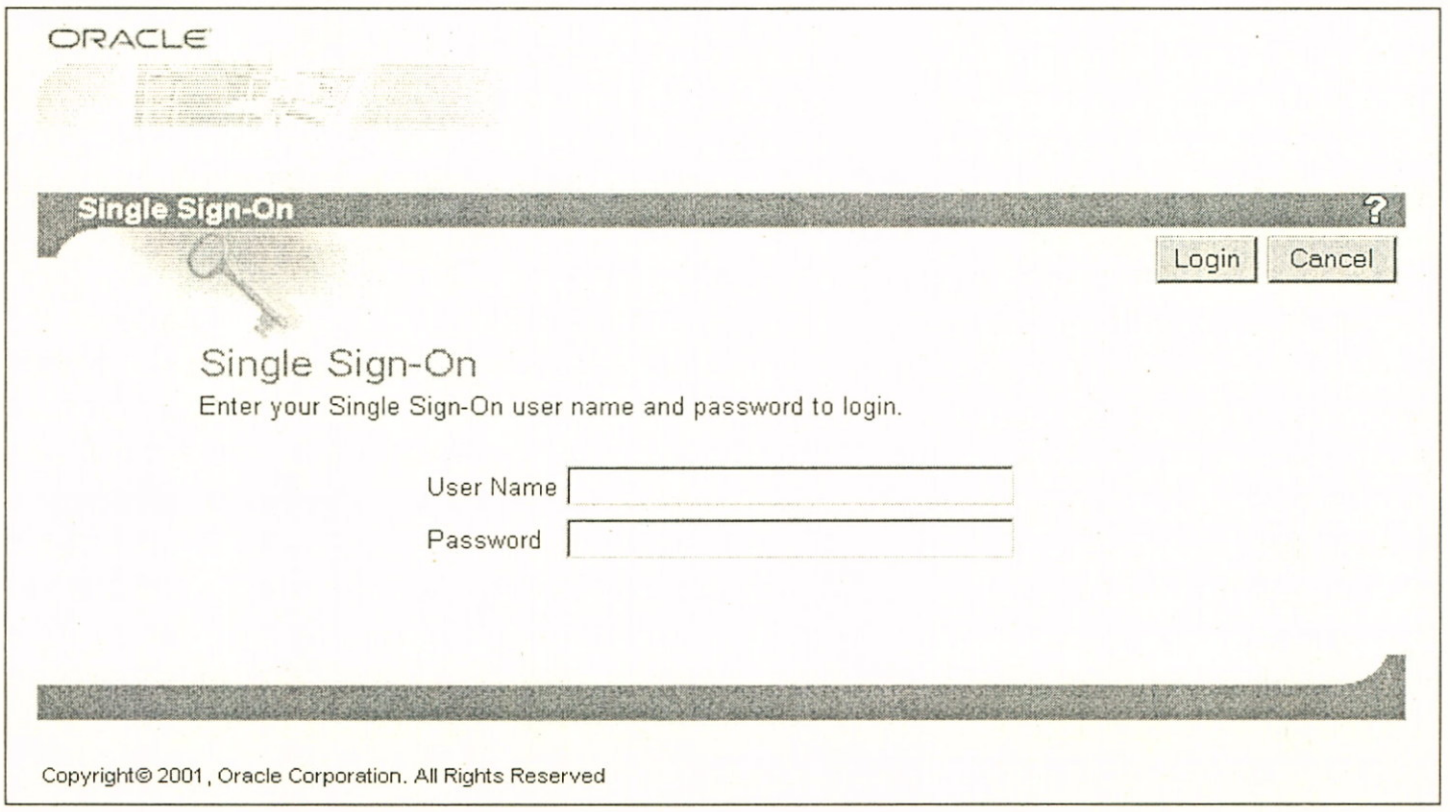
Az Oracle Portal telepítésekor a telepítő közli velünk a telepített Portal példány URL-jét. Ha beírjuk a böngészőbe ezt az URL-t, akkor az üdvözlő lap jelenik meg. Itt a jobb felső sarokban a `Login` / *Bejelentkezés* linkre kattintsunk, ez továbbít az *Oracle Single Sign-On* oldalra. A beléptetést a Single Sign-On (SSO) végzi (4.4. ábra), amely cookie alapú azonosítást használ. Erről további információt a [22] talál.

Felhasználói nevet a portáladminisztrátortól kérjünk, vagy használjuk a telepítéskor megadott sémanévvel létrehozott felhasználót. Ennek alapértelmezett neve `PORTAL30`, ez a felhasználó rendelkezik a legfőbb jogosultságokkal a Portal példányon belül.

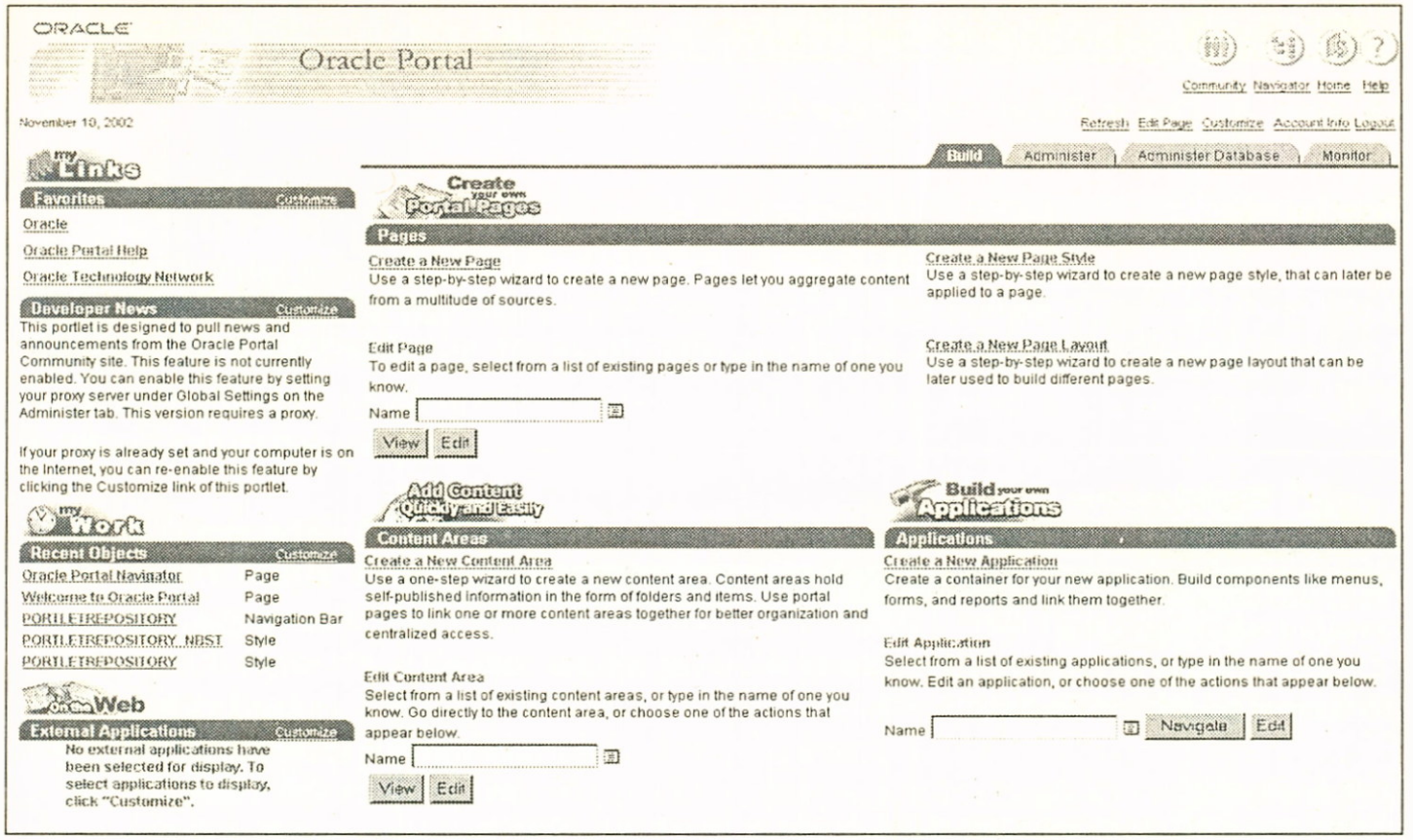
A bejelentkezéshez adjuk meg felhasználónevünket és jelszavunkat, majd kattintsunk a `Login` gombra. A sikeres bejelentkezés után a testre szabott főoldal jelenik meg (4.5. ábra).

Az oldal egy már használatban levő rendszerben az alapértelmezettől eltérhet, megjelenése függ attól, hogy milyen jogosultságokkal rendelkezünk, és attól is, hogy milyen egyéni vagy csoportos beállítások vannak érvényben az oldalra.

Feltételezzük, hogy az Olvasó számára az Oracle Portal Home Page van beállítva főoldalnak. Ha ez nem így van, akkor a navigátor segítségével juthat el erre az oldalra. A navigátor használatáról a 4.1.8. alfejezetben lesz szó. Ha nincs navigátor link az oldalon, kérje a portáladminisztrátor segítségét, valószínűleg olyan felhasználót használ, amely nem rendelkezik megfelelő jogosultságokkal a portál építéséhez, adminisztrálásához.



4.4. ábra. *Oracle Single Sign-On*



4.5. ábra. *Oracle Portal Home Page*

Az Oracle Portal Home Page oldal az az oldal, amiről kiindulva minden portálépítő, adminisztrációs és felügyeleti rendszert elérhetünk. A következőkben áttekintjük az oldal felépítését, a rajta található eszközöket.

Lapok

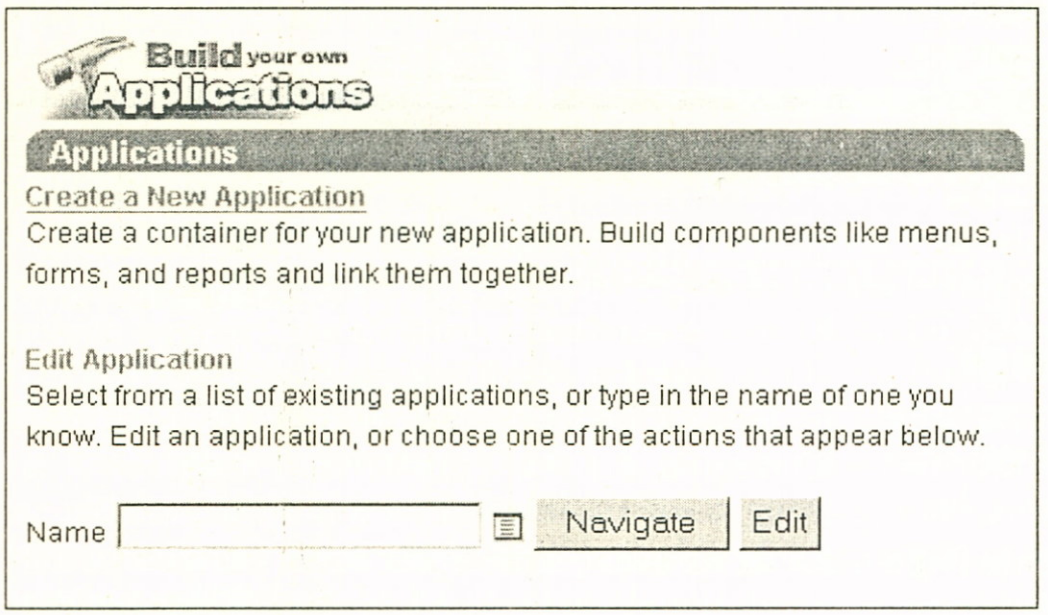
Az oldalon több fül van, ezekre kattintva a fülek által szabályozott régiócsoportok közt válthatunk. A fülek a következők (lásd 4.6. ábra):

- A `Build` / *Épít* lapon találhatók az oldalak, adatterületek és alkalmazások létrehozását, szerkesztését segítő portletek.
- Az `Administer` / *Adminisztrál* lapon találhatók a portáladminisztrációs eszközök, mint például felhasználók és csoportok létrehozása, globális jellemzők beállítása. Alapértelmezés szerint ehhez az oldalhoz csak a portáladminisztrátorok férnek hozzá.
- Az `Administer Database` / *Adatbázis adminisztrálása* lapon találhatók az adatbázist adminisztráló eszközök, beleértve a sémák létrehozását, a privilégiumok adományozását és az adatbázis-felügyeleti eszközöket. Alapértelmezés szerint ehhez az oldalhoz csak az adatbázis-adminsztrátorok férnek hozzá.
- A `Monitor` / *Felügyel* lapon találhatók az oldalak, az adatterületek és az alkalmazások felügyeletét biztosító eszközök. Alapértelmezés szerint ehhez az oldalhoz csak a portáladminsztrátorok férnek hozzá.

4.6. ábra. *Az Oracle Portal Home Page fülei*

Portletek

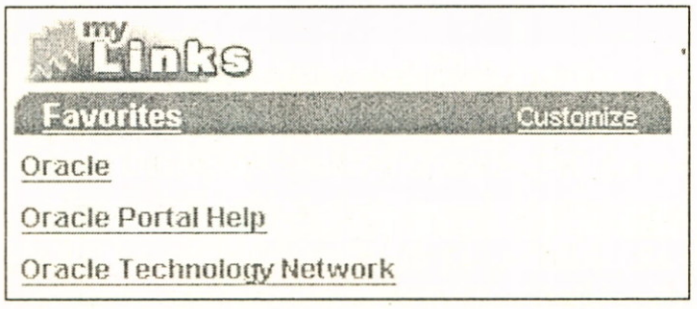
A lapokon található portletekkel a Portal eszközöket érhetjük el. Példaként nézzük a 4.7. ábrán látható portletet, amely az alkalmazások készítéséhez nyújt belépési pontot.



4.7. ábra. *Alkalmazások portlet*

Az egyes lapokhoz tartozó fülek csak akkor jelennek meg az oldalon, ha megfelelő jogosultságokkal rendelkezünk. A jogosultságok azt is befolyásolják, hogy a fülekhez tartozó lapokon milyen portletek lesznek láthatók. Ha például nincs jogosultságunk alkalmazások kezelésére, akkor az előbb említett alkalmazások portlet nem fog megjelenni.

A Portal Home Page bal szélén található portletek nem tartoznak egy fülhöz sem. Ezeken általánosabb jellegű információkat találunk. Ilyen például a `Favorites` / *Kedvencek* portlet (4.8. ábra), amin a felhasználó gyakran használt linkjei érhetők el. Többek között ez a portlet is testre szabható. Ezt a portlet fejlécén található `Customize` /



4.8. ábra. *Favorites portlet*

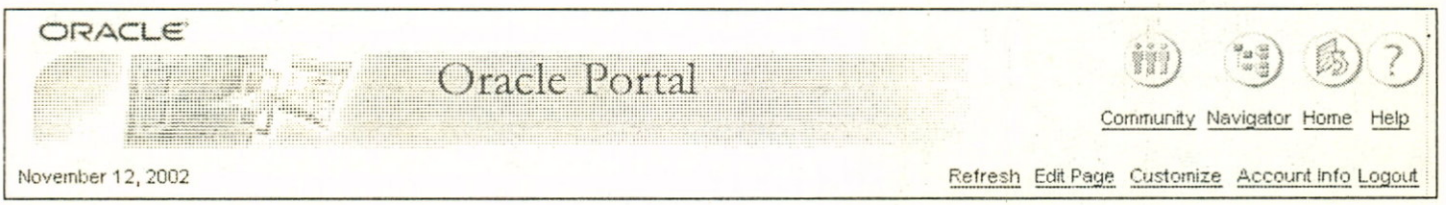
Testre szabás link jelzi. Így a linkeket minden felhasználó saját maga választhatja ki, szerkesztheti.

Fejléc

A fejléc az oldal felső részén elhelyezhető sáv (4.9. ábra).

Az oldal fejlécének bal alsó sarkában az aktuális dátum, jobb oldalon – a *gyorsindító sávon* – hasznos linkek láthatók. A gyorsindító felső részén helyezkednek el az ikonnal is ellátott, úgynevezett elsődleges linkek. Ezek esetében a szövegre vagy az ikonra is kattinthatunk:

- A `Community` / *Közösség* link a `http://portalcommunity.oracle.com` webhelyre mutat, ami a portálfejlesztők on-line közössége az oracle oldalon belül, itt nagyon sok hasznos információt szerezhetünk más fejlesztőktől, megoszthatjuk tapasztalatainkat, segítséget kérhetünk.
- A `Navigator` / *Navigátor* link az azonos nevű oldalra vezet, ahol a portál objektumai közt navigálhatunk (4.1.8. alfejezet).
- A `Home` / *Főoldal* link mindig a felhasználó számára beállított főoldalra mutat. Már korábban feltételeztük, hogy az Olvasó esetében ez maga az Oracle Portal Home Page. Ugyanerre az oldalra mutat az Oracle Portal Logo kép is, ez teszi ki a fejléc legnagyobb részét.
- A `Help` / *Súgó* linkkel a beépített súgórendszert érhetjük el, használatával a következő alfejezet foglalkozik.



4.9. ábra. *Fejléc*

Az elsődleges linkek alatt helyezkednek el a másodlagos linkek, melyek kevésbé gyakori funkciókat takarnak, ezért ezek elérését nem segíti ikon.

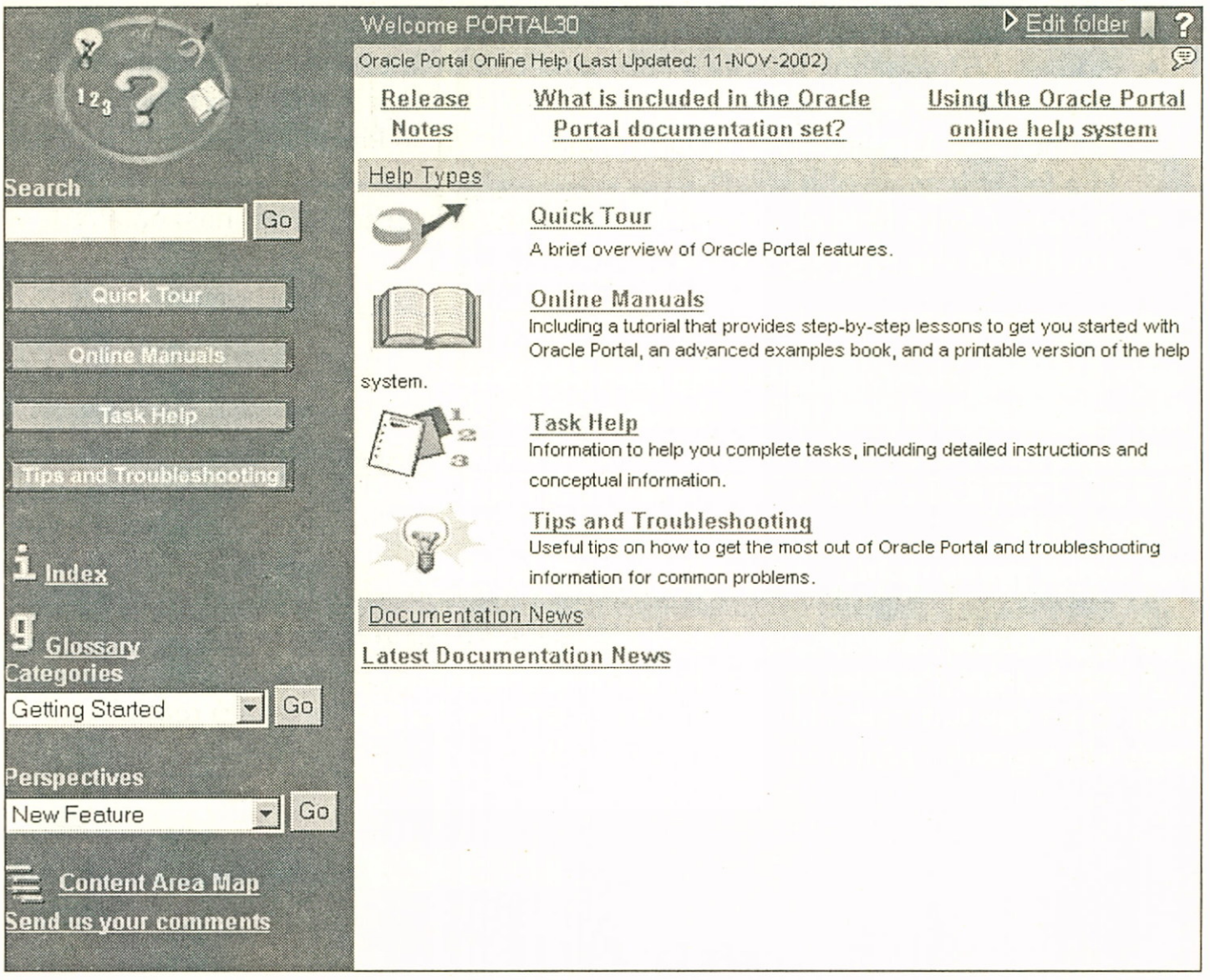
A linkek, mint ahogyan az általuk elérhető funkciók is, jogosultságokhoz vannak kötve. Ezért egy link csak akkor kerül rá egy oldalra, ha a felhasználó rendelkezik a megfelelő jogosultsággal a funkció használatához.

4.1.7. Súgó

Az Oracle Portalban a súgó nyelve angol. Kétféle súgó létezik:

- *On-line súgó* – Részletes segítséget nyújt az Oracle Portal használatához. Tartalmaz lépésenkénti oktatóanyagot, referenciát, bevezető jellegű információkat és a gyakran felmerülő problémákhoz is segítséget nyújt. Eléréséhez kattintsunk a gyorsindító sáv ? ikonjára.
- *Mező szintű súgó* – Az egyes oldalakon található eszközök, portletek használatáról ad információt. Ezt a portletek fejlécén található ?-re kattintva érhetjük el.

Az on-line súgó (4.10. ábra) tulajdonképpen egy adatterület, amely már fel van töltve tartalommal. Ez az adatterület a telepítés során automatikusan jön létre.



4.10. ábra. *On-line súgó*

Az on-line súgó új böngészőablakban jelenik meg. Bal oldalán, mint az adatterületek esetén általában, a navigációs sáv található. A jobb oldalon látjuk a mappák tartalmát. Ismerkedésként hajtsuk végre a következőket:

1. Kattintsunk a `Task Help` / *Segítség feladatokhoz* linkre vagy a navigációs sávon, vagy a `Help Types` / *Segítség típusai* mappában.
2. Itt válasszuk a `The Basics` / *Alapok* linket.
3. Válasszuk a `What is Oracle Portal?` / *Mi az Oracle Portal?* linket.
4. Új böngészőablakban megjelenik a keresett információ.
5. A súgó oldalak esetenként több témában is segítséget nyújtanak. Az oldal alján mindig ott találjuk a `Related topics` / *Kapcsolodó témák* részt. Kattintsunk a `What is a page?` / *Mi az oldal?* linkre, hogy az oldalakról szóló súgóoldalt megtekintsük.

Segítséghez nem csak a mappákban navigálva juthatunk. Használhatjuk a navigációs sáv `Index` / *Tárgymutató* linkjét adott szóhoz kapcsolódó oldalak kereséséhez. Ugyanitt található a `Glossary` / *Szójegyzék*, ahol a Portalhoz kapcsolódó szakszavak magyarázatát lelhetjük meg.

Az információkat kategóriájuk szerint is elérhetjük. Ehhez válasszuk ki a megfelelő kategóriát a `Categories` / *Kategóriák* lenyíló listából, majd kattintsunk a `Go` / *Mehet* gombra. Ha például az `Administration Tasks` / *Adminisztrációs feladatok* kategóriát választjuk, akkor adatbázis-objektumok és felhasználók kezeléséről szóló témák listáját kapjuk.

Hasonlóképpen használhatjuk a `Perspectives` / *Szempontok* lenyíló listát. Itt más csoportosítás szerint érhetjük el a súgóoldalakat. A szempontok – egy kivételével – a felhasználói szerepkörök szerint rendszerezik az elérhető információkat.

Ha a mappák közt navigálva elveszettnek érezzük magunkat, és már a böngésző `Vissza` gombja sem segít, használjuk a `Content Area Map` / *Adatterület térképe* linket, ahol az adatterületek, illetve mappáik közt választhatunk. Ha pedig csak egyszerűen a súgó nyitóoldalára szeretnénk visszajutni, kattintsunk a bal felső sarokban a nagy kérdőjelre.

Mindemellett használhatjuk a `Search` / *Keresés* funkciót. Írjuk be a kereső mezőbe a keresett szót. Amennyiben az *inter*Media opció telepített és engedélyezett, akkor egy *inter*Media Text index is létrehozásra került, így a témák címén túlmenően a tartalmukban is keres a kereső.

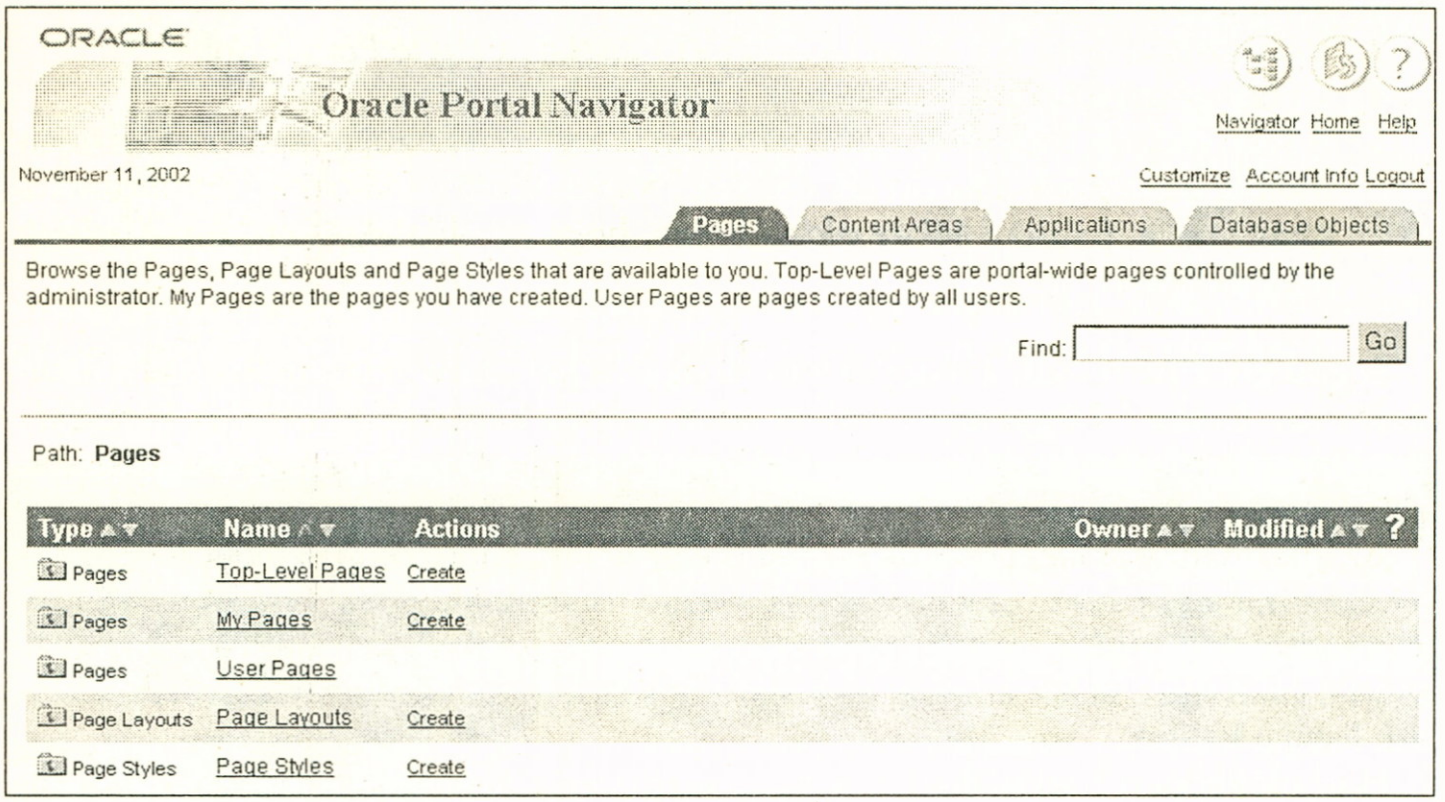
Az on-line súgót a böngészőablakkal együtt zárhatjuk be.

A mező szintű súgó használatának megismeréséhez hajtsuk végre a következőket:

1. A gyorsindító sávon kattintsunk a navigátorra.
2. A navigátor oldalán a portlet fejlécén kattintsunk a ?-re.
3. Új böngészőablakban megjelenik a portlethez kapcsolódó segítség.

4.1.8. Navigátor

A navigátor oldalon (4.11. ábra) a portál különböző objektumai közt navigálhatunk, új objektumokat hozhatunk létre, a létező objektumokon a jogosultságainktól függően műveleteket végezhetünk.



4.11. ábra. *A navigátor oldal*

Az oldalon elhelyezett fülek (4.12. ábra) segítségével válthatunk a Portal objektumtípusai között.

4.12. ábra. *A navigátor oldal fülei*

A fülek lapjain a következő típusú objektumokat kezelhetjük:

- A `Pages` / *Oldalak* lapon a portáloldalakat, oldalelrendezéseket és oldalstílusokat tekinthetjük meg, kezelhetjük.
- A `Content Areas` / *Adatterületek* lapon új adatterületeket hozhatunk létre, a meglévőket kezelhetjük és az adatterületek által közösen használt `Shared Objects` / *Megosztott objektumokat* (személyes mappák, kategóriák, navigációs sávok, szempontok, stílusok és saját típusok) érhetjük el.
- Az `Applications` / *Alkalmazások* lapon az alkalmazások mellett szintén találhatók megosztottan használt komponensek `Shared Components` néven (színek, betűtípusok, képek, JavaScriptek és felhasználói felületek).

- A `Database Objects` / *Adatbázis-objektumok* lapon az adatbázis objektumait kezelhetjük, amennyiben adatbázis-adminisztrátori jogosultsággal is rendelkezünk.

A navigátorban is használhatjuk a keresőt. A `Path` / *Útvonal* után láthatjuk, hogy az objektumok közt navigálva éppen hol járunk, linkjein az úton felfelé navigálhatunk, például:

Path: Applications > Shared Components > **Colors**

4.2. Oldalak létrehozása

Ebben a fejezetben áttekintjük egy egyszerű oldal létrehozásának lépéseit. Az oldalakhoz kapcsolódó biztonsági kérdések tárgyalásához létrehozzuk az `ORAWEB` felhasználót.

4.2.1. Az ORAWEB felhasználó létrehozása

A továbbiakban több példán keresztül bemutatjuk, hogyan lehet különböző Portal objektumokat létrehozni, kezelni. Ebbe beletartoznak az oldalakra vonatkozó biztonsági beállítások is. Ahhoz, hogy erre is példát adhassunk, most létrehozzuk az `ORAWEB` nevű felhasználót. Ehhez rendelkezni kell megfelelő jogosultságokkal (SSO-adminisztrátor és a portálra nézve `All Users : Manage` globális jogosultsággal). A lépések:

1. A felhasználó létrehozásához válasszuk az `Administer` fület az Oracle Portal Home Page oldalon.
2. Itt a `User` / *Felhasználó* felíratú portleten kattintsunk a `Create New Users` / *Új felhasználók létrehozása* linkre.
3. Most létrehozunk egy felhasználót, aki SSO-felhasználó és Portal-felhasználó is lesz egyben. A `User Name` / *Felhasználó neve* mezőbe írjuk be: `ORAWEB`. A `Password` / *Jelszó* és a `Confirm Password` / *Jelszó megerősítése* mezőkbe írjunk be egy tetszőleges, általunk választott jelszót. A többi mező kitöltése számunkra nem lényeges most. A biztonság kedvéért azért ellenőrizzük, hogy a lap alján a `Login Server Privilege Level` / *Bejelentkezési szerver (SSO) jogosultsági szint* mező értéke `End User` / *Végfelhasználó* maradt-e. A kis- és nagybetűk nem különböznek sem a felhasználói névben, sem a jelszóban.
4. A felhasználó létrehozásához kattintsunk az oldal tetején a `Create` / *Létrehoz* gombra.

5. A felhasználó létrehozása után a rendszer felajánlja a felhasználó profiljának szerkesztését, ettől azonban most eltekintünk. Mivel több felhasználót már nem hozunk létre, kattintsunk a `Close` / *Bezárás* gombra.

A felhasználó ellenőrzéséhez lépjünk be próbaként a rendszerbe `ORAWEB` néven. Rendelkezik-e ez a felhasználó adminisztrátori jogosultságokkal?

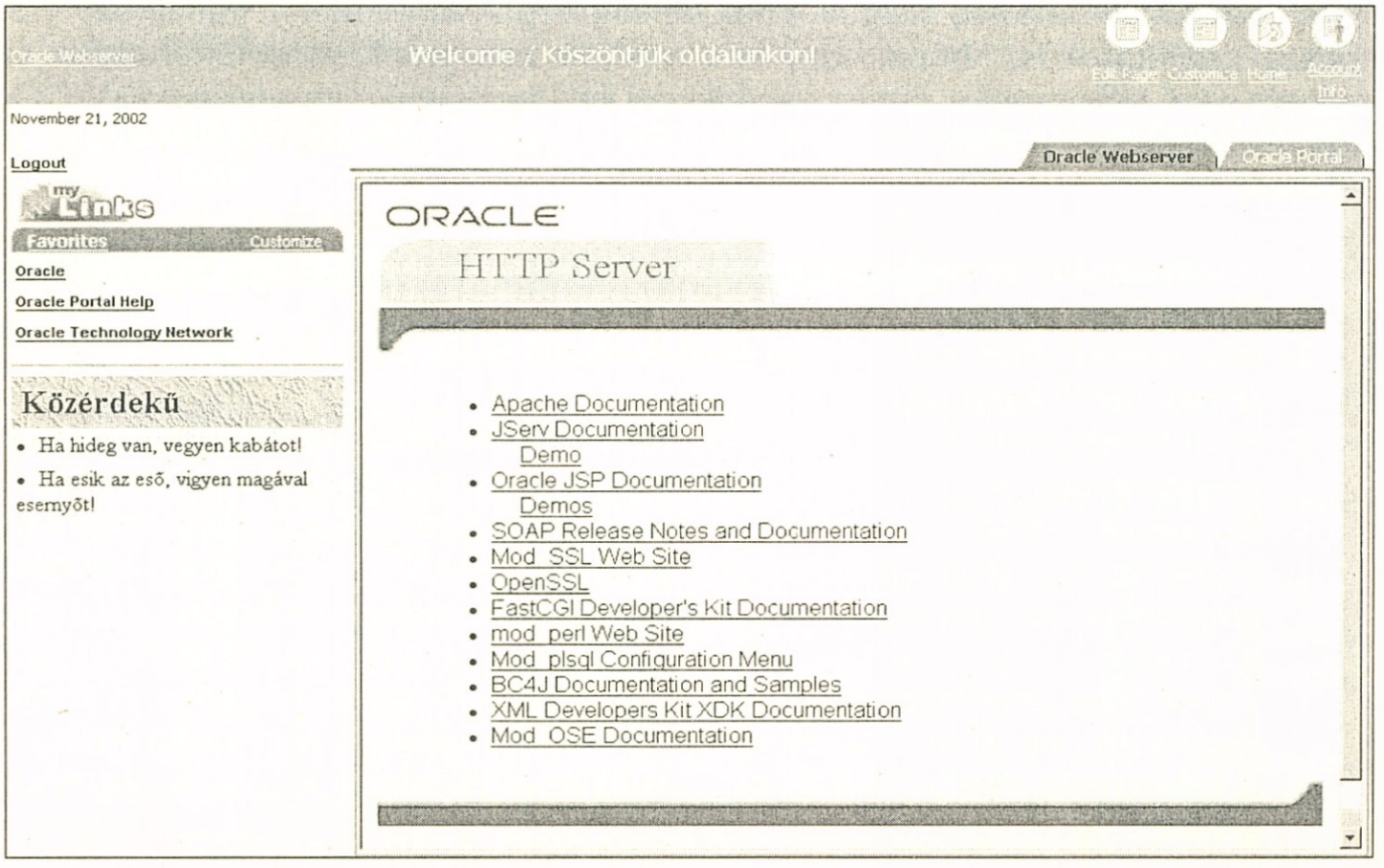
Ha az SSO-t nem csak a Portal-felhasználók hitelesítésére használjuk, akkor megtehetjük, hogy csak SSO-felhasználót hozunk létre, ekkor azonban figyeljünk oda, hogy a felhasználó ne legyen feljogosított Portal-felhasználó!

Felhasználók és felhasználói csoportok kezelésével részletesebben a 4.5. fejezetben foglalkozunk.

4.2.2. Oldal létrehozása

Mivel még nem ismerjük a portált, ezért csak egy nagyon egyszerű oldal létrehozását tűzzük ki célul. Később megismerjük az adatterületeket és alkalmazásokat, ezek komponenseit szintén felhasználhatjuk oldalainkon. A portál eszközeitől független alkalmazásokat is integrálhatunk az oldalakra a PDK használatával (4.6. fejezet).

A létrehozandó oldal képét a 4.13. ábrán láthatjuk. Ezen szerepel: egy testre szabott fejléc, egy beléptető portlet, hasznos linkjeink listája, egy üzenőtábla és egy megadott URL-en található oldal.



4.13. ábra. *Az ORAWEB Nyitóoldal*

Oldalakat kétféleképpen hozhatunk létre:

1. Az Oracle Portal Home Page oldal `Build` lapján válasszuk a `Pages` / *Oldalak* portlet `Create a New Page` / *Új oldal létrehozása* linket.
2. A navigátorban válasszuk a `My Pages` / *Saját oldalak* mellett a `Create` / *Létrehoz* linket.

Mindkét esetben ugyanaz a varázsló vezet végig az oldal paramétereinek megadásán. Általában is igaz, hogy az objektumok létrehozását varázslók segítik. A varázsló oldalain használhatunk mező szintű segítséget. A varázsló mindig kiírja, hogy a folyamatban hol tartunk: `Step 1 of 4` ▶▷▷▷. Eszerint összesen 4 lépés van, és mi az 1. lépésnél tartunk.

Fő jellemzők

Első lépésben meg kell adnunk a lap tulajdonságait a `Page Properties` / *Lap tulajdonságai* varázsló oldalon:

- Adjuk meg az oldal nevét a `Name` / *Név* mezőben. Ez a név a Portal szerkesztői számára látszik, az objektum neve. Az adatbázisbeli objektumok neveihez hasonlóan nem tartalmazhat speciális karaktereket és szóközt sem. Írjuk be: `ORAWEB_NYITO` (a kis- és nagybetűk egyenértékűek).
- Adjuk meg az oldal megjelenített nevét a `Display Name` / *Megjelenített név* mezőben. Ez a név jelenik meg az oldal címeként, az oldal fejlécben és a navigátorban is. Írjuk be: `Oraweb Nyitóoldal`.

A további mezőket hagyjuk változatlanul. Röviden felsoroljuk a további mezők jelentését is:

- `Description` / *Leírás* – az oldalhoz tartozó leírás.
- A `Display Options` / *Megjelenítési opciók* alatt választhatjuk ki, hogy akarunk-e fejlécet, és azt, hogy milyen megjelenítési sablont használunk az oldalon.
- A `Page Caching` / *Gyorsítótár alkalmazás* alatt választhatjuk ki, hogy a rendszer alkalmazzon-e gyorsítótárat az oldal definíciójának, vagy definíciójának és tartalmának tárolására, vagy sem. A tartalom tárolása nem javasolt az erősen dinamikus adatokat megjelenítendő oldalak esetén.

Kattintsunk a `Next` / *Következő* gombra.

Oldalelrendezés és stílus

A `Page Layout And Style` / *Oldalelrendezés és stílus* varázsló oldalon kiválaszthatjuk az oldalelrendezést és a stílust. Az oldalelrendezés, mint azt a 4.2.5. alfejezetben majd látjuk, lényegében egy előkonfigurált és elmentett régióelrendezés. Oldalunk-

hoz válasszuk a `Two Column Layout` / *Kétoszlopos elrendezést* a lenyíló listából. Figyeljük meg, hogy a lenyíló lista alatti bemutató követte a változtatást.

Az oldal stílusa az oldalon megjelenő szövegek betűtípusának, színeinek és az egyéb látványelemek megjelenésének elmentett konfigurációja. Minden felhasználónak lehet *saját alapértelmezett stílusa*, amit minden olyan oldal felvesz, amelynél ez van beállítva `<Use User's Default Style>`. Az oldalunkon ezt állítsuk át `Style for Public Home Page` / *Publikus nyitóoldal stílusa* stílusra a lenyíló listában. Itt is megjelenik egy bemutató.

Kattintsunk a `Next` / *Következő* gombra.

Portletek hozzáadása

Az `Add Portlets` / *Portletek hozzáadása* varázsló oldal a legfontosabb a megjelenő tartalom szempontjából. Itt konfigurálhatjuk be az oldalon megjelenő portleteket, régiókat és a fejlécet is.

A gyakorlatban általában szeretjük folyamatosan ellenőrizni szerkesztéseink eredményét. Most nem alkalmazzuk ezt a technikát, csak megadjuk a lépéseit, hogy később az Olvasó használhassa, ha úgy dönt, hogy ezt a technikát alkalmazza az oldalak létrehozása és szerkesztése során. Ilyenkor átugorjuk ezt a képernyőt, így létrejön és elérhetővé válik az oldal. Megnyitunk egy új böngészőablakot és ott elnavigálunk az oldalra (ami ekkor még természetesen csupasz). Az eredeti ablakban kiválasztjuk a navigátorban vagy a `Pages` / *Oldalak* portletben az oldalt és az `Edit` / *Szerkesztés* linkre, illetve gombra kattintunk. Ezután szerkesztjük az oldalt és minden látni kívánt szerkesztés után az `Apply` / *Érvényesít* gombra kattintunk. Az újonnan nyitott ablak tartalmát frissítve láthatóvá válnak a módosítások. Bánjunk óvatosan ezzel a technikával, mert az érvényesítés után nincs lehetőségünk a változtatások visszavonására!

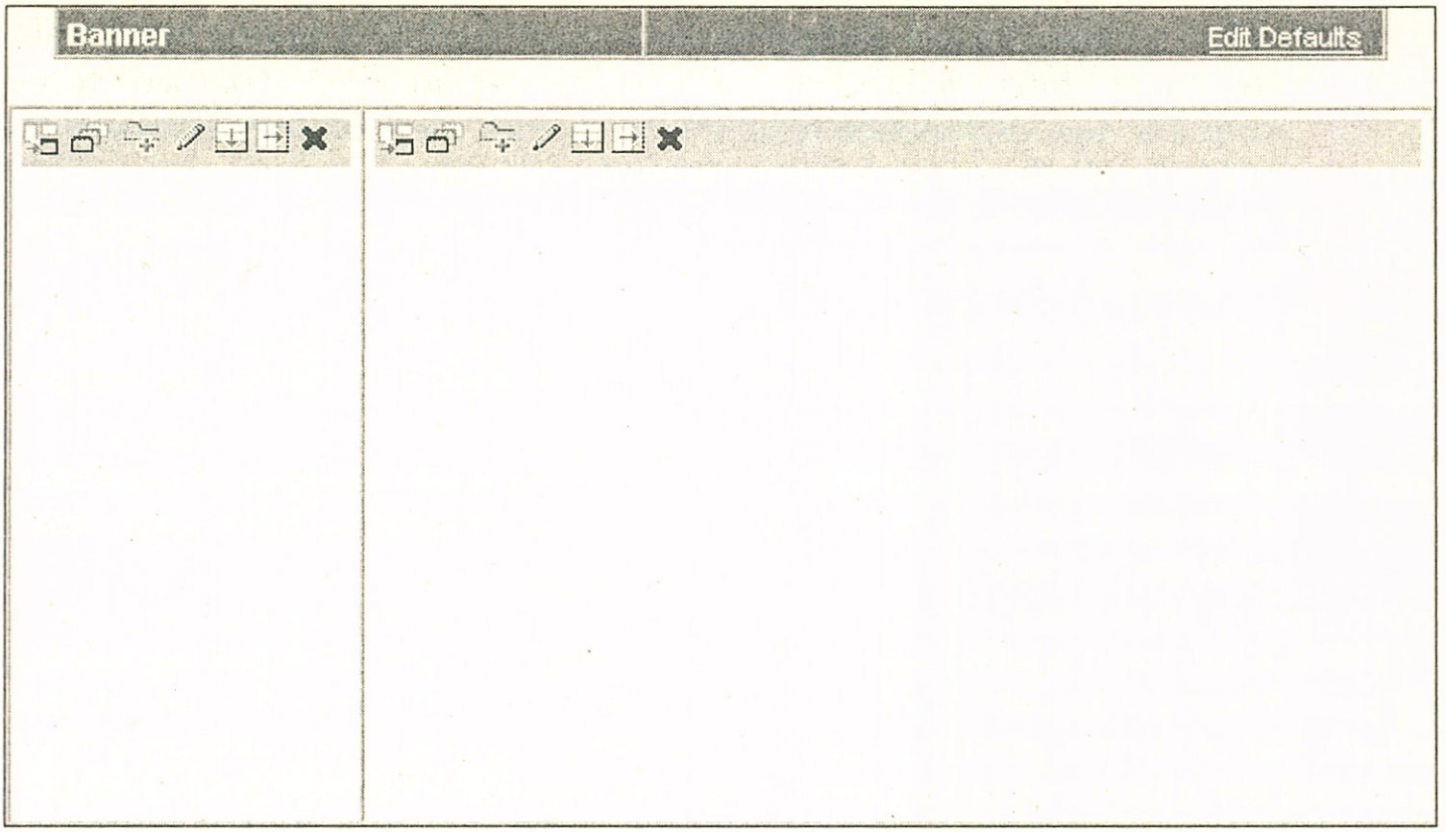
Az oldal szerkesztését kezdjük a fejléccel. Kattintsunk a `Banner` / *Fejléc* portlet `Edit defaults` / *Alapbeállítások szerkesztése* linkre. A továbbiakban alapbeállítások szerkesztése alatt mindig egy ilyen nevű linket értünk majd.

Minden olyan portlet esetén, ahol az alapbeállítások módosíthatók, a portletenként eltérő szerkesztő oldalon adhatjuk meg a beállításokat.

Az `Edit Portlet Banner` / *Fejléc portlet szerkesztése* oldalon írjuk be a `Greeting` / *Üdvözlés* mezőbe: `"Welcome / Köszöntjük oldalunkon!"`. Lehetőség van szöveg megadása helyett kép feltöltésére is.

A `Background Image And Banner Height Settings` / *Háttérkép és fejléc magasságának beállításait* hagyjuk üresen.

A `Banner Logo Settings` / *Fejléc logója beállításainál* a `Label` / *Címke* mezőbe írjuk be `Oracle Webserver`. Az URL-ben vegyük észre a speciális jelentéssel bíró `#HOMEPAGE#` értéket. Ez mindig az aktuális felhasználó főoldalát jelenti. Cseréljük le ezt a linket a `http://`***<szervernév>***`:`***<port>***`/` linkre (például `http://oracleias.unideb.hu:7777/`) linkre. A logo alatt megjelenő kép mellett kattintsunk a piros ✖-re a kép törléséhez. Tölthetünk fel új képet itt is, ilyenkor azonban a szöveg nem jelenik meg.



4.14. ábra. *Fejléc és régiók a szerkesztő ablakban*

A `Banner Links` / *Fejléc linkjei* a fejléc elsődleges linkjei, amelyek címke mellett egy ikonnal is rendelkezhetnek. Most nem adunk meg saját linkeket. A beépített linkek közül válasszuk ki az `Edit Page` / *Oldal szerkesztése*, `Customize Page` / *Oldal testre szabása*, `Home` / *Főoldal*, `Account Information` / *Account információ* linkeket.

A `Secondary Links` / *Másodlagos linkek* az elsődleges linkek alatt jelennek meg és nem rendelkeznek ikonokkal. Ezek közül most ne válasszunk ki egyet se (töröljük a kiválasztottakat).

Kattintsunk az `OK` gombra.

A portletek hozzáadása oldalra tértünk vissza, ahol a fejléc portlet alatt látjuk az oldal elrendezését (4.14. ábra).

A régiókat egy-egy téglalap jelzi, fejlécükön több kis ikont látunk. Ezekre kattintva érhetjük el a következőket:

- *Portlet hozzáadása* – A portletszolgáltatók portletjei közül kiválaszthatjuk a régión megjelenítendő portleteket. Egy régión belül több portletet is elhelyezhetünk, melyek közt lehet több azonos típusú is.
- *Portletek rendezése* – A régió portletjeit rendezhetjük sorba.
- *Fül hozzáadása a régióhoz* – Új fület adhatunk a régióhoz. Minden fülhöz egy lap tartozik, minden lapon a régiók továbboszthatók, szabadon, egymástól függetlenül konfigurálhatók.
- *Fülek rendezése* – Ha már vannak a régión füleink, akkor azok sorrendjét adhatjuk meg.

- *Régió tulajdonságai* – A régió tulajdonságait állíthatjuk be egy felugró ablakban. Itt állíthatjuk be a régió szélességét és azt, hogy a régióban a portletek sorokba vagy oszlopokba rendezve jelenjenek meg, egymástól milyen távol helyezkedjenek el, legyen-e keretük, fejlécük.
- *Régió hozzáadása sorként* – Az eredeti régió alatt az eredetivel megegyező magasságú új régiót hoz létre.
- *Régió hozzáadása oszlopként* – Az egy sorban lévő régiók jobb szélén új régiót hoz létre. Az egy sorban elhelyezkedő régiók szélessége azonos lesz (a sor fele, harmada, negyede stb.).
- *Régió törlése* – Törli a régiót. Ha a régió portleteket tartalmaz, akkor azokat átmozgathatjuk egy másik régióba. Ha vannak a régión fülek, akkor minden fület is töröl.

Előbb a bal, majd a jobb oldali régióhoz adjunk hozzá portleteket. Kattintsunk a portlet hozzáadása ikonra a bal oldali régió fejlécén. A megjelenő oldal két részből áll: bal oldalon a portletszolgáltatók vannak felsorolva, jobb oldalon a régió portletjeit láthatjuk (ez most még üres). A portletszolgáltatók két csoportba oszthatók: `Seeded Providers` / *Kiemelt szolgáltatók* és az `Other` / *Egyéb* szolgáltatók. Ha még nincsenek alkalmazásaink, adatterületeink és külső szolgáltatóink sem a rendszerben, akkor biztosan nem lesznek egyéb szolgáltatóink.

A szolgáltatók valójában egy adatterület mappáiban vannak tárolva és mi most ezt a mappát látjuk. Ez az adatterület a navigátorból az adatterületek közül kiválasztható.

A kiemelt szolgáltatók közül válasszuk a `Login Server` / *Bejelentkezési szerver* szolgáltatót. Most csoportokba rendezve láthatjuk a szolgáltató portletjeit. Válasszuk a `Login` / *Bejelentkezés* portletet. A kattintás hatására a Login portlet megjelenik a jobb oldali listában.

A portletek alatt megtaláljuk az utat, amin a portlethez eljutottunk: `Portlet Repository : Seeded Providers : Login Server`. Kattintsunk a `Seeded Providers` linkre, majd válasszuk az `Oracle Portal` szolgáltatót. Ez a szolgáltató számos portletet nyújt, köztük azokat, amelyekkel már az Oracle Portal Home Page oldalon találkoztunk. Válasszuk most a `Favorites` / *Kedvencek* és a `HTML Portlet` portleteket. Ezek is megjelennek a jobb oldalon a kiválasztott portletek listájában.

A lista elemeit rendezhetjük, ha kiválasztjuk valamelyik portletet, a nyilakkal felle mozgathatjuk, az X-szel törölhetjük. A sorrend módosítására később is lesz lehetőségünk.

A lista elfogadásához kattintsunk az `OK` gombra.

A bal oldali régióban ott vannak a kiválasztott portletek.

A jobb oldali régióba, az előzőekhez hasonlóan, vegyünk fel egy HTML Portlet portletet.

A bal oldali régió fejlécén kattintsunk a tulajdonságok ikonra. A megjelenő ablakban töröljük a `Show Portlet Borders` / *Portletek keretezése* beállítást, majd kattintsunk előbb az `Apply` / *Alkalmaz,* utána a `Close` / *Bezárás* gombra.

Minden portlettől balra van egy üres négyzet. Ezt kipipálva jelölhetjük ki a portleteket műveletek végrehajtására. Hasonló módon, az oldalakon található fülek is kijelölhetők lesznek. A műveletek az oldal közepén látható 4 gombbal érhetők el, ezek:

- `Hide` / *Elrejt* – A kijelölt elemek nem jelennek meg az oldalon, de nem is törlődnek.
- `Show` / *Mutat* – A kijelölt rejtett elemek újra láthatók lesznek az oldalon.
- `Delete` / *Töröl* – A kijelölt elemek törlődnek, nem állíthatók vissza automatikusan.
- `Move` / *Mozgat* – Csak portletek esetén lehetséges, a kijelölt portleteket más régióba mozgathatjuk.

A portletek szerkesztését még folytatjuk, most azonban kattintsunk a `Next` / *Következő* gombra.

Hozzáférés szabályozása

A `Control Access` / *Hozzáférés szabályozása* varázsló oldalon hagyjuk kipipálva a `Display Page To Public User` / *Az oldalt elérheti minden felhasználó* beállítást. Ha ezt töröljük, akkor az oldalt csak azok a bejelentkezett felhasználók érhetik el, akiknek explicit jogosultság adásával ezt megengedjük.

Ha a `Publish As Portlet` / *Portletként publikál* beállítás ki van pipálva, akkor az oldal portletszolgáltatóvá válik, és portletként más oldalakra illeszthető lesz. Mivel ezt az oldalt úgy terveztük, hogy egy egész képernyőt elfoglaljon, nem célszerű más oldalakra beilleszteni, hagyjuk tehát üresen ezt a négyzetet.

A `Grant Access` / *Hozzáférés engedélyezése* szekcióban egy felhasználónak vagy felhasználói csoportnak adhatunk az oldalra explicit hozzáférési jogokat. A létező felhasználók és csoportok listájából választhatunk, ha a `Grantee` / *Feljogosított* mező melletti megfelelő ikonra kattintunk. A lenyíló listából a hozzáférés szintjét választhatjuk ki. A jogosultságok közül a `Manage` / *Kezelés* a legerősebb jogkör, a `View Only` / *Csak megtekintés* a leggyengébb. Minden jogkör magába foglalja a tőle gyengébbeket. A jogosultságok részletes leírását lásd a 4.2.7. alfejezetben.

Az oldalt elérhetővé tettük mindenki számára, ez csak megtekintési jogkört jelent. Engedélyezzük az `ORAWEB` felhasználó számára a `Customization (Add-Only)` / *Testre szabás (csak hozzáadás)* jogosultságot:

- A `Grantee` mezőbe írjuk be: `ORAWEB`. (Használhatjuk a mező melletti  ikont is a felhasználó kiválasztására.)
- A lenyíló listában válasszuk ki a `Customization (Add-Only)` jogosultságot.
- Kattintsunk az `Add` / *Hozzáad* gombra.

Az oldal alján, a `Change Access` / *Hozzáférés módosítása* szekcióban a feljogosított felhasználók és csoportok jogosultságait törölhetjük a piros X-szel, vagy megváltoztathatjuk az engedélyezett hozzáférési szinteket, a lenyíló listában a megfelelőt választva. Itt most a létrehozó felhasználó (nálunk `PORTAL30`) és az `ORAWEB` felhasználó látható ábécé sorrendben.

Elkészítettük az oldalt, kattintsunk a `Finish` / *Befejezés* gombra.

A varázsló használata során bármikor választhattuk volna a `Finish` / *Befejezés* gombot. Ezzel átugorjuk a hátralevő oldalakat, ezeken így az alapértelmezett beállítások lépnek érvénybe.

Megjelent az elkészített oldal. Ez még nem teljesen olyan, mint amit a 4.13. ábrán célul tűztünk ki, mert a portletjeink még nincsenek megfelelően konfigurálva. A következő fejezetben konfiguráljuk az oldal portletjeit.

4.2.3. Oldal szerkesztése

A meglévő oldalakat szerkeszthetjük, ha rendelkezünk megfelelő jogosultsággal. Mivel mi hoztuk létre az oldalt, ezért a `Manage` / *Kezel* jogosultsággal rendelkezünk, bármit megváltoztathatunk az oldalon.

Szerkesszük az Oraweb Nyitóoldalt. Ehhez a navigátorban keressük meg az oldalt a saját oldalak között (`Pages > My Pages`), majd az oldal neve mellett kattintsunk az `Edit` / *Szerkesztés* gombra. A szerkesztő oldalon is 4 fül van, ami az oldal varázsló 4 lépésének felel meg. A `Portlets` / *Portletek* fülön állunk alapértelmezés szerint, mert itt változtathatjuk meg a tartalmat.

A jobb oldali régiót osszuk két részre fülek segítségével. Ehhez kattintsunk a  fül hozzáadása ikonra a jobb oldali régió fejlécén. Két fül jelent meg. Módosítsuk a nevüket a  fül tulajdonságai ikonra kattintva. A bal oldali fül neve legyen: `Oracle Webserver`. A fül tulajdonságai oldalon is két lap van, az általános tulajdonságok mellett a fülhöz tartozó lap hozzáférési szintjét a tartalmazó oldal beállításaitól függetlenül is szabályozhatjuk. A fülek esetén megadhatunk két képet `Active` / *Aktív* és `Inactive Image` / *Inaktív kép*, ilyenkor a szöveg helyett a képek kerülnek a fülekre. Mielőtt az ablakot bezárjuk, kattintsunk az `Apply` / *Alkalmaz* gombra.

A jobb oldali fül feliratát is változtassuk meg, a felirat legyen: `Oracle Portal`.

Igazítsuk a füleket a régió jobb oldalára, ehhez a fülek melletti  régió tulajdonságai ikonra kattintsunk. A `Tab Alignment` / *Fül igazítása* lenyiló listában válasszuk a `Right` / *Jobbra* értéket. Alkalmazzuk a változtatott beállításokat és zárjuk be az ablakot.

Célunk, hogy a fülek lapjain a füleknek megfelelő nevű oldalak jelenjenek meg.

Az `Oracle Webserver` lapon már van egy HTML Portlet portletünk, szerkesszük az alapbeállításokat (`Edit defaults`). Kapcsoljuk ki a `Display Portlet Header` / *Portlet fejléc megjelenítése beállításait* (ezt a régió tulajdonságai közt egy régió minden portletjére is beállíthajuk). A `Display Name` / *Megjelenített név* mezőbe írhatjuk be a portlet fejlécén megjelenített nevet, mivel azonban a fejléc most nem látható, a mező tartalma lényegtelen. A `Content` / *Tartalom* mezőbe írjuk be:

```
<IFRAME width="100%" height="500" src="http://<szervernév>:<port>/">
</IFRAME>
```

A *<szervernév>:<port>* értékét helyettesítsük be a saját beállításunknak megfelelően (például: `oracleias.unideb.hu:7777`). Ellenőrzésképpen kattintsunk a `Preview` / *Bemutató* gombra. Ha nem írtunk el semmit, akkor zárjuk be a bemutató ablakot és kattintsunk az OK gombra a portlet beállításai ablakban.

Hasonló helyzetet akarunk elérni az `Oracle Portal` fül esetében is. A fülre kattintva látjuk, hogy ezen a lapon még nincs egyetlen portlet sem. Vegyünk fel egy HTML Portletet és változtassuk meg az alapbeállításait:

- Kapcsoljuk ki a portlet fejlécét.
- A `Content` tartalma legyen a következő:

```
<IFRAME width="100%" height="500" src="http://<szervernév>:<port>/pls/">
</IFRAME>
```

Itt feltételeztük, hogy a megadott URL tényleg az Oracle Portal URL-je. Ha ez nem így van, akkor változtassuk meg értelemszerűen.

Egy új böngészőablakban ellenőrizhetjük a változtatásainkat: Tényleg kikapcsoltuk a portlet fejlécét? Megjelenik az oldalon az URL tartalma?

Konfiguráljuk a bal oldali régió HTML-portletjét is:

- Kapcsoljuk ki a portlet fejlécét.
- A `Content` tartalma legyen a következő:

```
<hr>
<table cellpadding="6" cellspacing="0" width="100%"
background="/images/banstra2.jpg">
  <tr><td>
    <font size="+2" color="#220066"><b>Közérdekű</b></font>
  </td></tr>
</table>
<table bgcolor="#F0F0FF" cellpadding="3" cellspacing="0"
width="100%">
  <tr><td>
    <li> Ha hideg van, vegyen kabátot!
  </td></tr>
  <tr><td>
    <li> Ha esik az eső, vigyen magával esernyőt!
  </td></tr>
</table>
```

Az itt megadott JPEG képet (`/images/banstra2.jpg`) az Oracle Portal telepíti. Ha másik képet szeretne használni, akkor az Oracle HTTP Server adminisztrátorával egyeztessen a képek elhelyezéséről és elérhetővé tételéről.

Finomítsuk a fejléc beállításait:

- Az üdvözlő szövegét igazítsuk középre. Ehhez a `Greeting` / *Üdvözlés* mezőben cseréljük le a szöveget: `<center>Welcome / Köszöntjük oldalunkon!</center>`. A legtöbb szövegmezőben használhatunk HTML-tag-eket a szöveg igazítására.
- Az Oracle Webserver link két sorban jelenik meg, jelenítsük meg egy sorban. Ehhez a fejléc logója `Label` / *Címke* mezőjének tartalmát javítsuk ki: `Oracle Webserver`. A legtöbb szövegmezőben használhatunk HTML-entitásokat, így kiküszöbölhetjük a szerver és kliens közti különböző karakterkódolásból adódó inkompatibilitási hibákat is.

Az oldalunk végül a 4.13. ábrán látható formába került.

Oldal ellenőrzése

Az oldalt három felhasználó szemszögéből is megnézzük. Elsőként az oldalt elkészítő felhasználót használjuk (nálunk `PORTAL30`). Jelentkezzünk be és navigáljunk az Oraweb Nyitóldalra. A navigációt legegyszerűbben úgy végezhetjük, hogy a `Pages` / *Oldalak* portlet `Name` / *Név* mezőjébe beírjuk: `ORAWEB_NYITO`, majd a `View` / *Megtekintés* gombra kattintunk.

Vegyük szemügyre az oldalt, különös tekintettel a fejlécen látható linkekre, a `Favorites` / *Kedvencek* portlet fejlécére és a `Login` / *Bejelentkezés* portletre.

Mentsük a vágólapra az oldal URL-jét, majd lépjünk ki a bejelentkezés portlet `Logout` linkjére kattintva.

Most a `PUBLIC` felhasználó jogosultságaival látjuk a Portalt mindaddig, míg újra be nem jelentkezünk. Nyissuk meg az oldalt az elmentett URL segítségével.

Vegyük észre, hogy a bejelentkezés portlet megváltozott, a fejlécen nem szerepel minden link és a kedvencek portlet fejlécén sem látható a `Customize` / *Testre szabás* link. Használjuk a bejelentkezés portletet és lépjünk be `ORAWEB` felhasználóként.

A linkek egy része újra megjelent, a bejelentkezés portlet és a kedvencek portlet is a régi formáját öltötte. Az `Edit Page` / *Oldal szerkesztése* linkhez azonban az `ORAWEB` felhasználó nem rendelkezik elég jogosultsággal. (Ahhoz, hogy megjelenjen, legalább a `Manage Style` / *Stílus kezelése* jogosultsággal kellene rendelkeznie.)

Oldalak másolása, mozgatása, törlése, kapcsolt oldalak létrehozása

Már létező oldalakat a navigátorban a `Copy` / *Másolás* linkre kattintva duplikálhatunk. Ilyenkor új névvel kell ellátnunk az oldalt. A létrejövő új oldal ezután az eredetitől függetlenül szerkeszthető, attól különböző jogosultságokkal látható el.

Oldalakat a navigátorban a `Delete` / *Törlés* linkre kattintva törölhetünk.

Bármely oldalhoz létrehozhatunk kapcsolt oldalakat a navigátorban a `Create SubPage` / *Kapcsolt oldal létrehozása* linkre kattintva. Ilyenkor a kapcsolt oldalak és a

szülőoldal közt az oldalak fejlécének középen megjelenő linkekkel tudunk majd navigálni, ha ezt a szülő oldal fő tulajdonságai közt engedélyezzük.

Oldalakat a navigátorban a `Move` / *Mozgatás* linkre kattintva helyezhetünk át. Így kapcsolt oldalból főoldalt is csinálhatunk, egyes oldalakat más oldalakhoz kapcsolhatunk stb.

4.2.4. Legfelső szintű, saját és felhasználói oldalak

Minden az Oracle Portalba bejelentkezett felhasználó hozhat létre oldalakat és azokhoz kapcsolódó objektumokat, ha a `Create` / *Létrehozás* jogosultságot explicit módon nem vettük el tőlük. (Minden bejelentkezett felhasználó tagja az `AUTHENTICATED_USERS` csoportnak, amely rendelkezik az `All Pages : Create` / *Minden oldal : Létrehozás* jogosultsággal.) Egy oldal a navigátorban többféleképpen is elérhető.

A navigátor `Pages` lapján a következő öt link közül választhatunk:

- `Top-Level Pages` / *Legfelső szintű* oldalak – A portáladminisztrátor jogában áll, hogy azokat az oldalakat, amelyek több felhasználó számára is érdekesek lehetnek, a legfelső szintű oldalak közt is elérhetővé tegye, így azok könnyen megtalálhatók a navigátorral. Ha egy oldal még nincs a legfelső szinten, akkor a navigátorban az oldal neve mellett a portáladminisztrátor számára megjelenik a `Make Top-Level` / *Legfelső szinten elérhető.* A legfelső szintű oldalak visszaminősíthetők a mellettük levő `Remove from Top-Level` / *Legfelső szintről visszavon* linkkel.
- `My Pages` / *Saját oldalak* – Az aktuálisan bejelentkezett felhasználó által eddig létrehozott oldalakat láthatjuk, kezelhetjük.
- `User Pages` / *Felhasználói oldalak* – Az egyes felhasználók által létrehozott összes oldalt tartalmazza. Egy konkrét felhasználó oldalait a felhasználó nevének kezdőbetűjét, majd nevét kiválasztva érhetjük el. A saját oldalakat így is elérhetjük, de a `My Pages` link segítségével gyorsabban érhetjük el ezeket.
- `Page Layouts` / *Oldalelrendezések* – Az oldalak létrehozásakor az egyedi stílus kialakítása érdekében gyakran azonos elrendezésű oldalakat hozunk létre. Elrendezés alatt Oracle Portalban egy oldal régiókra történő felosztását értjük. Egy-egy felosztást saját névvel ellátva tárolhatunk és az oldalaink létrehozásakor felhasználhatjuk azokat.
- `Page Styles` / *Oldalstílusok* – Minden oldal rendelkezik egy oldalstílussal, ez határozza meg az oldalakon megjelenő elemek színeit, a betűk típusát, méretét.

Az oldalak rendszerezését röviden a következőképpen foglalhatjuk össze: egy oldal mindig megtalálható a tulajdonos felhasználó oldalai közt. Az aktuális felhasználó oldalait láthatjuk ezenfelül a saját oldalak közt is. A fontosabbnak tartott oldalakat pedig a legfelső szintű oldalak közt érhetjük el nevük szerinti ábécé sorrendben.

4.2.5. Oldalelrendezés és stílus

Az összetartozó portáloldalakat egységes stílussal kell felruháznunk, hogy a felhasználók könnyen tájékozódjanak a különböző oldalakon. Ezt segítik elő az oldalelrendezések és az oldalstílusok.

Oldalelrendezések jellemzői

Az Oracle Portal telepítés után is rendelkezik már néhány elrendezéssel. Ilyen volt például az általunk használt `Two Column Layout` / *Kétoszlopos elrendezés*. A navigátor segítségével létrehozhatunk saját elrendezéseket. Ilyenkor egy varázslón keresztül megadhatjuk az elrendezés nevét (`Display Name`), rövid leírását (`Description`) és azt, hogy az elrendezést mindenki felhasználhatja-e (`Make Public`). Ezután a régiókat konfigurálhatjuk az oldalak létrehozásához hasonlóan. Azonban elrendezések esetén nincs lehetőség fülek és portletek előkonfigurálására.

Már meglévő elrendezéseket szerkeszthetünk, törölhetünk is, ez azonban nem befolyásolja azokat az oldalakat, amelyek létrehozásakor a módosított vagy törölt elrendezést felhasználtuk.

Oldalstílusok jellemzői

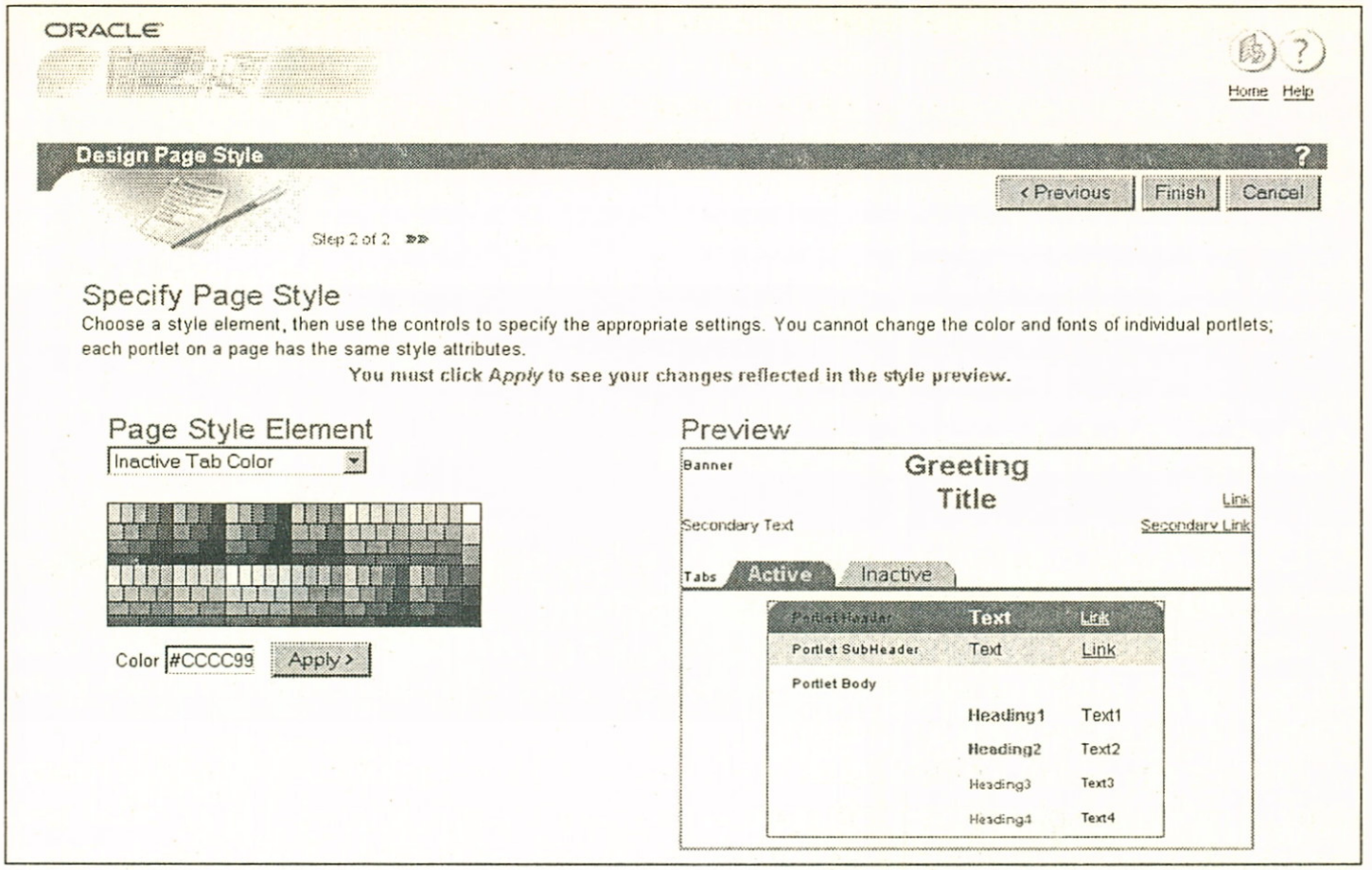
Minden oldal pontosan egy oldalstílussal rendelkezik. Ez az oldalstílus lehet egy konkrét, az oldalhoz rögzített stílus, illetve lehet a felhasználó alapértelmezett stílusa. Minden felhasználóhoz egyedileg, illetve minden felhasználói csoporthoz megadható egy alapértelmezett stílus (a csoport szintű beállítások érvénybe lépéséről lásd 4.5. fejezet). Ezenfelül rendszer szinten is megadható egy alapértelmezett stílus. Így minden felhasználó, még az is, aki nem jelentkezett be, rendelkezik saját stílusbeállítással. Azok az oldalak, amelyekhez a `<Use User's Default Style>` stílusbeállítás van rendelve, mindig az aktuális felhasználó saját stílusát veszik fel és így egységesen jelennek meg számára. Ha a felhasználó alapértelmezett stílusa megváltozik, ezek az oldalak követik a változást, így minden felhasználó, illetve csoport számára egyedi külsővel jelenhet meg ugyanaz az oldal.

A bejelentkezett felhasználó számára alapértelmezett stílust a navigátorban a stílus neve melletti `Make Default` / *Alapértelmezésként beállít* linkre kattintva adhatunk meg.

Csoportok számára a csoport tulajdonságai beállításakor választhatjuk ki az alapértelmezett stílust (lásd 4.5.1. alfejezet).

A rendszer alapértelmezett stílusát a portáladminisztrátor az Oracle Portal Home Page oldal `Administer` / *Adminisztrál* lapján a globális beállítások portletben adhatja meg (lásd 4.5.2. alfejezet).

Oldalstílust a navigátorban varázsló segítségével hozhatunk létre. Meg kell adnunk a stílus nevét (`Display Name`), rövid leírását (`Description`), azt, hogy az új



4.15. ábra. *Oldalstílus elemeinek beállítása*

stílust mindenki felhasználhatja-e (`Make Public`) és a gyorsítótár használatára vonatkozó beállításokat (`Page Caching`). Ezután a stíluselemeket konfigurálhatjuk (4.15. ábra).

A különböző stíluselemeket egy lenyíló listából választhatjuk ki. Az aktuális beállításokat a jobb oldali bemutatón láthatjuk. Minden stíluselem esetén megadható egy szín, ezt a HTML-ben szokásos módon lehet megadni a `Color` / *Szín* mezőben, vagy a palettára kattintva választhatunk. Ezen felül a szöveg jellegű stíluselemek rendelkeznek a szokásos betűtípus beállításokkal:

- `Font Face` / *Betűtípus* – A megadott típusok közül választhatunk.
- `Font Size` / *Betűméret* – A pontokban megadott betűméretet a listából választhatjuk ki.
- `Font Style` / *Betűstílus* – `Regular` / *Szokásos*, `Bold` / *Vastag*, `Italic` / *Dőlt*, `Underline` / *Aláhúzott* és ezek megadott kombinációja lehet.

Ha kiválasztottuk a beállításokat, akkor az `Apply` / *Alkalmaz* gombra kattintva a bemutatón is megjelennek a változtatások.

Már meglévő stílusainkat a navigátorban szerkeszthetjük, másolással duplikálhatjuk.

Stílusokat nem csak a navigátorban hozhatunk létre és szerkeszthetünk. Az oldalak szerkesztésekor a `Style` / *Stílus* lapon új stílust hozhatunk létre egy meglévő má-

solásával, az oldalhoz rendelt stílust a stílus nevével megegyező linkre kattintva szerkeszthetjük.

Ha egy stílust módosítunk, akkor minden oldal, amely a stílust használja követi a változtatásokat (ellentétben az oldalelrendezésekkel, amelyek változtatása már nem befolyásolja az oldalak megjelenését).

Ha egy konkrét stílust törlünk, akkor minden oldalnál, ahol a törölt stílus van beállítva, a stílus átállítódik a rendszer alapértelmezett stílusára. (A felhasználók esetén sajnos nincs automatikus stílusátállítás!)

4.2.6. Oldal testre szabása

Az Oracle Portal oldalainak kialakításakor fontos szempont a testre szabhatóság. A testre szabás nem keverendő össze a szerkesztéssel.

Legegyszerűbb úgy gondolni egy oldalra, hogy annak több változata létezik. Létezik egy közös, publikus változat és emellett az egyes felhasználóknak lehetnek testre szabott privát változataik az oldalról.

Amikor egy oldalt szerkesztünk (`edit page`), akkor a publikus változatot módosítjuk, a módosítások minden felhasználó számára láthatók lesznek. Ha az oldalt testre szabjuk (`customize`), akkor a saját privát változatunkat módosítjuk, a módosítások csak a testre szabást végző felhasználó számára lesznek láthatók.

Oldal testre szabásához az oldalra legalább a `Customization (Add-Only)` / *Testre szabás (csak hozzáadás)* jogosultsággal kell rendelkeznünk. Ezzel a jogosultsággal azonban nem rejthetünk el az oldalon portleteket és füleket. Ehhez már `Customization (Full)` / *Testre szabás (teljes)* jogosultságra van szükség.

4.2.7. Oldalakra adható jogosultságok

Az oldalakra a következő jogosultságok adhatók:

- `Manage` / *Kezel* – A feljogosított tetszőleges módon megváltoztathatja (szerkesztheti, testre szabhatja) az oldal privát és publikus változatát is, beleértve ebbe a következőket:
  - az oldal fő jellemzőinek (Cím, Név, Leírás) szerkesztése;
  - portletek és fülek hozzáadása, törlése;
  - portletek és fülek elrejtése, megjelenítése (`Hide`, `Show`);
  - régiók hozzáadása, törlése;
  - stílus megváltoztatása;
  - hozzáférés szabályozása.

  Automatikusan rendelkezik ezzel a jogosultsággal a létrehozó és azok, akik az `All Pages : Manage` / *Minden oldal : Kezelés* rendszerprivilégiummal rendelkeznek (adatbázis-adminisztrátorok, portáladminisztrátorok).

- `Edit Contents` / *Tartalom szerkesztése* – A feljogosított megváltoztathatja az oldal privát és publikus változatát is (szerkesztheti, testre szabhatja) a következő módon:
  - portlet hozzáadása létező régiókhoz;
  - portletek és fülek hozzáadása, törlése;
  - portletek és fülek elrejtése, megjelenítése (`Hide`, `Show`);
  - stílus megváltoztatása.

  Noha a jogosultság erre utal, nem lehetséges ezzel a jogosultsággal:
  - régiók hozzáadása, törlése.
- `Manage Style` / *Stílus kezelése* – A feljogosított megváltoztathatja az oldal privát és publikus változatának stílusát, és teljes mértékben testre szabhatja a privát változatot.
- `Customization (Full)` / *Testre szabás (teljes)* – A feljogosított megváltoztathatja az oldal privát változatát (a publikust nem) a következő módon:
  - portletek és fülek elrejtése, megjelenítése (`Hide`, `Show`);
  - portlet hozzáadása (amit csak a feljogosított lát ezután).
- `Customization (Add-only)` / *Testre szabás (csak hozzáadás)* – A feljogosított megváltoztathatja az oldal privát változatát (a publikust nem) a következő módon:
  - portlet hozzáadása létező régiókhoz (amit csak a feljogosított lát ezután);
  - a feljogosított által hozzáadott portletek törlése.

  Ezzel a jogosultsággal nem lehetséges még a privát oldalon sem:
  - régiók létrehozása, törlése;
  - azon portletek elrejtése, megjelenítése, amelyeket nem a feljogosított hozott létre;
  - portletek törlése.
- `View Only` / *Csak megtekintés* – A feljogosított az oldalt eléri, beállíthatja saját nyitólapnak, másolatot készíthet róla. A másolat a másolást végző tulajdonába kerül, arra `Manage` / *Kezel* jogosultságokkal rendelkezik, tetszőleges módon megváltoztathatja azt.

4.3. Adatterületek

Az adatterületek összetartozó adatok rendszerezett tárolására szolgálnak. Jó példa az adatterületre a beépített on-line súgó. Ebben a fejezetben létrehozunk egy példaadatterületet egy tanintézet és az oktatók közérdekű adatainak tárolására.

Adatterületet kétféleképpen hozhatunk létre:

1. Az Oracle Portal Home Page oldal `Build` lapján válasszuk a `Content Areas` / *Adatterületek* portlet `Create a New Content Area` / *Új adatterület létrehozása* linket.

2. A navigátorban válasszuk a `Content Areas` / *Adatterületek* lapon a `Create New... Content Area` / *Új adatterület létrehozása* linket.

Az *Adatterület varázsló* egyetlen oldalán adjuk meg az új adatterület adatait:

- A `Name` / *Név* mezőbe írjuk: `ORAWEB_TTK`.
- A `Display Name` / *Megjelenített név* mezőbe írjuk: `Oraweb Természettudományi Kar`.
- A `Default Language` / *Alapértelmezett nyelv* lenyíló listában válasszuk ki az `English` / *Angol* nyelvet. A példában nem a magyar nyelvet választjuk, mert nem feltételezzük a magyar nyelvi támogatás telepítését az Olvasó Portal rendszerében.
- A `Content Area Administrator` / *Adatterület-adminisztrátor* mezőbe írjuk be a saját felhasználónk nevét (ha az még nincs beírva). A felhasználót a mező melletti ikonra kattintva egy listából is kiválaszthatjuk.

Az *Adatterület-adminisztrátor* feladata, hogy az adatterület szerkezetét kialakítsa, igény szerint módosítsa. Az *adatterület stílusadminisztrátorának* feladata a megfelelő stílus kialakítása. A felhasználók feladata az adatterület feltöltése, megtekintése, használata szerepkörüktől függően.

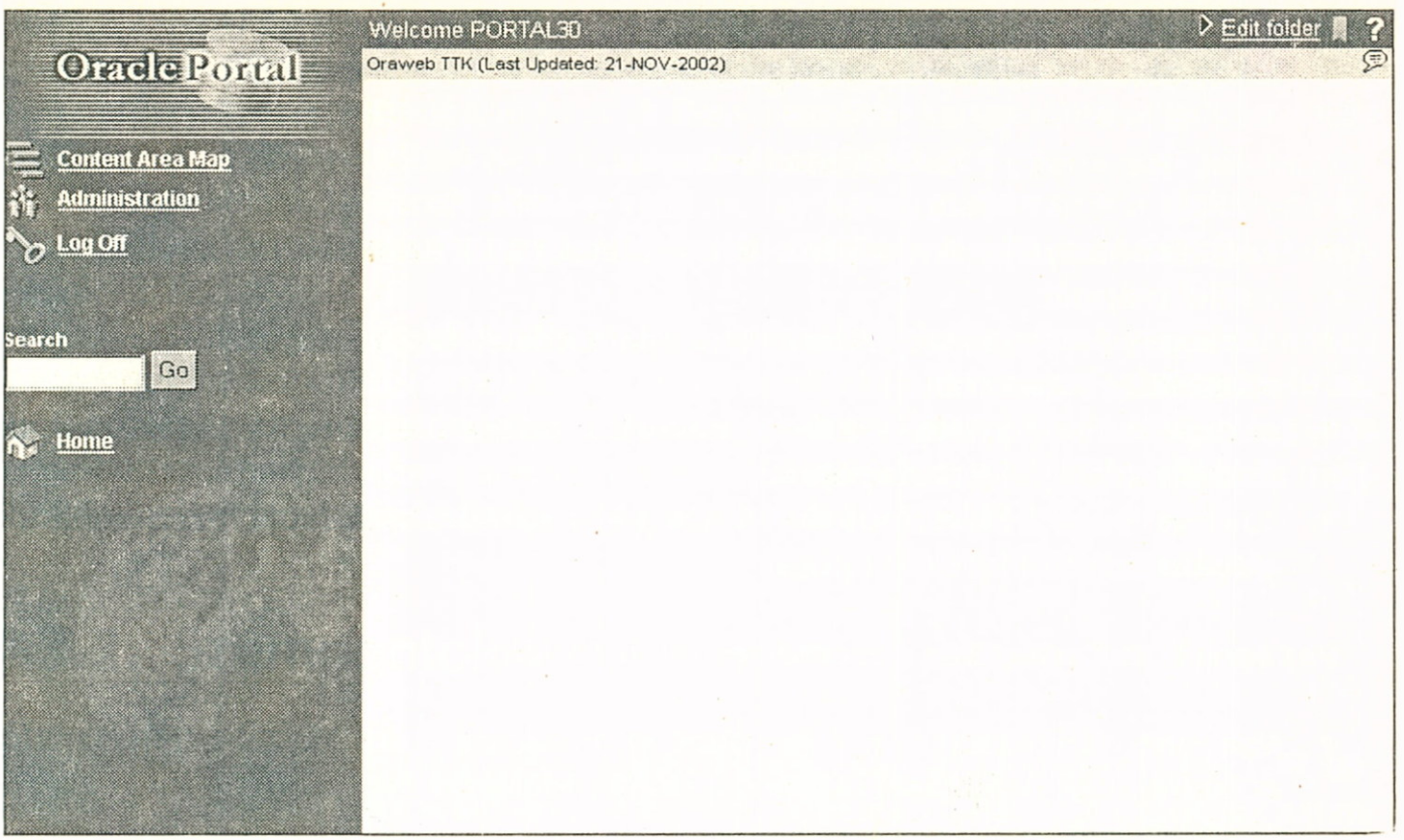
Kattintsunk a `Create` / *Létrehoz* gombra. Az oldalon megjelenik egy üzenet, mely tartalmazza az `Oraweb Természettudományi Kar` linket (az ékezetes karaktereket a rendszer a megfelelő HTML-entitással helyettesítheti), erre kattintva az adatterület további tulajdonságait konfigurálhatjuk. Ezen tulajdonságok áttekintésével a 4.3.5. alfejezetben foglalkozunk. Most kattintsunk a `Bezárás` / *Close* gombra.

Az adatterület mellett a következő parancslinkek (`actions`) vannak:

- `Contents` / *Tartalom* – A tartalmazott objektumokat listázza típusuk szerint.
- `Edit Properties` / *Tulajdonságok szerkesztése* – Az adatterület egészére vonatkozó beállításokat végezhetjük el, lásd még a 4.3.5. alfejezetben.
- `Delete` / *Törlés* – Az adatterület törlésére szolgál, ilyenkor minden, az adatterületen tárolt objektum törlődik.
- `Edit Root Folder` / *Nyitómappa (gyökérkönyvtár) szerkesztése* – A nyitómappa minden adatterület speciális mappája. Ez a mappa nem található meg a többi mappa közt, ezért tulajdonságainak beállítására csak itt van lehetőség.
- `Copy Root Folder` / *Nyitómappa másolása* – A nyitómappa másolható, ekkor az új mappa egy közönséges mappa lesz.

Kattintsunk az adatterület nevére, hogy az üres adatterületre kerüljünk (4.16. ábra).

Az adatterület képernyőjének bal oldalán a navigációs sáv, jobb oldalán az adatterület mappái helyezkednek el. Itt most a nyitómappát látjuk, amely az adatterület egyetlen mappája jelenleg. A mappák közt a tartalmazásból adódó hierarchia szerint navigálhatunk, a navigációs sáv lehetővé teszi, hogy ettől eltérően jussunk el a keresett információhoz.



4.16. ábra. *Oraweb Természettudományi Kar üres adatterület*

Az adatterület szerkezetének kialakítását a következő sorrendben hajtjuk most végre:

- Megadjuk az adatterület kategóriáit és szempontjait, mivel minden tartalmi elemet, beleértve a mappákat is, valamilyen kategóriába kell sorolni és egy vagy több szempontot lehet hozzájuk társítani.
- Létrehozzuk az adatterület néhány mappáját.
- Konfiguráljuk a navigációs sávon megjelenő elemeket.
- Áttekintjük az adatterület egészére vonatkozó beállításokat.
- Az adatterületre helyezünk néhány adatelemet.

Az adatterület adminisztrálásához használhatjuk a navigátort, vagy a navigációs sáv `Administration` / *Adminisztráció* linkjét. Kattintsunk most az utóbbira.

Az adminisztrációs eszközök a következők:

`Content Area Managers` / *Adatterület-menedzserek:*

- `Navigate Content Area` / *Adatterület a navigátorban* – A navigátor oldal adatterületek lapja.
- `Content Area` / *Adatterület* – Az adatterület egészére vonatkozó beállítások.
- `Style` / *Stílus* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület stílusai.

- `Navigation Bar` / *Navigációs sáv* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület navigációs sávjai.

`Content Managers` / *Tartalommenedzserek:*

- `Folder` / *Mappák* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület mappái.
- `Category` / *Kategóriák* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület kategóriái.
- `Perspective` / *Szempontok* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület szempontjai.
- `Custom Type` / *Saját típusok* – A navigátor adatterületek lapja, ezen belül az aktuális adatterület saját típusai.

`Personal Information` / *Személyes információk:*

- `Account information` / *Account információk* – Az aktuális felhasználó accountjának beállításai (azonos a portáloldalakon megtalálható megegyező nevű linkkel).

Ezek az eszközök így gyorsabban érhetők el, mint a navigátorból, éppen ezért sok adminisztrációs feladat elvégzésekor célszerű ezt az oldalt egy böngészőablakban nyitva tartani és a linkeket új ablakokban megnyitni.

4.3.1. Kategóriák

Az adatterületen található tartalmi elemeket és mappákat a kereshetőség érdekében kategóriákba soroljuk. Minden elem és mappa pontosan egy kategóriába sorolandó, így a kategóriák diszjunkt halmazokba foglalják az adatterület elemeit.

A kategóriák a „Milyen témához kapcsolódnak az elemek?" kérdésre adott válaszok. Legyenek nálunk a kategóriák: `Tananyagok`, `Publikációk`, `Hírek`. Ezenfelül mindig létezik a `General` / *Általános* kategória is, amely nem törölhető és nem is módosítható. Ezt használjuk majd, ha olyan elemet hozunk létre, amely egyetlen saját kategóriánkba sem illik bele.

Kategóriák létrehozásához használjuk a navigátort vagy a navigációs sávról elérhető adminisztrációs eszközök táblázatát.

A navigátorban az út most: `Content Areas > Oraweb Természettudományi Kar > Categories` kell legyen. Kattintsunk a `Create New... Category` / *Új kategória létrehozása* linkre.

A továbbiakban ha valamit létre kell hozni a navigátorban, mindig a `Create New...` utáni megfelelő linkre kattintsunk.

A `Category Properties` / *Kategória tulajdonságai* oldalon írjuk be a kategória nevét (`Name`) és megjelenített nevét (`Display Name`) is: `Tananyagok`. Ezután kattint-

sunk a `Create` / *Létrehoz* gombra. A kategóriák neve nem tartalmazhat szóközt, ékezetes és speciális karaktereket, a kis- és nagybetűk egyenértékűek.

Az előzőhöz hasonló módon hozzuk létre a további kategóriáinkat is: `Publikációk`, `Hírek`, a név mezőben ne használjunk ékezeteket.

A kategóriáknak lehetnek alkategóriáik is, így kategóriahierarchia alakítható ki. Alkategóriát a navigátorban a kategória neve melletti `Create` / *Létrehoz* linkkel hozhatunk létre.

A kategóriák rendelkeznek további tulajdonságokkal, ezeket a nevük melletti `Edit` / *Szerkesztés* linkkel változtathatjuk meg. Ezek közé a tulajdonságok közé tartozik az, hogy a kategóriák portletként elérhetők-e vagy sem. A kategóriákhoz több képet is rendelhetünk, melyek a navigációs sávon és a kategóriák mappájában jelennek meg. Erről bővebben a mező szintű súgóból kaphatunk információt.

4.3.2. Szempontok

A szempontok a kategóriáktól abban különböznek, hogy az elemekkel nem 1-*N*, hanem 1-*M* opcionális kapcsolatban állnak. Nem kötelező tehát az elemekhez szempontot rendelni, ugyanakkor egy elemhez több szempont is rendelhető.

A navigátor vagy az adminisztrációs eszközök táblázatának segítségével hozzuk létre a következő szempontokat: `Oktatóknak`, `Hallgatóknak`, `Tantárgyakról`, ezen belül: `Programozás`, `Adatbázisok`.

Itt is ügyeljünk arra, hogy a név mezőben ne használjunk ékezeteket. Figyeljünk oda, hogy a `Programozás` és az `Adatbázisok` szempontokat a `Tantárgyakról` szempont alszempontjaiként hozzuk létre.

A szempontok – a kategóriákhoz hasonlóan – rendelkeznek még néhány tulajdonsággal, melyek a navigátorból elérhetők. Így megadhatók hozzájuk képek és portletként portáloldalakon is megjeleníthetők.

4.3.3. Mappák

Egy böngészőablakban navigáljunk az adatterület képernyőjére (4.16. ábra). A képernyő jobb oldalán találhatók a mappák, most a nyitómappát látjuk. A mappákat tartalmazó régió fejlécén található:

- az aktuális felhasználót köszöntő üzenet (nálunk: `Welcome PORTAL30`);
- az `Edit folder` / *Mappa szerkesztése* link, feltéve hogy rendelkezünk a szükséges jogosultságokkal a mappára (a létrehozó, az adatterület adminisztrátora és a portáladminisztrátor mindig rendelkezik ezzel a jogosultsággal);
- a  *hozzáadás kedvencekhez* ikon, ha erre kattintunk, akkor a kedvencek portletben megjelenik majd egy link, amellyel erre az adatterületre és mappába ugorhatunk;

- a ? *mező szintű súgó* ikonja;
- az aktuális mappa neve és utolsó módosításának dátuma;
- a mappához tartozó *tulajdonságok lapot* megnyitó ikon, ilyen linkkel az elemek is rendelkeznek majd.

Kattintsunk a mappa szerkesztése linkre, hogy az adatterületet szerkesztés módban érjük el.

Az oldal megváltozott, számos szerkesztő link jelent meg a navigációs sávon és a mappáknál is. Az `Edit folder` link most `View folder` / *Mappa megtekintése* feliratú lett, ezzel kapcsolhatunk vissza szerkesztői módból. A további linkek közül vegyük sorra a mappákhoz tartozó szerkesztő linkeket:

- `Add item` / *Elem hozzáadása* – Az aktuális mappához ad hozzá egy elemet.
- `Create Folder` / *Mappa létrehozása* – Az aktuális mappán belül új mappát hoz létre.
- `Folder Properties` / *Mappa tulajdonságai* – Az aktuális mappa beállításait változtathatjuk meg.
- `Folder Style` / *Mappa stílusa* – A mappa stílusát adhatjuk meg, ha az különbözik az adatterület egészén használt stílustól.
- `Item Bulk Action` / *Együttes művelet elemekre* – Több elemen hajthatunk végre együttesen egy műveletet (törlés, másolás, mozgatás, kategória vagy szempont megadása stb.).
- `Administration` / *Adminisztráció* – Az adminisztrációs eszközök táblázata.

Hozzunk létre egy új mappát, kattintsunk a megfelelő linkre.

A mappák létrehozását segítő varázslóban első lépésként az új mappa típusát kell megadnunk, ez a következők egyike lehet:

- `Container` / *Konténer* – A mappa elemeket és további mappákat tartalmaz.
- `URL` – A mappa tartalma a megadott URL tartalma lesz, ha a mappára kattintunk, a megadott URL-re jutunk (lényegében egy linkről van szó).
- `Search` / *Keresés* – Egy keresési feltételt adhatunk meg, amikor a mappát megnyitjuk ez a keresés végrehajtódik és az eredményoldalt kapjuk.
- `PL/SQL` – A mappa megnyitásakor a megadott PL/SQL-kód fut le és ennek eredményét tartalmazza a mappa.

Ha vannak saját mappatípusaink, akkor azok is megjelennek. Saját mappatípusokkal valamilyen meglévő mappatípushoz adhatunk meg további tulajdonságokat.

Válasszuk ki a `Container` típust és kattintsunk a `Next` / *Következő* gombra.

Az új mappa neve: `Oktatók`. Ügyeljünk, hogy a név nem tartalmazhat ékezeteket itt sem, de a megjelenített név igen. Kategóriaként hagyjuk a `General`-t kiválasztva.

Kattintsunk a `Finish` / *Befejezés* gombra.

Megjelent a `Sub-folders` / *Almappák* szekció, ahol a mappa almappái vannak felsorolva. Mivel szerkesztői módban vagyunk, a mappa neve mellett a következő szerkesztői eszközök is megtalálhatók:

- a mappa tulajdonságait szerkeszthetjük;
- új mappát hozhatunk létre a mappában;
- törölhetjük a mappát és tartalmát;
- a mappát és tartalmát áthelyezhetjük;
- másolatot készíthetünk a mappáról és tartalmáról.

Hozzunk létre még egy mappát az aktuális mappában az előzőhöz hasonló módon `Tanulmányi Osztály` névvel (a mappa nevében nem lehet ékezetes karakter és szóköz).

Lépjünk be az `Oktatók` mappába és ott hozzunk létre egy mappát `dr. Kiss István` néven.

Még egy oktató oldalát szeretnénk mappaként létrehozni, de ez az oktató vendégoktató, van már saját weboldala, ezt kívánjuk itt is elérhetővé tenni:

- az új mappa `URL` típusú;
- a név `dr. John Smith`;
- a név mellett egy URL-mező is szerepel, írjuk be: `http://foo.edu/~jsmith`.

Minden oktató mappája érdekes lehet a hallgatóknak, ezért mindkét mappára hajtsuk végre a következőt:

- válasszuk a mappa tulajdonságainak szerkesztése ikont;
- az opcionális fülön válasszuk ki a `Hallgatóknak` szempontot a listából és a > nyíl segítségével aktiváljuk (átkerül a jobb oldali listába);
- kattintsunk az `OK` gombra.

Az előbb láthattuk, hogy a mappák a nevükön kívül sok további tulajdonsággal is rendelkeznek. Ezekkel részletesen nem foglalkozunk, csak felsoroljuk azokat, amelyek minden mappa esetén megadhatók:

- `Required` / *Kötelező* – Ezen a lapon kell megadnunk a mappa nevét, kategóriáját és a mappa típusától függően esetlegesen további attribútumokat is.
- `Optional` / *Opcionális* – Itt megadhatjuk a következőket: a mappában elhelyezett elemekre alkalmazunk-e verziókezelést, a mappa tulajdonosának e-mail címét, gyorsítótár alkalmazását engedélyezzük-e, mely almappákra mutató linkek jelenjenek meg a mappában, milyen szempontokhoz rendeljük hozzá a mappát.
- `Style` / *Stílus* – Az adatterület minden mappája egyedi stílussal rendelkezhet, ezt választhatjuk ki a meglévő stílusok közül, szerkeszthetjük a kiválasztott stílust, vagy új stílust hozhatunk létre. Minden létrejövő új mappa a létrehozás-

kor a szülő mappa stílusát örökli, ezután viszont a két mappa stílusa egymástól függetlenül szabályozható.

- `Image` / *Kép* – Képeket adhatunk meg a mappához, ezek egy része akkor jelenik meg, ha a mappát a navigációs sávon linkként is fel szeretnénk tüntetni, valamint megadható egy fejlécként használt kép is.
- `Nav Bar` / *Navigációs sáv* – Minden mappa esetén megadható egy navigációs sáv, ami akkor jelenik meg, ha ez a mappa az aktuális.
- `Access` / *Hozzáférés* – Minden mappa rendelkezik egy *hozzáférési listával* (ACL), amin az egyes felhasználók és csoportok jogosultságait állíthatjuk be. Ezek a jogosultságok (az oldalakétól eltérően) nem tartalmazó jellegűek, ezek a következők: `Own Folder` / *Mappa tulajdonosa*, `View Content` / *Tartalom megtekintése*, `Edit Style` / *Stílus szerkesztése*, `Manage Items` / *Elemek kezelése*, `Create With Approval` / *Létrehozás jóváhagyással*. Ez utóbbi azt jelenti, hogy a feljogosított által felvitt adatelemek csak akkor jelennek meg a mappában ténylegesen, ha a mappa tulajdonosa azt jóváhagyja. Az ACL-en kívül megadhatjuk: a mappa mindenki számára elérhető legyen-e, a konténer típusú mappáknál az adatelem szintű hozzáférés szabályozást. A mappa beállításait az almappák beállításaihoz hozzáadhatjuk vagy azokat lecserélhetjük.

4.3.4. Navigációs sávok

Alapértelmezés szerint bal oldalon helyezkedik el a navigációs sáv. Ez egy olyan oszlop, amiben különböző formában megjelenő linkek helyezkednek el. A linkek segítségével különböző mappákat érhetünk el gyorsan, vagy az adatterületen kívüli oldalakra ugorhatunk. A navigációs sávon elemeket helyezünk el, amelyek egyenként konfigurálhatók, törölhetők.

A navigációs sávot a mappákkal egy időben szerkeszthetjük, ha a mappák régió fejlécén az `Edit Folder` / *Mappa szerkesztése* linkkel szerkesztés módba váltunk.

Ekkor a navigációs sávon a következő szerkesztői linkek jelennek meg:

- `Add Element` / *Elem hozzáadása* – Új elemet adhatunk a navigációs sáv aljára egy varázsló segítségével.
- `Nav Bar Properties` / *Navigációs sáv tulajdonságai* – Megadhatjuk a navigációs sáv tulajdonságait: a navigációs sáv nevét, a tartalmazott elemek igazítását, a navigációs sáv mindenki által elérhető-e (felhasználható mások által), portletként felhasználható-e portáloldalakon. Az oldal füleire kattintva további beállításokat is megváltoztathatunk (stílus, tartalmazott elemek). Ez az oldal a navigátorból is elérhető.
- `Nav Bar Style` / *Navigációs sáv stílusa* – Kiválaszthatjuk a navigációs sávon használt stílust, szerkeszthetjük azt, vagy új stílust hozhatunk létre. Ugyanaz az oldal, mint az előző link esetében, csak másik fül aktív.
- `Choose A Nav Bar` / *Navigációs sáv kiválasztása* – Az aktuális mappához társított navigációs sávot választhatjuk ki.

A linkeken kívül a navigációs sáv minden eleme mellett megtaláljuk a szerkesztői eszközök ikonjait:

- új elemet vehetünk fel az aktuális navigációs sávra az adott elem után;
- a navigációs elemet konfigurálhatjuk;
- az elemet törölhetjük a sávról;
- az elemet az elemek sorrendjében föntebb mozgatjuk (csak több elem esetén jelenik meg).

Az alapértelmezésként létrejött navigációs sávon egyetlen elem található. Most kialakítjuk a 4.17. ábrán látható navigációs sávot.

4.17. ábra. *Oraweb TTK navigációs sáv*

- Szerkesszük a meglevő elemet a elem konfigurációja ikonra kattintva. Jobb oldalon láthatjuk az elemen megjelenő alapelemeket. Töröljük, illetve rendezzük az alapelemeket úgy, hogy végül a következők maradjanak (ebben a sorrendben):

`Content Area Name`	Az adatterület neve, erre kattintva az adatterület nyitóoldalára jutunk vissza (a Content Area Logo képre kattintva ugyanez történik).
`--BLANK LINE--`	Egy üres sor.
`Portal Home`	A portál aktuális felhasználó szerinti nyitóoldala.
`Administration Link`	Az adminisztrációs eszközök táblázata.
`Log On/Log Off Link`	Be-/kijelentkezés linkje.
`--BLANK LINE--`	Újabb üres sor.
`Basic Search`	Egyszerű keresési mező, a beírt szót keresi az adatterület egészén.

Kattintsunk az OK gombra.

- Vegyünk fel egy új elemet (kattintsunk az elem hozzáadás ikonra). Az Element Type / *Elem típusa* lenyíló listából válasszuk ki: Categories / *Kategóriák*. A következő oldalon a Label / *Címke* mezőbe írjuk be: "Válasszon témát!". A bal oldali listából a General kategórián kívül mindhármat vigyük át a jobb oldali listába a > nyíl segítségével. A Display Mode / *Megjelenítési mód* lehetőségei közül válasszuk a Drop-down List / *Lenyíló lista* megoldást. Kattintsunk a Finish / *Befejezés* gombra.
- Az előző ponthoz hasonlóan vegyük fel a szempontokat új elemként. Itt az elem típusa Perspectives / *Szempontok* lesz. A címke mező maradjon üresen. Válasszuk ki a következő szempontokat és a nyíl segítségével aktiváljuk őket: Tantárgyakról, Oktatóknak, Hallgatóknak. A megjelenési mód maradjon Default / *Alapértelmezett*.
- Végül egy új elemben felrakjuk a Content Area Map / *Adatterület térképe* linket. Az új elem típusa legyen Basic Elements / *Alapelemek*. Válasszuk ki a következő alapelemeket és a nyíl segítségével aktiváljuk őket: --BLANK LINE--, Content Area Map. Kattintsunk a Finish / *Befejezés* gombra.

A mappa fejlécén a View Folder / *Mappa megtekintése* linkkel kapcsoljuk ki a szerkesztés módot.

Navigációs sávokon az előzőekben megismert elemtípusokon kívül lehetőség van még képtérképre (image map), mappákra, tetszőleges URL-re mutató linkek és egyszerű szöveges elemtípusok használatára.

4.3.5. Adatterületek tulajdonságainak beállítása

Miután az adatterület építőelemeivel jobban megismerkedtünk, áttekinthetjük az adatterületek általános tulajdonságait. A beállítások oldalához az adminisztrációs eszközök táblázatából vagy a navigátorból is eljuthatunk.

Az oldalon a tulajdonságok hét fül alatt vannak csoportosítva.

1. Main / *Fő tulajdonságok*
   - Basic Content Area Properties / *Adatterület alaptulajdonságai* – Itt adhatjuk meg az adatterület nevét (Name), megjelenített nevét (Display Name) és az adatterület adminisztrátorának e-mail címét (Contact E-mail).
   - Content Area Quota / *Adatterület kvótája* – Az adatterületen elhelyezett adatelemek által maximálisan elfoglalható hely (Maximum Space) méretét szabályozhatjuk, illetve az eddig elhelyezett elemek összméretéről kaphatunk információt (Used Space).

- `Folder Properties` / *Mappabeállítások* – A mappákra általánosan vonatkozó beállítások. Itt engedélyezhetjük, hogy a mappatulajdonosok megváltoztathassák a saját mappáik stílusát (egyébként csak az adatterület vagy a stílusadminisztrátor adhatja meg a stílust) és azt, hogy a mappák elérését gyorsítótár segítse.
- `Content Area Logo` / *Adatterület logója* – Feltölthetünk egy képet az adatterület logójaként, vagy törölhetjük a meglevőt a piros X-re kattintva.

2. `Objects` / *Objektumok*
Az adatterületen az adatterületet tartalommal feltöltő felhasználók által elérhető kategóriákat (`Categories`), szempontokat (`Perspectives`), adatelemek típusát (`Item Types`), mappatípusokat (`Folder Types`) adhatjuk meg.

3. `Items` / *Adatelemek*
- `Item Versioning` / *Adatelemek verziókezelése* – Az adatterületen tárolt adatelemekre három verziókezelési szint írható elő: `None` / *Nincs*, `Simple` / *Egyszerű*, `Audit` / *Követés*. Az első esetben nincs verziókezelés, bármely elem lecserélésekor a korábbi verzió törlődik. Az egyszerű verziókezelésnél a felvitelkor dönthetünk a korábbi verzió megőrzéséről. A harmadik esetben teljes verziókezelés történik, egy elem lecserélésekor mindig új verzió jön létre.
- `Deleted Items` / *Törölt elemek* – A törölt elemek kezelését írhatjuk elő. `Retain Deleted Items` / *Törölt elemek megőrzése* esetén az elemek törlése csak logikai törlést jelent. A törölt elemek mindaddig megmaradnak, amíg azokra fizikai eltávolítást nem alkalmazunk (`purge`). Ha a `Display Deleted Items` / *Törölt elemek megjelenítése* be van kapcsolva, akkor a logikailag törölt elemek is megjelennek az adatterületen, szerkesztési módban `Deleted` / *Törölt* jelölés szerepel mellettük.
- `New And Updated Icons` / *Új és módosított ikon* – Megadhatjuk, hogy az adatterületre felvitt új és módosított elemek mellett mennyi ideig jelenjen meg az ezt jelző ikon.
- `System Purge` / *Törlések véglegesítése* – A törlésre kijelölt elemeket (`Deleted Items`) és a lejárt elemeket (`Expired Items`) törölhetjük véglegesen, fizikailag is.

4. `Labels` / *Címkék*
Az adatterületen megjelenő speciális címkéket és ikonjaikat cserélhetjük le. Cseréljük le a következő címkéket a magyar megfelelőjükre:

`Log Off Link`	`Kijelentkezés`
`Content Area Map Link`	`Térkép`
`Search Label`	`Keresés`
`Administration Link`	`Adminisztráció`

Kattintsunk az Apply / *Alkalmaz* gombra.

5. `Translations` / *Fordítások*
Ha a Portalhoz telepítettünk nyelvi támogatásokat, akkor az adatterületekhez megadhatunk ezeken a nyelveken fordításokat.

6. `Page` / *Oldal*
Az adatterület a Portalban speciális portáloldalakon keresztül jelenik meg. Ezek az oldalak nem érhetők el a navigátorban a többi oldal közül. Minden adatterülethez automatikusan létrejön egy oldal, amelyet szerkeszthetünk, emellett új oldalt is létrehozhatunk. Az oldalt a `Select Page` / *Oldal választása* lenyíló listából választhatjuk ki, majd a mellette levő `Edit` / *Szerkesztés* gombra kattintva szerkeszthetjük. Az adatterület oldalak szerkesztésekor az oldalaknál már tárgyalt fülek közül a `Main` / *Fő jellemzők* és a `Portlets` / *Portletek* lapon szerkeszthetünk, nem adhatunk meg az oldalnak saját stílust, mert azt a mappák és a navigációs sávok stílusai adják meg. Hasonlóképpen hiányzik a hozzáférést szabályozó fül is. Az oldalak szerkesztésével, új portletek hozzáadásával az adatterületeken is meg tudunk jeleníteni tetszőleges forrásból származó információt.

7. `Access` / *Hozzáférés*
Az adatterület egészére vonatkozóan adhatunk meg jogosultságokat. Itt három jogosultsági szint közül választhatunk:

- `Administer` / *Adminisztrál* – A feljogosított tetszőleges feladatot végrehajthat az adatterületen, adatterületadminisztrátori jogkör.
- `Manage Style` / *Stílus kezelése* – A feljogosított tetszőleges stílushoz kapcsolódó feladatot végrehajthat az adatterületen, stílusadminisztrátori jogkör.
- `Make Public` / *Publikussá nyilvánítás* – A feljogosított az általa létrehozott objektumokat mások számára elérhetővé teheti.

4.3.6. Adatelemek

Adatterületeket elsősorban azért használunk, hogy adatelemeket tároljunk rajtuk, amiket így osztottan elérhetővé teszünk. Elemek felviteléhez váltsunk át szerkesztés módba.

Vigyünk fel egy hírt a `Tanulmányi Osztály` mappába:

- Lépjünk be a mappába, annak nevére kattintva.
- Kattintsunk az `Add Item` / *Elem hozzáadása* linkre. Az elemek hozzáadását egy varázsló segíti.

Az `Add An Item` / *Egy elem hozzáadása* varázsló első oldalán az új elem típusát választjuk ki:

- `File` / *Állomány* – Egy adott állományt tölthetünk fel az adatterületre.

- `Text` / *Szöveg* – Maximum 32 KB méretű szöveges információt adhatunk meg.
- `URL` – Egy tetszőleges URL linkjét adhatjuk meg.
- `Folder Link` / *Mappahivatkozás* – Az adatterület egy mappáját adatelemként is felvehetjük, így elkerüljük a mappa duplikálását (soft link).
- `PL/SQL` – A megadott PL/SQL-kód fut le és ennek eredménye jelenik meg.
- `Application Component` / *Alkalmazáskomponens* – Egy alkalmazáskomponenst elemként használhatunk az adatterületen.
- `Image` / *Kép* – Képet, illetve képtérkép linket vehetünk fel az adatterületre.
- `Java Application` / *Java-alkalmazás* – JSP-alkalmazást helyezhetünk el az adatterületen.
- `Zip File` / *Zip-állomány* – ZIP tömörített állományokat vehetünk fel az adatterületre. Ezek azért különböznek más állományoktól, mert tartalmukat az adatterületen is kitömöríthetjük, így egy műveletként az állományrendszer egy könyvtárrendszerét is feltölthetjük.

Ha vannak saját elemtípusaink, akkor azokat is választhatjuk. Saját elemtípusokkal valamilyen meglévő elemtípushoz adhatunk meg további tulajdonságokat, vagy egy új típust hozhatunk létre egyedi tulajdonságokkal.

Válasszuk a `Text` típust, majd kattintsunk a `Next` / *Következő* gombra.

A varázsló második oldalán kell megadnunk:

- `Folder Region` / *Mappa régió* – A mappa több régióból áll, ezek elrendezését a mappa stílusa tartalmazza. Válasszuk ki az `Announcements` / *Hirdetmények* mezőt.
- `Primary Item Attributes` / *Adatelem elsődleges tulajdonságai* – A tulajdonságok egy része az elem típusától függ. A `Text` / *Szöveg* mezőbe írjuk be hirdetményünk szövegét: `"A téli vizsgaidőszak 2002. december 23-án kezdődik!"`. A `Display Name` / *Megjelentett név* mezőbe írjuk be: `"<b>Téli vizsgaidőszak</b>"`. Az elem kategóriája `Hírek` legyen.
- A `Publish Date` / *Publikálás dátuma* – Ennek a mezőnek a tartalma határozza meg, hogy az elem mikor jelenik meg a mappában.
- Az `Expiration Date` / *Lejárat dátuma* – Ebben a szekcióban adhatjuk meg az adatelem érvényességének végét. Ezután az elemet az adatterületről a többi lejárt elemmel együtt is törölhetjük (`purge`). Az `Expires On` / *Lejár ekkor* mezőbe írjuk be a megadott formátumban: `24-DEC-2002`.

Kattintsunk a `Next` / *Következő* gombra.

A varázsló utolsó oldalán adhatjuk meg a következőket:

- `Secondary Item Attributes` / *Adatelem másodlagos attribútumai* – Az elemhez rendelt szempontokat adhatjuk meg. Válasszuk ki az `Oktatóknak` és a `Hallgatóknak` szempontokat és aktiváljuk őket. A további tulajdonságokról a mező szintű súgóból tájékozódhatunk.

- `Display options` / *Megjelenítési opciók* – Az adatelemeket négyféleképpen jeleníthetjük meg a mappában:
  1. `Link That Displays Item In Folder Area` – Az elem neve egy linkként jelenik meg, amire kattintva az elem tartalma a mappaterületen jelenik meg.
  2. `Item Displayed Directly In Folder Area` – Az elem tartalma közvetlenül a mappaterületen jelenik meg.
  3. `Link That Displays Item In Full Browser Window` – Az elem neve linkként jelenik meg, amire kattintva az elem az egész böngészőablakban jelenik meg (lecserélve az adatterület oldalát).
  4. `Link That Displays Item In New Browser Window` – Az elem neve linkként jelenik meg, amire kattintva az elem az egy új böngészőablakban jelenik meg (az adatterület oldala az eredeti ablakban nyitva marad).

Válasszuk ki a 2. opciót.

Az előzőhöz hasonlóan vegyük fel az `Oktatók/dr. Kiss István` mappába a következő URL-t adatelemként:

Régió	`Regular Items` / *Szokásos elemek*
URL	`http://`*<szervernév>:<port>*`/`
Megjelenített név	`Oracle Webserver`
Lejárat dátuma	`Never Expires`
Megjelenítési opciók	4. opció, új böngészőablakban történő megjelenítés

A többi tulajdonság legyen az alapértelmezett.

4.3.7. Saját típusok

A mappák és az adatelemek létrehozásakor ki kell választanunk ezek típusát. Itt a beépített típusokon túl saját típusokat is használhatunk. Ilyen típusokat a navigátorban az adatterületen belül a `Custom Types` / *Saját típusok* link segítségével hozhatunk létre (ide is eljuthatunk az adminisztrációs eszközök táblázatából).

Saját típusként mappatípust és elemtípust tudunk létrehozni:

- Valamilyen beépített típust választunk ki alaptípusként és azt egészíthetjük ki saját attribútumokkal, melyeket korábban az `Attributes` / *Attribútumok* közt hoztunk létre. Elemtípusoknál nem szükséges alaptípust megadni, választható a speciális `<None>` / *Nincs* érték.
- A saját típusokhoz eljáráshívásokat definiálhatunk. A hívások megjelenhetnek linkként az ilyen saját típusú mappák, illetve elemek mellett. A linkre kattintva megtörténik a hívás és annak eredménye jelenik meg. Másik lehetőség, hogy a hívás már a mappa, illetve elem megjelenítésével egy időben törté-

nik meg és az eredményt közvetlen láthatjuk. Kétféle eljáráshívás közül választhatunk:

a) `HTTP` – egy URL-t adhatunk meg, ami után a kiválasztott paraméterek kerülnek felsorolásra a HTTP paraméterátadásának megfelelően.

b) `PL/SQL` – az adatbázisban tárolt PL/SQL-eljárást hívhatunk meg. Az eljárást a séma nevével minősítjük.

A létrehozott saját típusokat ezután a beépített típusokkal egyező módon használjuk mappák, illetve elemek típusaként, de nem adhatók meg további saját típusok alaptípusaként.

4.3.8. Megosztott objektumok

Az adatterület szerkezeti objektumait eddig a saját adatterületünkön belül hoztuk létre, más adatterületeken ezek nem használhatók. A mappákon kívül minden szerkezeti objektumot megosztottan is létrehozhatunk a navigátor adatterületek lapján a `Shared Objects` / *Megosztott objektumok* között.

Csak megosztottan érhetők el a `Personal Folders` / *Személyes mappák,* amelyek az egyes felhasználók megosztott mappái. Ezek létrehozásáról a portáladminisztrátor dönthet a rendszer globális beállításainál, erről bővebb információt az on-line súgó tartalmaz.

Az adatterületen privát módon és megosztott objektumként is létrehozhatunk kategóriákat, navigációs sávokat, szempontokat, stílusokat és saját típusokat.

Mappák nem hozhatók létre megosztottan.

4.4. Alkalmazások

Az alkalmazásokat adatbázisban tárolt adatok dinamikus megjelenítésére használjuk. Egy alkalmazás több komponenst tartalmazhat, melyek különböző dinamikus funkcionalitást valósítanak meg. Egy alkalmazás mindig egy adatbázissémához kötődik, az ebben tárolt adatokat kezeli. Az alkalmazások használata minden felhasználó számára lehetséges, létrehozásuk a portálfejlesztők (`PORTAL_DEVELOPERS` csoport) feladata.

Ebben az alfejezetben, a különböző típusú komponensek áttekintése után, létrehozunk egy adatterületet és azon belül néhány komponenst.

Alkalmazások létrehozásához a navigátort vagy az Oracle Portal Home Page adminisztrál lapján az `Applications` / *Alkalmazások* portletet használhatjuk.

4.4.1. Komponensek típusai

A komponensek speciális weboldalak, amiket PL/SQL tárolt eljárások valósítanak meg, létrehozásukhoz viszont nincs szükség a PL/SQL nyelv ismeretére, varázslókat használhatunk. A létrehozott komponensek az alkalmazás keretein belül önállóan, vagy az alkalmazást portletszolgáltatóként megjelölve portletekként portáloldalakon használhatók. Bizonyos komponensek, funkciójuknál fogva, sohasem használhatók portletként. A komponensek típusa a következők egyike lehet:

- `Forms` / *Űrlapok* – Felhasználói felületet nyújt egy vagy több adatbázisban tárolt tábla, nézet vagy eljárás eléréséhez.
- `Reports` / *Riportok* – Egy SQL-lekérdezés eredményét jelenítik meg táblázatos formában.
- `Charts` / *Diagramok* – Egy SQL-lekérdezés eredményét jeleníti meg oszlopdiagramként. A diagram alapjául szolgáló lekérdezésnek két oszlopot kell visszaadnia, az egyik az oszlopok neveit, a másik az oszlopok méretét adja meg.
- `Calendars` / *Naptárak* – Egy SQL-lekérdezés eredményét jeleníti meg naptár formájában. A lekérdezésnek vissza kell adnia egy időpontot, amely az adott táblasor naptárbeli helyét határozza meg.
- `Dynamic Pages` / *Dinamikus oldalak* – Dinamikus adatbázisbeli adatokat jelenít meg SQL- vagy PL/SQL-utasítás alapján. Az utasítás a komponens minden elérésekor végrehajtásra kerül.
- `XML Components` / *XML-komponensek* – XML-oldalakat jelenít meg.
- `Hierarchies` / *Hierarchiák* – Egy önhivatkozó tábla vagy nézet adatait jeleníti meg. (Az önhivatkozó táblában egy külső kulcs van definiálva, ami magát a táblát hivatkozza.) Egy hierarchia komponensben maximum három szint kerül megjelenítésre. Önhivatkozó táblákban tárolhatunk alá-fölérendeltségi viszonyokat, menükapcsolatokat stb.
- `Menus` / *Menük* – Olyan weboldal, amivel más komponensek közti navigációt adhatunk meg linkek segítségével.
- `URL Components` / *URL-komponensek* – Egy adott URL tartalmát jeleníti meg.
- `Frame Drivers` / *Keretvezérlők* – Két részre, két keretre osztott weboldal, ahol az egyik (a vezérlő) keret egy SQL-lekérdezést tartalmaz, ami meghatározza a másik (a vezérelt) keret tartalmát.
- `Links` / *Linkek* – Komponensekre vagy más HTML-oldalakra mutató linkeket hozhatunk létre. Ezek a linkek az URL-en kívül a komponensek lehetséges HTTP-paramétereiről is információt hordoznak.
- `Lists of Values (LOV)` / *Értéklisták* – A statikus vagy dinamikus értéklistákat más komponensek paraméterei értékének megadására használjuk. Az értéklisták többféleképpen jeleníthetők meg: lenyíló listákkal, rádiógombokkal stb.

A komponensek varázslói a komponens típusától függnek és általában jóval bonyolultabbak, mint a portáloldalak és adatterületek esetében voltak. Megismerésükhöz több időre és tapasztalatra van szükség, a mező szintű súgó nagy segítséget jelent az egyes varázslóoldalak kitöltésekor.

A komponensek egyenként vagy az alkalmazás egészében exportálható egy SQL-parancsállományként. Ezt az állományt végrehajtva az alkalmazás más Portal példányokban egyszerűen telepíthető.

4.4.2. A *Vizsga* példaalkalmazás

Az alkalmazás, amit létrehozunk, egy oktatási intézmény vizsgáinak nyilvántartására, megtekintésére szolgál. Mivel minden alkalmazás egy adatbázissémához kötődik, ezért szükség van egy adatbázissémára, amiben az adatokat tároljuk. A használt sémát az adatbázis-adminisztrátorral egyeztetve válasszuk ki. A továbbiakban erre a sémára `ORAWEB` néven hivatkozunk, ez nem keverendő össze a 4.2.1. alfejezetben létrehozott azonos nevű felhasználóval. Az Olvasó a példákban ezt a sémanevet mindig helyettesítse az általa használt sémanévvel.

A *Vizsga* alkalmazásban két táblát használunk. A következő SQL-kóddal hozzuk létre a táblákat és töltsük fel néhány adattal azokat. (Amennyiben a portálban használt felhasználónk rendelkezik adatbázis-adminisztrátori jogkörrel, használhatjuk a navigátort vagy az Oracle Portal Home Page `Administer Database` / *Adatbázis adminisztrálása* lapját a séma és a szükséges táblák létrehozásához.)

```
CREATE TABLE vizsgak (
  vizsga_id    NUMBER PRIMARY KEY,
  tantargy     VARCHAR2(100) NOT NULL,
  oktato       VARCHAR2(100) NOT NULL,
  idopont      DATE NOT NULL
)
/
CREATE TABLE vizsgajegyek (
  vjegy_id     NUMBER PRIMARY KEY,
  vizsga_id    NUMBER NOT NULL REFERENCES vizsgak,
  hallgato     VARCHAR2(100) NOT NULL,
  erdemjegy    NUMBER
)
/
INSERT INTO vizsgak VALUES (
  10, 'Programozás', 'dr. Kiss István', SYSDATE + 1
);
INSERT INTO vizsgak VALUES (
  20, 'Algorithms', 'dr. John Smith', SYSDATE + 1
);
INSERT INTO vizsgak VALUES (
  30, 'Adatbázisok', 'dr. Kiss István', SYSDATE + 3
);
INSERT INTO vizsgajegyek VALUES (
```

```
  11, 10, 'Szabó Péter', 4
);
INSERT INTO vizsgajegyek VALUES (
  12, 10, 'Fehér Imre', 4
);
INSERT INTO vizsgajegyek VALUES (
  13, 10, 'Kiss Sándor', 5
);
INSERT INTO vizsgajegyek VALUES (
  14, 20, 'Nagy Balázs', null
);
INSERT INTO vizsgajegyek VALUES (
  15, 20, 'Szabó Péter', 4
);
```

A `VIZSGAK` táblában egy tantárgy egy oktató által egy időpontban tartott vizsgájának adatai vannak. A `VIZSGAJEGYEK` tábla az ezeken a vizsgákon a hallgatók által elért jegyeket tartalmazza.

Ha az ékezetes karakterek tárolásával vagy megjelenítésével problémáink vannak, használhatjuk az ékezetes és más speciális karakterek helyett a megfelelő HTML-entitásokat (például `&#225;` = `&aacute;` = á).

Ezután hozzuk létre az alkalmazást (használjuk a navigátort vagy az Oracle Portal Home Page oldalt).

A varázsló egyetlen oldalán adjuk meg az alkalmazás nevét (`Name`): `VIZSGA_APP`, a megjelenített nevet (`Display Name`): `Vizsga alkalmazás`, és sémát (`Schema`): nálunk `ORAWEB`. Kattintsunk az `OK` gombra.

Az alkalmazás melletti `Manage` / *Kezel* linkre kattintva az alkalmazás kezelését segítő eszközöket megjelenítő oldalra kerülünk. Ezek az eszközök a navigátorból közvetlen linkeken keresztül is elérhetők. Az alkalmazást az `Edit` / *Szerkesztés* linkkel szerkeszthetjük. A `Delete` / *Törlés* linkkel törölhetjük, az `Export` linkkel exportálhatjuk. A `Grant Access` / *Hozzáférés engedélyezése* linkkel állíthatjuk be, hogy mely felhasználók milyen mértékben férnek hozzá az alkalmazáshoz és azt is, hogy az alkalmazás megjelenjen-e a portletszolgáltatók közt. A hozzáférést a komponensek szintjén felüldefiniálhatjuk, így egyes komponenseket elrejthetünk vagy elérhetővé tehetünk egyes felhasználók, csoportok számára.

A komponensek létrehozása során nem részletezzük minden oldal leírását, az egyes lépések végrehajtása után értelemszerűen használjuk a `Next` / *Következő* gombot.

A *Vizsgalap* űrlap

Létrehozunk egy űrlapot, amivel mindkét tábla tartalmát kezelhetjük:

- A navigátorban nyissuk meg a `VIZSGA_APP` alkalmazást.
- A `Create New...` / *Új... létrehozása* felirat után láthatjuk a létrehozható komponensek típusait. Válasszuk a `Form` / *Űrlap* linket.

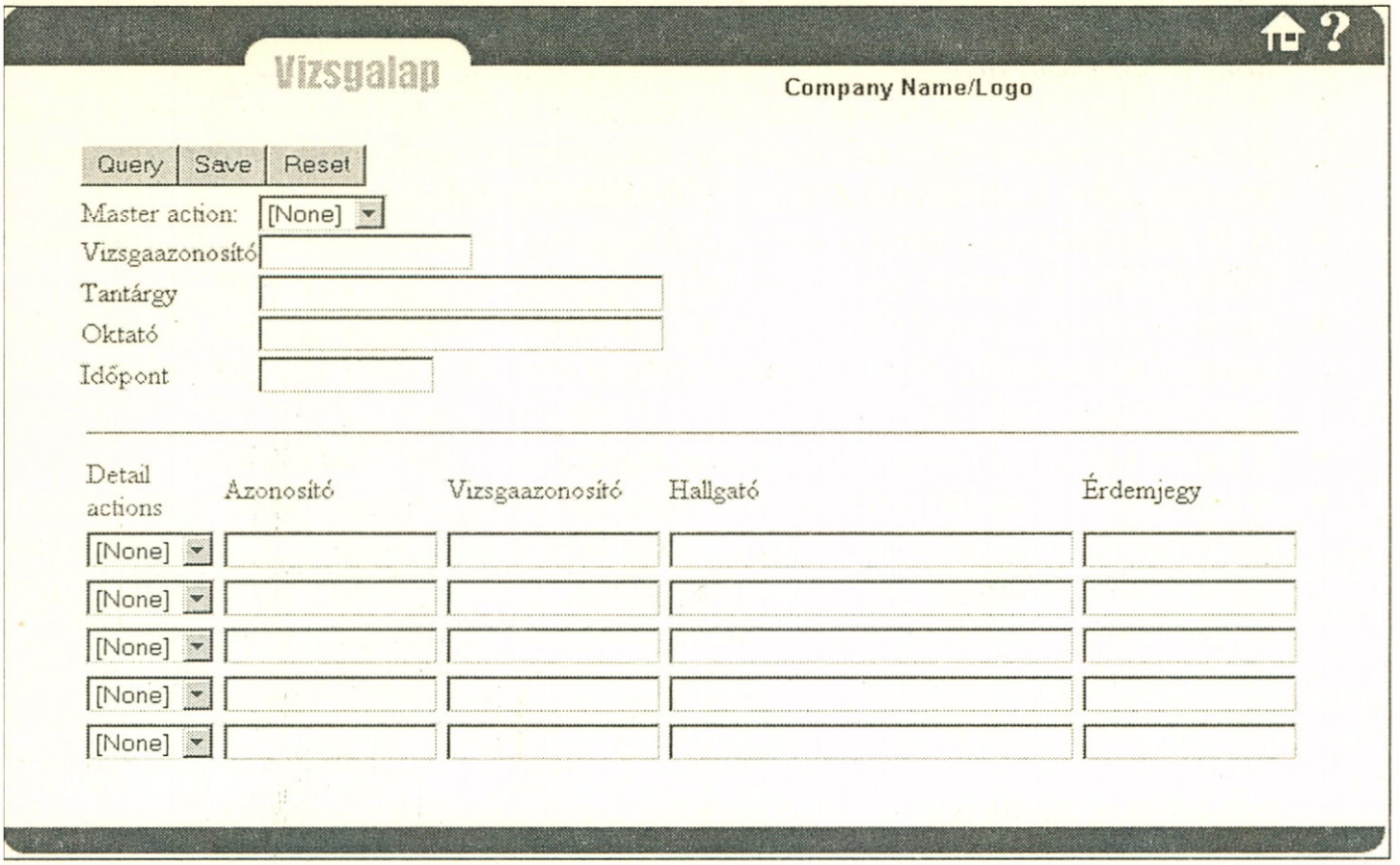
- Többféle űrlapot hozhatunk létre, válasszuk a `Master-detail form` / *Főtábla-alárendelt tábla űrlapot.*
- Adjuk meg az űrlap nevét (`Name`): `MDFORM_VIZSGALAP`, megjelenített nevét (`Display name`): `Vizsgalap`. Az alkalmazás (`Application`) maradjon a `VIZSGA_APP`. Kattintsunk a `Next` / *Következő* gombra.
- A fő tábla (`Master Table or View`) legyen: `ORAWEB.VIZSGAK`, az alárendelt tábla (`Detail Table or View`) legyen: `ORAWEB.VIZSGAJEGYEK`.
- A `Join Conditions` / *Összekapcsolás feltételei* oldalon fogadjuk el a felkínált kapcsolási feltételt, amely alapértelmezés szerint természetes összekapcsolás lesz (`VIZSGA_ID`).
- A `Form Layout` / *Űrlap elrendezése* legyen `Tabular` / *Táblázatos.*
- A varázsló 5. oldalán a főtábla kezelésekor megjelenő gombok, mezők konfigurálhatók. Megváltoztathatjuk a feliratok szövegét, stílusát, kezelőgombokat törölhetünk, felvehetünk. Tanulmányozzuk az oldalt, kattintsunk a bal oldalon az elemekre, hogy megismerjük azok tulajdonságait. Kattintsunk a `VIZSGA_ID` elemre és a `Label` / *Címke* mezőbe írjuk be: `Vizsgaazonosító`. Hasonlóképpen módosítsuk a többi oszlopnév címkéjét: `Tantárgy`, `Oktató`, `Időpont`. A `TANTARGY` és az `OKTATOK` mezőknél az `Input Width` / *Mező szélessége*-t állítsuk `30`-ra. Fogadjuk el a beállításokat.
- Az alárendelt tábla oszlopainak címkéjét is javítsuk: `Azonosító`, `Vizsgaazonosító`, `Hallgató`, `Érdemjegy`. A `HALLGATO` mező szélessége legyen `30`.
- A varázsló 9. oldalán kiválaszthatjuk a megjelenítés alapjául szolgáló sablont (`Template`) és megadhatunk HTML-kódrészleteket amik az oldal adott pontjain jelennek meg. A beállításokat változatlanul fogadjuk el.
- Az utolsó oldalon végrehajtandó PL/SQL-kódrészleteket adhatunk meg. Hagyjuk üresen a mezőket és kattintsunk a `Finish` / *Befejezés* gombra.

A komponens tulajdonságai oldalon a komponenst: szerkeszthetjük (`Edit`), futtathatjuk (`Run`, `Run as Portlet`), testre szabhatjuk (`Customize`), felvehetjük a kedvencek közé (`Add to Favorites`), információt kérhetünk róla (`About`), törölhetjük (`Delete`).

Az oldal fülein további funkciókat is kiválaszthatunk. Így például a komponens hozzáférését célszerű lenne úgy szabályozni, hogy azt csak az oktatók érhessék el, a hallgatók ne.

A komponensek szerkesztésekor mindig egy új verzió jön létre a komponensből, a korábbi verziókat explicit módon kell törölnünk. A továbbiakban fontos lesz még a `Run Link` / *Futtatás linkje* tulajdonság a komponensek közti kommunikáció megteremtéséhez.

Futtassuk a komponenst (4.18. ábra). Lekérdezéshez töltsünk ki néhány mezőt és kattintsunk a `Query` / *Lekérdez* gombra. Vegyünk fel néhány új vizsgaidőpontot, illetve a vizsgákhoz hallgatókat érdemjeggyel vagy anélkül és mentsük a módosításokat. Próbáljunk meg üres értéket megadni ott, ahol tudjuk, hogy nem lehet (piros mezők).



4.18. ábra. MDFORM_VIZSGALAP *komponens*

A további komponensek egymással kapcsolatban lesznek:

- CAL_VIZSGAK – Egy naptárat készítünk, amiben a vizsgákat tüntetjük fel. A vizsgák nevei a vizsgákon elért jegyek diagramjára linkelnek (CHART_VJEGYEK).
- CHART_VJEGYEK – Egy paraméterezett oszlopdiagramot hozunk létre, ahol a paraméter a vizsga azonosítója. A diagram azt mutatja, hogy hány hallgató érte el a különböző érdemjegyeket a vizsgán. Az érdemjegyek linkek, amikre kattintva az adott vizsgán az adott jegyet elért hallgatók listájához jutunk (RPT_VJEGYEK). Az oldal fejlécén link található a vizsgára jelentkezett összes vizsgázó listájára (RPT_VIZSGAZOK).
- RPT_VIZSGAZOK – Egy paraméterezett riportot készítünk, ahol a paraméter a vizsga azonosítója. A riport az adott vizsgára jelentkezett összes vizsgázót listázza érdemjegy szerint csökkenő sorrendben, név szerint ábécé sorrendben. Az oldal fejlécén link található a vizsgán elért jegyek diagramjára (CHART_VJEGYEK).
- RPT_VJEGYEK – Egy paraméterezett riportot készítünk, ahol a paraméter a vizsga azonosítója és a vizsgán elért eredmény. A riporton a hallgatók neve szerinti ábécé sorrendben rendezett eredmények látszanak. Az oldal fejlécén link található a vizsgára jelentkezett összes vizsgázó listájára (RPT_VIZSGAZOK).

Minden komponenshez egy link típusú komponenst is létrehozunk LINK_<*komponens név*> néven. Néha azonban a komponens tulajdonságainál le-

olvasható `Run Link` / *Futtatás linkje* címet közvetlenül is felhasználjuk link megadásához.

A hivatkozások miatt a komponensek létrehozásának sorrendje rögzített, tehát az előző listán alulról fölfelé haladva kell létrehoznunk a komponenseket. A kölcsönös hivatkozások miatt még így is elkerülhetetlen lesz a komponensek utólagos szerkesztése.

Az RPT_VJEGYEK riport

- Kattintsunk a `Create New... Report` / *Új riport létrehozása* linkre a navigátorban.
- A riportot varázslóval hozzuk létre, válasszuk a `Reports From Query Wizard` linket.
- Nevezzük el a riportot: `RPT_VJEGYEK`; a megjelenített név legyen: `"Azonos eredményt elért vizsgázók"`.
- A riport alapját képező tábla nevéhez írjuk be: `ORAWEB.VIZSGAJEGYEK`, majd kattintsunk az `Add` / *Hozzáadás* gombra.
- A megjelenítendő oszlopok közül válasszuk a hallgató (`VIZSGAJEGYEK.HALLGATO`) és érdemjegy oszlopokat (`VIZSGAJEGYEK.ERDEMJEGY`) és vigyük át a > nyíllal a jobb oldalra.
- A `Column Conditions` / *Oszlopfeltételek* oldalt hagyjuk üresen.
- Az elrendezés maradjon `Tabular` / *Táblázatos*.
- A `Column Formatting` / *Oszlopok formázása* oldalon lássuk el ékezetekkel a fejléc neveit (`Column heading`).
- A `Formatting Conditions` / *Formázási feltételek* oldalt hagyjuk üresen.
- A `Display Options` / *Megjelenítési opciók* oldalon a `Show NULL Values as` / *NULL értékek megjelenítése* mezőbe írjuk be: `Nincs`. Az oldal alján találjuk a rendezést előíró mezőket (`Row Order Options`), itt az első helyen (`Order by`) válasszuk ki a `VIZSGAJEGYEK.HALLGATOK` oszlopot és az `Ascending` / *Növekvő* rendezést, a név szerinti rendezéshez.
- A `Customization Form Display Options` / *Testre szabás űrlap megjelenítési opciói* oldalon a `Column Name` / *Oszlop neve* mezőkbe a lenyíló listákból válasszuk ki az 1. sorban a `VIZSGAJEGYEK.VIZSGA_ID`, a 2. sorban a `VIZSGAJEGYEK.ERDEMJEGY` oszlopot.
- Kattintsunk a `Finish` / *Befejezés* gombra (átugorjuk a hátralevő oldalakat).

Az elkészített riportot futtathatjuk, mert a paraméterek opcionálisként lettek megadva. Ekkor minden vizsgázást láthatunk, nem csak az egy adott vizsgához és jegyhez tartozókat.

Az oldalról hiányzik a link az összes vizsgázó listájára, mivel az még nem ismert. Ezt az `RPT_VIZSGAZOK` riport létrehozása után tesszük meg. A link csatolásához PL/SQL-t fogunk használni. Mivel ez a link nem lényeges eleme az alkalmazásnak, a kevésbé érdeklődő Olvasók átugorhatják azt.

Hozzuk még létre a komponenshez a `LINK_RPT_VJEGYEK` linket:

- Kattintsunk a `Create New... Link` / *Új link létrehozása* linkre a navigátorban.
- Adjuk meg a link nevét: `LINK_RPT_VJEGYEK`.
- Az `Oracle Portal Component` / *Oracle Portal komponens* rádiógomb legyen megjelölve, a `Target Component or URL` / *Célkomponens vagy URL* mező mellett kattintsunk az ikonra és a megjelenő ablakban válasszuk ki a `VIZSGA_APP.RPT_VJEGYEK` linket.
- Kattintsunk a `Finish` / *Befejezés* gombra (átugorjuk a hátralevő oldalt).

A linket futtatva láthatjuk, hogy a riport megjelenik.

Az `RPT_VIZSGAZOK` riport

- Kattintsunk a `Create New... Report` / *Új riport létrehozása* linkre a navigátorban.
- A riportot varázslóval hozzuk létre, válasszuk a `Reports From Query Wizard` linket.
- Nevezzük el a riportot: `RPT_VIZSGAZOK`; a megjelenített név legyen: `Vizsgaeredmények`.
- A riport alapját képező tábla nevéhez írjuk be: `ORAWEB.VIZSGAJEGYEK`, majd kattintsunk az `Add` / *Hozzáadás* gombra.
- A megjelenítendő oszlopok közül válasszuk a hallgató (`VIZSGAJEGYEK.HALLGATO`) és érdemjegy oszlopokat (`VIZSGAJEGYEK.ERDEMJEGY`) és vigyük át a > nyíllal a jobb oldalra.
- A `Column Conditions` / *Oszlopfeltételek* oldalt hagyjuk üresen.
- Az elrendezés maradjon `Tabular` / *Táblázatos*.
- A `Column Formatting` / *Oszlopok formázása* oldalon lássuk el ékezetekkel a fejléc neveit (`Column heading`).
- A `Formatting Conditions` / *Formázási feltételek* oldalt hagyjuk üresen.
- A `Display Options` / *Megjelenítési opciók* oldalon a `Show NULL Values as` / *NULL értékek megjelenítése* mezőbe írjuk be: `Nincs`. Az oldal alján találjuk a rendezést előíró mezőket (`Row Order Options`) itt az első helyen (`Order by`) válasszuk ki a `VIZSGAJEGYEK.HALLGATOK` oszlopot és az `Ascending` / *Növekvő* rendezést, a név szerinti rendezéshez.
- A `Customization Form Display Options` / *Testre szabás űrlap megjelenítési opciói* oldalon a `Column Name` / *Oszlop neve* mezőbe az 1. sorban a lenyíló listából válasszuk: `VIZSGAJEGYEK.VIZSGA_ID`.
- Kattintsunk a `Finish` / *Befejezés* gombra (átugorjuk a hátralevő oldalakat).

Az elkészített riportot futtathatjuk, mert a `VIZSGA_ID` paraméter itt is opcionálisként lett megadva. Itt is minden vizsgázást láthatunk, nem csak egy adott vizsgához tartozókat.

Az oldalról hiányzik a link a vizsgázók által elért eredmények diagramjára, mivel az még nem ismert. Ezt a `CHART_VJEGYEK` diagram létrehozása után tesszük meg. A link csatolásához itt is PL/SQL-t fogunk használni.

Hozzuk még létre a komponenshez a LINK_RPT_VIZSGAZOK linket. A létrehozás lépései megegyeznek a LINK_RPT_VJEGYEK linknél leírtakkal, azzal az eltéréssel, hogy itt a célkomponens természetesen a VIZSGA_APP.RPT_VIZSGAZOK lesz.

A CHART_VJEGYEK diagram

- Kattintsunk a Create New... Chart / *Új diagram létrehozása* linkre a navigátorban.
- A diagramot varázslóval hozzuk létre, válasszuk a Charts From Query Wizard linket.
- Nevezzük el a riportot: CHART_VJEGYEK; a megjelenített név legyen: Vizsgajegyek.
- A diagram alapját képező tábla nevéhez írjuk be: ORAWEB.VIZSGAJEGYEK.
- A diagram oszlopainak címkéi a jegyek lesznek, ezért a Label / *Címke* mezőben válasszuk ki az ERDEMJEGY oszlopot. (Ha ennél bonyolultabb feliratot szeretnénk használni, például: Jegy: 5, akkor ne varázslóval, hanem saját SQL-utasítással adjuk meg a diagram adatainak forrását, vagy használjunk olyan nézetet, ami ezt a transzformációt megvalósítja.) A Link mezőben válasszuk ki a LINK_RPT_VJEGYEK linket, majd kattintsunk a  link tulajdonságai ikonra. A megjelenő ablakban konfigurálhatjuk a link paramétereit. Az első két paramétert rögzítjük:
- A vizsgajegyek.vizsga_id paraméter esetén a Condition / *Feltétel* egyenlőség (=) legyen, a Static value / *Statikus érték* mező maradjon üres, a Column Name / *Oszlopnév* legyen VIZSGA_ID.
- A vizsgajegyek.erdemjegy paraméter esetén is egyenlőség legyen a feltétel, az oszlop pedig az EREDMENYEK oszlop.
- Az OK gombbal zárjuk be ezt az ablakot.
- A Value / *Érték* mezőben válasszuk ki a konstans 1 értéket, a Group Function / *Csoportfüggvény* mezőben a SUM függvényt.
- A Column Conditions / *Oszlopfeltételek* oldalon a Column Name / *Oszlopnév* mezőben válasszuk ki az EREDMENY oszlopot, a Condition / *Feltétel* mezőben a not null feltételt.
- A Display Options / *Megjelenítési opciók* oldalon az Order By / *Rendezés... szerint* mezőben válasszuk az ORDER BY LABEL DESC / *Címke szerint csökkenően* értéket.
- A Customization Form Display Options / *Testre szabás űrlap megjelenítési opciói* oldalon a Column Name / *Oszlop neve* mezőbe az 1. sorban a lenyíló listából válasszuk: VIZSGAJEGYEK.VIZSGA_ID.
- A Chart and Customization Form Text / *A diagramon és a testre szabása űrlapon megjelenő szövegek* oldalt ugorjuk át. Kattintsunk a Finish / *Befejezés* gombra (átugorjuk a hátralevő oldalakat).
- Az Additional PL/SQL Code / *További PL/SQL-kódok* oldalon lekérdezzük a vizsga nevét az adatbázisból és megjelenítjük. Ugyanitt létrehozzuk a linket, amivel a diagram komponensről az RPT_VIZSGAZOK komponenst érjük el. Eh-

hez írjuk be a következő PL/SQL-kódot a `Chart` oszlopban az `... after displaying the header` / *fejléc megjelenítése után* mezőbe:

```
DECLARE
  v_Vizsga_id  NUMBER;
  v_Vizsga     oraweb.vizsgak%ROWTYPE;
BEGIN
  v_Vizsga_id := get_value('vizsga_id');
  SELECT *
    INTO v_Vizsga
    FROM oraweb.vizsgak
    WHERE vizsga_id = v_Vizsga_id;
  htp.p('<h3>Tantárgy: ' || v_Vizsga.Tantargy
    || ' (' || v_Vizsga.Oktato
    || ') ' || TO_CHAR(v_Vizsga.Idopont, 'YYYY-MON-DD')
    || '</h3>');
  htp.br;
  htp.p('<a href="ORAWEB.RPT_VIZSGAZOK.show?'
    || 'p_arg_names=vizsgajegyek.vizsga_id&p_arg_values='
    || v_Vizsga_id
    || '&p_arg_names=_vizsgajegyek_vizsga_id_cond&p_arg_values=%3D">'
    || 'Vizsgaeredmények</a>');
  htp.br;
  htp.br;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
```

A link megvalósításakor felhasználtuk az `RPT_VIZSGAZOK` riport futtatási linkjét (`Run link`) és az Oracle Portalban a komponensek közti kommunikációra használt HTTP URL-paramétereket. A komponensek közti paraméterátadásról az on-line súgó nyújt információt.

Ha a PL/SQL-kód nem fordult le, a komponenst szerkesztve (Edit) javítsuk ki.

- Kattintsunk a `Finish` / *Befejezés* gombra (átugorjuk a hátralevő oldalakat).

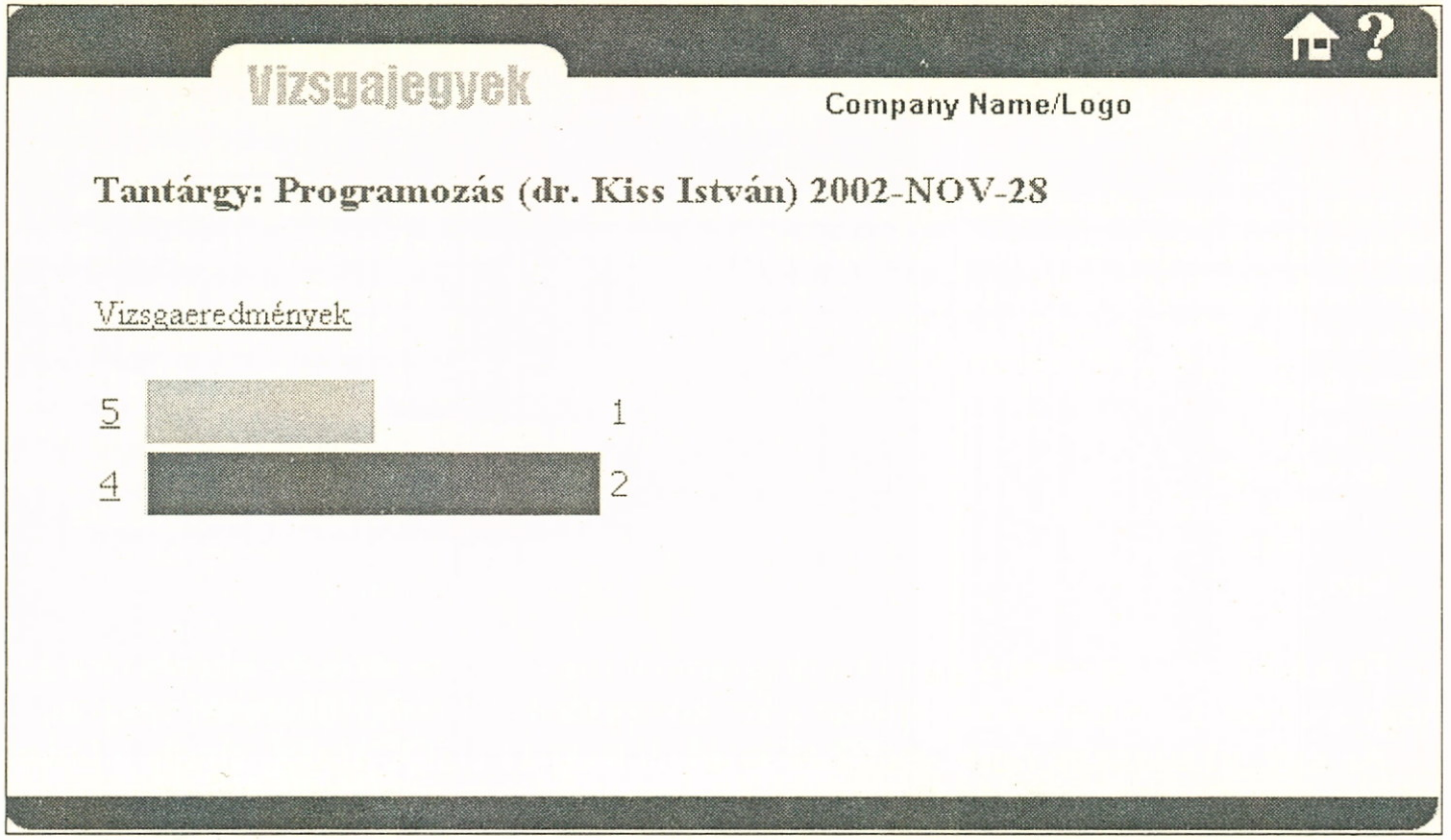
A diagramot futtathatjuk, ekkor azonban nem jelenik meg fejléc, mert hiányzik az opcionális `vizsga_id` paraméter. A következő URL-t megnyitva egy adott vizsga azonosítóhoz tartozó eredmények diagramját tekinthetjük meg:

```
http://<szervernév>:<port> <Portal_út>ORAWEB.CHART_VJEGYEK.show?p_arg_names=
vizsga_id&p_arg_values=<vizsga_id>
```

A mi esetünkben 10-es azonosítójú vizsga diagramját a következő URL adja meg:

```
http://oracleias.unideb.hu:7777/pls/portal30/ORAWEB.CHART_VJEGYEK.show?p_
arg_names=vizsga_id&p_arg_values=10
```

Itt már megjelenik a fejléc a tantárgy nevével és az összes vizsgázó eredményeit mutató linkkel (4.19. ábra).



4.19. ábra. CHART_VJEGYEK *komponens*

Hozzuk még létre a komponenshez a LINK_CHART_VJEGYEK linket. A létrehozás lépései megegyeznek a korábbi linknél leírtakkal, azzal az eltéréssel, hogy itt a célkomponens természetesen a VIZSGA_APP.CHART_VJEGYEK lesz.

Az RPT_VIZSGAZOK és az RPT_VJEGYEK riport szerkesztése

Az RPT_VIZSGAZOK és az RPT_VJEGYEK komponenseknél kihagytuk a linkek megadását, mivel a hivatkozott komponensek futtatási linkjei nem voltak ismertek. Szerkesszük most ezeket a komponenseket és egészítsük ki a megfelelő PL/SQL-kódokkal.

A komponensek szerkesztésekor a varázsló oldalain beállított tulajdonságokat változtathatjuk meg. Az oldal fülein most ikonokat találunk, amelyek a lapok tartalmára utalnak. A PL/SQL-kódok megadásához kattintsunk a jobb szélső fülre.

Az RPT_VJEGYEK riport esetében a Report oszlopban az ... after displaying the header / *fejléc megjelenítése után* mezőbe írjuk be:

```
DECLARE
  v_Vizsga_id  NUMBER;
  v_Vizsga     oraweb.vizsgak%ROWTYPE;
BEGIN
  v_Vizsga_id := get_value('vizsgajegyek.vizsga_id');
  SELECT *
    INTO v_Vizsga
    FROM oraweb.vizsgak
    WHERE vizsga_id = v_Vizsga_id;
```

```
  htp.p('<h3>Tantárgy: ' || v_Vizsga.Tantargy
    || ' (' || v_Vizsga.Oktato
    || ') ' || TO_CHAR(v_Vizsga.Idopont, 'YYYY-MON-DD')
    || '</h3>');
  htp.br;
  htp.p('<a href="ORAWEB.RPT_VIZSGAZOK.show?'
    || 'p_arg_names=vizsgajegyek.vizsga_id&p_arg_values='
    || v_Vizsga_id
    || '&p_arg_names=_vizsgajegyek_vizsga_id_cond&p_arg_values=%3D">'
    || 'Vizsgaeredmények</a>');
  htp.br;
  htp.br;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
```

Ez a kód csak egy sorban tér el a diagramnál megadott kódtól. Az eltérés oka az, hogy a `vizsga_id` paraméter értékét a két komponens különbözőképpen veszi át.

Az `RPT_VIZSGAZOK` riport esetében is adjuk meg a PL/SQL-kódot, amely a linket létrehozza. A szerkesztő oldal `Additional PL/SQL Code` / *További PL/SQL-kódok* lapján a `Report` oszlopban az `... after displaying the header` / *fejléc megjelenítése után* mezőbe írjuk be:

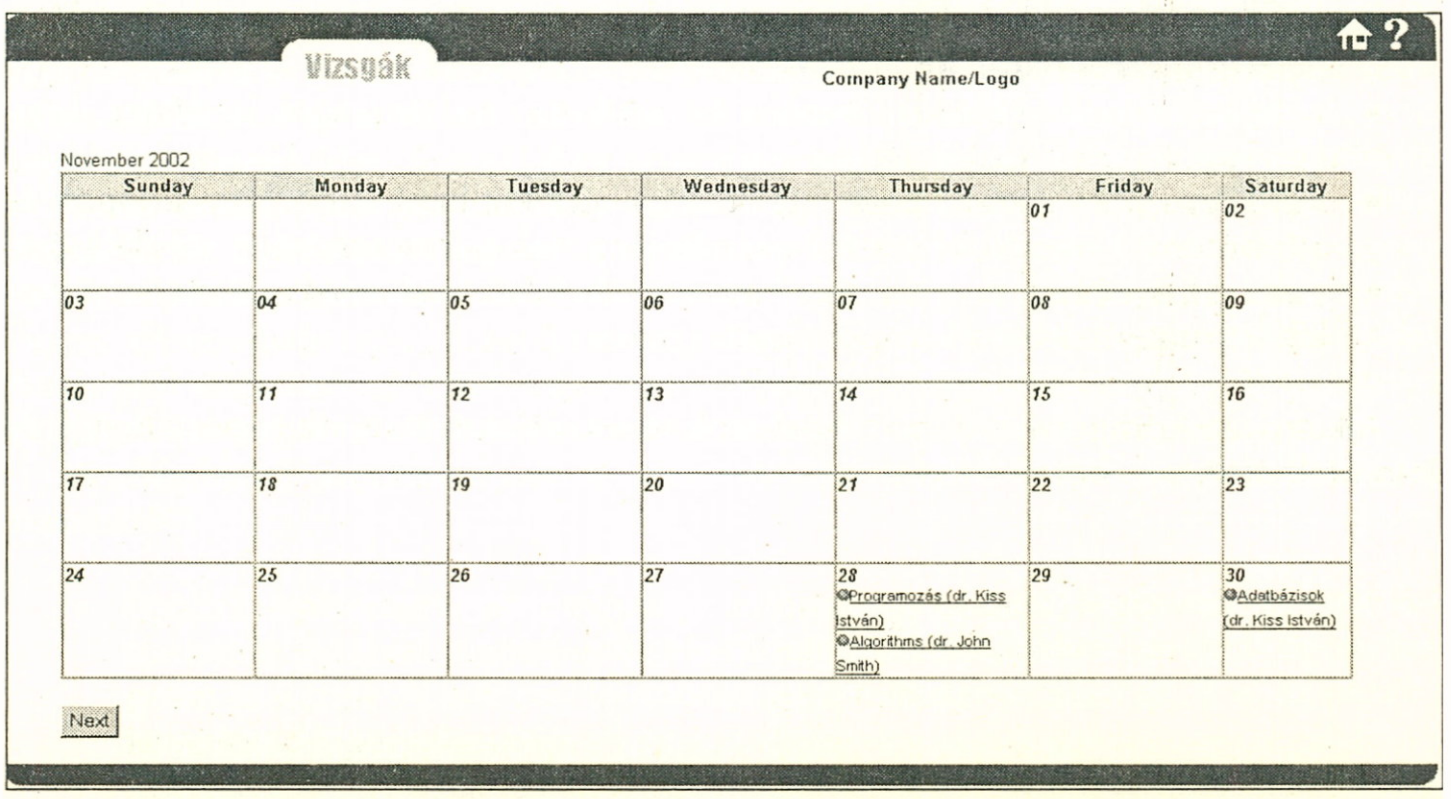
```
DECLARE
  v_Vizsga_id  NUMBER;
  v_Vizsga     oraweb.vizsgak%ROWTYPE;
BEGIN
  v_Vizsga_id := get_value('vizsgajegyek.vizsga_id');
  SELECT *
    INTO v_Vizsga
    FROM oraweb.vizsgak
    WHERE vizsga_id = v_Vizsga_id;
  htp.p('<h3>Tantárgy: ' || v_Vizsga.Tantargy
    || ' (' || v_Vizsga.Oktato
    || ') ' || TO_CHAR(v_Vizsga.Idopont, 'YYYY-MON-DD')
    || '</h3>');
  htp.br;
  htp.p('<a href="ORAWEB.CHART_VJEGYEK.show?'
    || 'p_arg_names=vizsga_id&p_arg_values='
    || v_Vizsga_id
    || '">Vizsgajegyek diagramja</a>');
  htp.br;
  htp.br;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
```

A kódokban fellelhető hasonlóságok kiemelése érdekében használhatunk PL/SQL tárolt eljárásokat az alkalmazásaink fejlesztése során.

A CAL_VIZSGAK naptár

- Kattintsunk a `Create New... Calendar` / *Új naptár létrehozása* linkre a navigátorban.
- Nevezzük el a naptárat: `CAL_VIZSGAK`, a megjelenített név legyen: `Vizsgák`.
- A naptár adatainak forrásaként egy SQL-utasítást használunk, aminek minden sora egy dátumot, a dátumhoz kapcsolódó eseményt és ezekhez adhat meg linkeket. Az általunk használt SQL minden sora egy vizsgát ad meg, és a vizsgák nevéhez a vizsgán elért jegyek diagramjának linkjét rendeli. Az `SQL Query` / *SQL-lekérdezés* mezőbe írjuk be:

```
SELECT
    idopont
        AS the_date,
    tantargy || ' (' || oktato || ')'
        AS the_name,
    'ORAWEB.CHART_VJEGYEK.show?'
    || 'p_arg_names=vizsga_id&p_arg_values='
    || vizsga_id
        AS the_name_link,
    NULL
        AS the_date_link,
    NULL
        AS the_target
FROM oraweb.vizsgak
ORDER BY idopont
```



4.20. ábra. `CAL_VIZSGAK` *komponens*

- A további oldalakon a megjelenítés tulajdonságait állíthatjuk be, most ezeket átugorjuk. Kattintsunk a `Finish` / *Befejezés* gombra.

Futtassuk a komponenst (4.20. ábra) és kattintsunk a vizsgák linkjeire, hogy a további komponenseket elérjük.

A példaalkalmazás ezzel elkészült. További komponensek létrehozását az Olvasóra bízzuk. Gyakorló feladatként hajtsuk végre a következőket:

- Hozzuk létre az `RPT_VIZSGAK` riportot, amin minden vizsga szerepel ábécé sorrendben. A vizsgák nevei a vizsgákon elért jegyek diagramjára (`CHART_VJEGYEK`) linkeljenek, mint a naptár esetén. Az oldal fejlécén legyen egy link a vizsgák naptárjára (`CAL_VIZSGAK`).
- A `CAL_VIZSGAK` naptár fejlécén legyen egy link az `RPT_VIZSGAK` riportra.

4.4.3. Megosztott komponensek

A navigátor alkalmazások lapján a konkrét alkalmazások mellett találjuk a `Shared Components` / *Megosztott komponensek* linkjét. Ezek nem olyan típusú komponensek, mint amiket mi is létrehozhatunk az alkalmazásainkban, hanem olyan elemek, amik bizonyos tulajdonságok megadásánál választhatók ki értékül.

Megosztott komponensként létrehozhatók:

- `Colors` / *Színek* – Színkonstansokat nevesíthetünk, amiket aztán táblázatok háttérszíneként, szövegek színeként stb. használhatunk fel.
- `Fonts` / *Betűtípusok* – Betűtípusokat nevesíthetünk, amiket szövegek, linkek betűtípusaként használhatunk fel.
- `Images` / *Képek* – Képeket vihetünk fel, amiket aztán háttérként, logóként, ikonként stb. használhatunk fel.
- `Javascripts` / *Javascriptek* – JavaScript-kódokat hozhatunk létre űrlapok és attribútumok mezőibe beírt felhasználói inputok érvényesítésére, validálására.
- `User Interface Templates` / *Felhasználó felület sablonok* – Háttérsablonokat hozhatunk létre a komponensek számára.

Alkalmazáskomponensek nem hozhatók létre megosztottan, de exportálhatók és importálhatók.

4.5. Felhasználók, rendszer szintű beállítások

Ebben az alfejezetben a Portal rendszer felhasználóiról, felhasználói csoportjairól, a Portal rendszer szintű beállításairól lesz szó és a speciális szerepet betöltő `PUBLIC` felhasználóval ismerkedünk meg. A portáladminisztrátor egyik fő feladata ezek konfigurálása, emellett sok más adminisztrátori teendője is van, amihez az Oracle Portal Home Page nyújt eszközöket. Erről bővebb információt az on-line súgó tartalmaz.

4.5.1. Felhasználók, felhasználói csoportok

Amikor a Portalról beszélünk, valójában három olyan rendszert is használunk, ahol a tevékenységek felhasználókhoz vannak kötve. Ez a három rendszer:

- az Oracle-adatbázis, ahová a rendszert telepítettük.
- a Single Sign-On / *Egyszeri belépés* rendszer, ami az adatbázisban tárolja a felhasználókat. Ezek a felhasználók nem csak a Portalban, hanem más Application Server-beli alkalmazásban is használhatók.
- az Oracle Portal alkalmazás, amit alapértelmezés szerint minden SSO-felhasználó használhat.

Az adatbázisban létrehozott felhasználók függetlenek a Portal-felhasználóktól. Az adatbázis sémái viszont használhatók a Portalon belül, például alkalmazások adatainak tárolására. A Portalt és az SSO-t megvalósító PL/SQL-eljárások, az ezekhez szükséges táblák és más adatbázis-objektumok szintén az adatbázisban tárolódnak. A telepítéskor jönnek létre a tároláshoz szükséges adatbázisbeli felhasználók és sémáik. A Portal rendszer az adatbázishoz ezekkel az adatbázisbeli felhasználói nevekkel kapcsolódik. Ezért bárki is legyen bejelentkezve a Portalba, az adatbázist mindig ugyanazokon a felhasználókon keresztül éri el. Ezért *nagyon fontos* a Portalon belüli *hozzáférésszabályozás*. Ez azért is hangsúlyos, mert a Portal rendszer adatbázis-felhasználói közt vannak DBA-jogkörrel rendelkezők, ugyanis csak így lehetséges a Portalon keresztül az adatbázis adminisztrálása is.

A telepítéskor automatikusan létrejönnek a következő Portal- (egyben SSO-) felhasználók:

- `PUBLIC` – Mindenki aki megnyitja a Portal egy oldalát, legkevesebb ezen felhasználó jogaival rendelkezik. Ha bejelentkezünk, akkor további jogosultságokat szerzünk és belépett (authenticated) felhasználóként érjük el az oldalt a következő kijelentkezésig.
- *<Sémanév>* – A telepítéskor megadott sémanévvel automatikusan létrejön egy felhasználó. Ez a Portal-felhasználó portáladminisztrátori és adatbázis-admi-

nisztrátori jogkörrel is rendelkezik. Alapértelmezés szerint a séma neve telepítéskor `PORTAL30`.

- *<Sémanév>*`_ADMIN` – Ez a felhasználó portáladminisztrátor lesz, nem rendelkezik adatbázis-adminisztrátori jogkörrel.

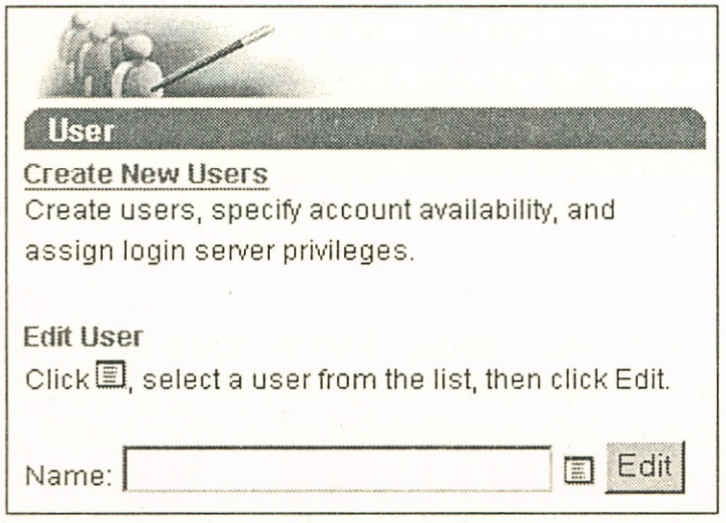
A továbbiakban a Portal felhasználóira koncentrálunk, nem foglalkozunk az adatbázis felhasználóival.

Portal-felhasználókat az Oracle Portal Home Page adminisztrál oldalán a `User` / *Felhasználók* portlet segítségével tudunk létrehozni, módosítani. Minden Portal-felhasználó létrehozásakor egy SSO-felhasználó is létrejön és viszont. A Portal-felhasználó rendelkezik az SSO-ban megadott tulajdonságokkal és csak a Portalra vonatkozó beállításokkal, profilokkal.

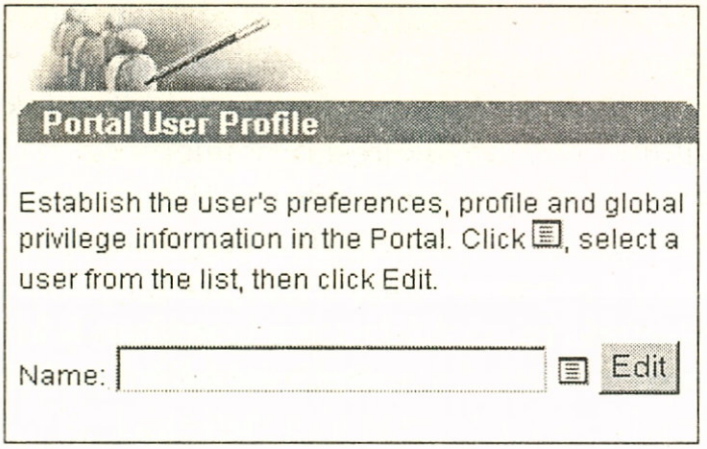
Az SSO-tulajdonságokat a `User` / *Felhasználók* portlettel adhatjuk meg (4.21. ábra). Szerkesszük az ORAWEB felhasználó beállításait:

- A portlet `Edit User` / *Felhasználó szerkesztése* szekciójában a `Name` / *Név* mezőbe írjuk be: `ORAWEB`; majd kattintsunk az `Edit` / *Szerkesztés* gombra.
- A `User Details` / *Felhasználó részletei* közt adhatjuk meg a felhasználói account jelszavát, e-mail címét.
- Az `Account Activation And Termination` / *Account aktiválása és megszüntetése* részben megadhatunk két dátumot, ami között a felhasználói account használható. A megszüntetés dátuma üresen hagyható.
- A `Login Server Privileges` / *Bejelentkezési szerver (SSO) jogosultsági szint* mezőben adhatjuk meg, hogy a felhasználói account rendelkezik-e SSO adminisztrátori jogkörrel. Ez a jogosultság a Portal-jogosultságoktól független, csak az SSO-ra vontakozik. (Viszont minden Portal-felhasználónak egyben SSO-felhasználónak is kell lennie.)

Az SSO-beállítások közül az átlagos felhasználók csak a jelszót változtathatják meg.



4.21. ábra. *User portlet*



4.22. ábra. *Portal User Profile portlet*

A Portalon belüli beállításokat a `Portal User Profiles` / *Portal-felhasználók profiljai* portletben adhatjuk meg (4.22. ábra). Ezen beállítások egy részét a felhasználók maguk is megváltoztathatják a portáloldalakon az `Account Info` / *Account információk* linkre kattintva. A felhasználói információk egy része a `Portal Directory` / *Portal névtár* alapján mások számára is elérhetővé tehető. A felhasználók maguk is dönthetnek arról, hogy a személyes információikat elérhetik-e mások. A névtárban a portáloldalakon elhelyezhető `People` / *Emberek* portlettel kereshetünk.

Szerkesszük az `ORAWEB` felhasználó profilját és ismerkedjünk meg a lehetséges beállításokkal:

- A portlet `Name` / *Név* mezőjébe írjuk be: `ORAWEB`, majd kattintsunk az `Edit` / *Szerkesztés* gombra.

Az oldalon több fül található:

- A `Main` / *Fő jellemzők* lapon is megadhatjuk a jelszót és az e-mail címet. Itt engedélyezhetjük, illetve tilthatjuk meg, hogy egy SSO-felhasználó a Portalba is bejelentkezzen (`Allow User To Log On`). Emellett kiválaszthatjuk, hogy a felhasználó bizonyos speciális csoportoknak tagja legyen-e, így például DBA, adatbázis-adminisztrátori jogkörrel ruházhatjuk fel stb. A további mezők jelentéséről lásd a mező szintű súgót.
- A `Preferences` / *Preferenciák* lapon további lehetőség van további személyes adatok megadására. Itt választhatjuk ki, hogy az információk a névtárban elérhetők legyenek-e. Engedélyezhetjük a személyes mappa (`Personal Folder`) használatát a felhasználó számára, amelyben adatelemeket és mappákat hozhat létre (az adatterületeknél megismert módon) és ezt mások számára elérhetővé teheti. A személyes mappák létrehozását nem elég itt engedélyezni, rendszerszinten is meg kell azt tenni (lásd a következő alfejezetet).
- A preferenciák lapon adhatjuk meg a felhasználó elsődleges csoportját (`Default Group`). Ennek akkor van jelentősége, ha a felhasználó több csoportba is tartozik és valamely csoport beállításait szeretné alapértelmezettként használni. A

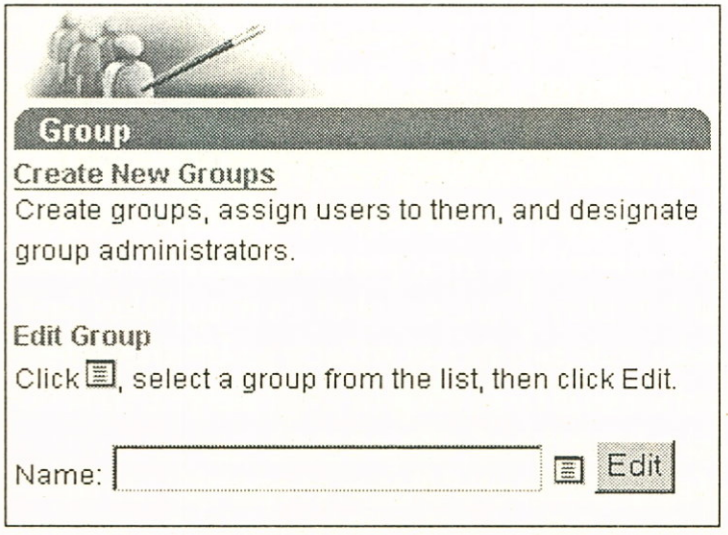
felhasználó effektív jogosultságait ez nem befolyásolja, mert az effektív jogosultság mindig az elérhető jogosultságok uniója lesz.

- A `Default Home Page` / *Alapértelmezett nyitóoldal* mezőben megadhatunk egy portáloldalt. Erre az oldalra kerülünk, miután sikeresen bejelentkezünk. Ha ezt a mezőt üresen hagyjuk, akkor az elsődleges csoportnál beállított nyitóoldal lesz az érvényes. Ha nincs elsődleges csoport megadva vagy a csoport számára nincs nyitóoldal beállítva, akkor a rendszer szinten megadott nyitóoldal lesz az érvényes. Az aktuális felhasználó nyitóoldala a navigátorban is kiválasztható az oldal neve melletti `Make Default` / *Alapértelmezésként beállít* linkre kattintva.
- A `Contact info` / *Elérhetőségek* lapon a névtárban megjelenő címeket, telefonszámokat adhatjuk meg. A felhasználó ezeket az adatokat maga is módosíthatja.
- A `Privileges` / *Jogosultságok* oldalon a felhasználó rendszer szintű jogosultságait adhatjuk meg. A jogosultságok az objektumok típusai szerint kategóriákba vannak sorolva: oldalakra, adatterületekre, alkalmazásokra, adminisztrátori tevékenységekre vonatkozó jogosultságok. Ezek részletes leírásáról lásd a mező szintű súgót.

A jogosultságok és beállítások adminisztrálásának megkönnyítésére, központosítására a legtöbb informatikai rendszerben alkalmaznak valamilyen csoportkezelést, adatbázisokban szerepköröket, operációs rendszereknél felhasználói csoportokat stb. A Portal is alkalmazza ezt a megoldást.

A felhasználói csoportokat a `Group` / *Csoportok* portlettel tudjuk adminisztrálni (4.23. ábra). Az összes csoport megtekintéséhez kattintsunk a listázás ikonra. A felhasználói csoportoknak felhasználók és más csoportok is tagjai lehetnek. A telepítéskor létrejönnek a következő csoportok:

- `AUTHENTICATED_USERS` / *Bejelentkezett felhasználók* – A csoportnak minden felhasználó tagja, a `PUBLIC` felhasználót kivéve. Itt adhatjuk meg a minden



4.23. ábra. *Group portlet*

belépett felhasználó által elérhető jogosultságokat. Így alapértelmezésként minden bejelentkezett felhasználó rendelkezik az `All Pages : Create` / *Minden oldal : Létrehozás* rendszerjogosultsággal, így minden új felhasználó létre tud hozni saját oldalakat.

- `DBA` / *Adatbázis-adminisztrátorok* – A csoport tagjai a Portalon belül minden feladatot végrehajthatnak. Csak az ebbe a csoportba tartozó felhasználók hajthatnak végre adatbázis-adminisztrációs feladatokat.
- `PORTAL_ADMINISTRATORS` / *Portáladminisztrátorok* – A csoport tagjai a Portalon belül minden feladatot végrehajthatnak, kivéve az adatbázis-objektumok kezelését és a nem tulajdonukban levő csoportok szerkesztését.
- `PORTAL_DEVELOPERS` / *Portál fejlesztői* – A csoport tagjai alkalmazásokat hozhatnak létre.
- `PORTLET_PUBLISHERS` / *Portlet publikálók* – A csoport tagjai az adatterületek és alkalmazások elemeit portletként elérhetővé tehetik, így a mappák, navigációs sávok, kategóriák, alkalmazások, kategóriák és szempontok felhasználhatók portáloldalakon.

Egy felhasználói csoport hatásköre vagy az egész portál, vagy egy kiválasztott adatterület. Utóbbi esetén a csoport csak a megadott adatterületen használható fel jogosultságok kezelésére.

Felhasználói csoport létrehozásakor és szerkesztésekor megadhatjuk:

- a csoport nevét, hatáskörét (`Applies To`), azt, hogy a csoport látható-e a nem tulajdonos felhasználók számára, a csoport nyitólapját és alapértelmezett stílusát;
- a csoport tagjait, akik már létező csoportok és felhasználók lehetnek;
- a csoport rendszer szintű jogosultságait.

A csoport hatáskörét csak létrehozáskor lehet megadni, az később nem változtatható.

Felhasználói csoportok törölhetők a szerkesztő ablakban a `Delete` / *Törlés* gombbal. Az SSO-felhasználók is törölhetők ugyanígy. SSO-felhasználók törlésekor a törölt accounttal a Portal sem érhető el. Ha viszont egy törölt SSO-felhasználó nevével ismét létrehozunk egy SSO-felhasználót, akkor a hozzá kapcsolódó Portal-felhasználó a törlés előtti beállításokkal fog rendelkezni. Ilyenkor szerkesszük a Portal-felhasználó profilját és kattintsunk a `Reset to Defaults` / *Alapértékek visszaállítása* gombra.

A `Self-registration` / *Automatikus regisztráció* modul lehetővé teszi új felhasználók automatikus létrehozását. A modul telepítése után a `Login` / *Bejelentkezés* portletben elérhető lesz az a link, amire kattintva az új felhasználó megadja az adatait, s ezután számára automatikusan létrejön egy Portal-felhasználó. A modul telepítéséről lásd [15].

4.5.2. Rendszer szintű beállítások

A Portal rendszer szintű beállításait az Oracle Portal Home Page oldal adminisztrál lapján a `Services` / *Szolgáltatások* portlet `Global Settings` / *Globális beállítások* linkjére kattintva érhetik el a portáladminisztrátorok.

Az oldalon a következő beállításokat végezhetjük el:

- A `Default Home Page` / *Alapértelmezett nyitóoldal* mezőben megadhatunk egy portáloldalt. Erre az oldalra kerülünk, miután sikeresen bejelentkezünk. Minden felhasználó és minden felhasználói csoport számára megadható egy nyitóoldal. Így ez a beállítás csak akkor lesz érvényes, ha sem felhasználói sem csoport szinten nem bíráljuk felül. Meg kell jegyeznünk még, hogy minden be nem jelentkezett felhasználóra a Portalban elsősorban a `PUBLIC` felhasználó beállításai érvényesek, a `PUBLIC` felhasználó is rendelkezik felhasználói profillal.
- A `Default Style` / *Alapértelmezett stílus* mezőben egy stílust választhatunk ki. Ez a beállítás csoport szinten felüldefiniálható.
- `Proxy Server` / *Proxyszerver* megadásával tűzfalon kívüli alkalmazásainkat érhetjük el (például egy URL tartalmát megjelenítő adatelem esetén).
- A `Login Server Settings` / *Bejelentkezési szerver beállításai* használatával megváltoztathatjuk a használatban levő SSO-szervert.
- Az `Activity Log Interval` / *Tevékenységek naplózási időköze* határozza meg, hogy a tevékenység naplókat milyen időközönként cserélje a rendszer. A portáladminisztrátorok állíthatják be, hogy milyen típusú elemek elérését, használatát naplózza a rendszer.
- A `Portlet Timing` / *Portletek időmérése* bekapcsolása esetén minden oldalon a megjelenő portletek hívási ideje is megjelenik. Ezt az információt használhatjuk a válaszidő hangolására portletek és bonyolultabb oldalak fejlesztése esetén.
- A `Temporary Directory` / *Temporális könyvtár*-at használja a rendszer az átmeneti jelleggel létrehozott állományok tárolására. Ilyenek az alkalmazások exportja és importja során használt SQL-parancsállományok.
- Ha a `Personal Folders` / *Személyes mappák* használatát engedélyezzük, akkor minden újonnan létrehozott felhasználó számára automatikusan létrejön egy személyes mappa, ahol a felhasználó adatelemeket és mappákat hozhat létre (ahogyan az adatterületeknél is), és azok tartalmát megoszthatja más felhasználókkal.
- A `Cookie Domain` / *Cookie-tartomány* a Portal elosztott telepítésekor állítható be, segítségével megadható a hálózati tartomány azon része, ami minden Portal szerver címében közös (így bármelyik Portal példány beállíthat olyan cookie-kat, amit mindegyik példánynak elküld a kliens böngésző). Amennyiben a Portalt nem elosztottan telepítettük, a cookie-tartomány a teljes gépnév lesz.

- A `Logout Behavior` / *Kijelentkezés viselkedése* beállításakor két opció közül választhatunk: a felhasználók kijelentkezéskor mind a Portal-rendszerből, mind az SSO-ból kijelentkeznek, vagy csak a Portal-rendszerből. Utóbbi esetben a Portalból kijelentkezett felhasználók továbbra is használhatják azokat a Portalon kívüli alkalmazásokat, amelyek az SSO-t használják a felhasználók be- és kiléptetésére.
- A `Beta Features` / *Béta lehetőségek*-nél dönthetünk a még teljesen nem tesztelt eszközök használatáról.
- A `Version Information` / *Verzióinformáció* a Portal verziószámát jeleníti meg.
- Amennyiben telepítettük a `Self-registration` / *Automatikus regisztráció* modult megadhatjuk, hogy a magukat regisztráló felhasználók a regisztráció után közvetlenül beléphetnek-e a Portalba (`Enable Users To Log On Immediately`), vagy ehhez a portáladminisztrátor engedélye szükséges.

4.5.3. A `PUBLIC` felhasználó

Az Oracle Portalban létezik egy speciális felhasználó, a `PUBLIC` felhasználó, aki a Portalba be nem jelentkezett felhasználókat jelenti. Minden felhasználó – függetlenül attól, hogy bejelentkezett vagy sem – rendelkezik a `PUBLIC` felhasználó jogosultságaival.

A `PUBLIC` felhasználó is rendelkezik profillal, amit a portáladminisztrátor szerkeszthet. Itt adhatjuk meg a publikus nyitóoldalt is.

A Portalba bejelentkezett felhasználók az oldalakat és a portleteket testre szabhatják (ha rendelkeznek megfelelő jogosultságokkal), a be nem jelentkezett felhasználók ezt sosem tehetik meg, mert a `PUBLIC` felhasználó nem jelent valódi személyiséget.

4.6. A Portal Development Kit – PDK

A Portal Development Kit (Portal fejlesztői csomag) egy Javában írt keretrendszer, amely lehetővé teszi saját web alapú alkalmazások fejlesztését, integrálását a Portal rendszerbe.

Már láttuk, hogy az Oracle Portal oldalai portletekből épülnek fel. Azt is láttuk, hogy a portleteket a portletszolgáltatókon keresztül érhetjük el. Portletszolgáltatóként egyelőre csak a kiemelt szolgáltatókat (`Seeded providers`) használtuk. A Portal elemei közül az egyéb (`Other`) portletszolgáltatók közt jelennek meg:

- azok az oldalak, amelyeket portletként elérhetővé teszünk;
- azok az adatterületek, ahol legalább egy elemet (kategória, szempont, navigációs sáv, mappa) portletként elérhetővé tettünk;

- azok az alkalmazások, ahol a hozzáférés beállításánál az alkalmazást portletszolgáltatóként jelöltük meg.

Előfordul azonban, hogy ezek a portletek az oldalainkon nehézkesen, vagy egyáltalán nem használhatók valamely kitűzött feladat megvalósítására. Ez természetes is, hiszen lehetetlen minden feladatra egy beépített portletet írni. Ilyenkor külső portletszolgáltatókat kell alkalmaznunk. Saját portleteket létrehozhatunk az adatbázisban PL/SQL nyelven, vagy a PDK keretrendszeren keresztül portletként használhatunk fel szinte tetszőleges web alapú alkalmazást (ezek HTTP-n keresztül elérhető állományok, weboldalak, JSP- és SQLJSP-oldalak, szervletek stb. lehetnek).

Ebben az alfejezetben egy portletszolgáltatót konfigurálunk két egyszerű web alapú portletettel. Ehhez portáladminisztártori jogosultsággal és az Oracle HTTP Server adminisztrálásához szükséges jogokkal kell rendelkeznünk.

A PDK egyes Portal-installációkban automatikusan rendelkezésre áll egy `jpdk.zip` nevű állományban. Ha az installáció nem tartalmazza az állományt, akkor az Oracle weboldaláról (otn.oracle.com) letölthető. A továbbiakban feltételezzük, hogy a PDK csomag telepítése megtörtént, és a csomag példaportletjei konfigurálva vannak.

A csomag használatáról és portletek készítéséről számos angol nyelvű cikket találhatunk az említett oldalon.

4.6.1. Példa portletszolgáltató

A PDK vagy JPDK (Java PDK) egy olyan Java nyelvű API, ami kapcsolódási felületet biztosít az Oracle Portalhoz. Az API kétféleképpen használható:

- a kommunikációt leíró interfészeket mi magunk implementálhatjuk Java nyelven;
- felhasználhatjuk a PDK-ban lévő, az interfészeket implementáló alaposztályokat más alkalmazások csatolására.

Általában az utóbbi módon használjuk fel a PDK-t, hogy a saját alkalmazásainkat a Portalba integráljuk, így lesz ez a most következő példában is. Ha a meglévő osztályok nem nyújtanak kellő rugalmasságot egy feladat megoldásában, akkor használjuk ki az interfész alapú tervezés rugalmasságát.

A portletszolgáltató a példánkban egy HTTP-szervlet lesz. A szervletet az API egyik osztálya (`oracle.portal.provider.v1.http.HttpProvider`) valósítja meg, nincs szükség saját kód írására. A szervlet induláskor egy `provider.xml` nevű XML konfigurációs állományt olvas be. Ez az állomány írja le a szolgáltató portletjeit.

Amikor egy portletet használunk, a portletet több üzemmódban láthatjuk: megjelenítés (`show`), testre szabás (`customize`), névjegy (`about`), súgó (`help`), szerkesztés (`edit`), bemutató (`preview`) stb. Éppen ezért egy portlet a programozó szempontjából az ezen üzemmódokban megjelenő HTML-forrásrészleteket előállító eszközökből (Java-kód, JSP-oldal, szervlet stb.) és azok konfigurációjából áll. Egy portlet HTML-

forrását egy üzemmódban előállító egységet a továbbiakban *forráskészítőnek* nevezzük. Ha a portleteket mobileszközökről elérhető oldalakon használjuk fel, akkor nem HTML-kódot, hanem az eszköznek megfelelő forrást kell szolgáltatniuk.

A példánk első portletje egy egyszerű portlet lesz, ami kiírja az aktuális dátumot és az aktuális Portal-felhasználó nevét. Egyetlen üzemmóddal rendelkezik, ez a megjelenítés (`show`) mód. Ezzel a móddal minden portletnek rendelkeznie kell.

Az üzemmódot egy Java nyelvű forráskészítő osztály a `DatumRenderer` valósítja meg. Ennek az osztálynak implementálnia kell az `oracle.portal.provider.v1.ManagedRenderer` interfészt. Ezt általában az `oracle.portal.provider.v1.http.BaseManagedRenderer` osztály származtatásával tesszük meg, ahol a `renderBody` módszert implementáljuk. Hozzuk létre a `DatumRenderer.java` állományt:

```
import java.io.*;
import java.util.Date;
import oracle.portal.provider.v1.*;
import oracle.portal.provider.v1.http.BaseManagedRenderer;

public class DatumRenderer extends BaseManagedRenderer {
    public void renderBody(PortletRenderRequest pr)
        throws PortletException {

        try {
            // A kérésből lekérdezzük a Writer objektumot,
            // ide írjuk a HTML-kódot.
            PrintWriter out = pr.getWriter();

            // Kiírjuk az aktuális dátumot.
            out.println("Aktu&#225;lis d&#225;tum: "
                        + new Date() + "<br>");
            // A kérésből lekérdezzük a felhasználó nevét és kiírjuk
            out.println("<b>Felhaszn&#225;l&#243;: "
                        + pr.getUser().getName() + "</b><br>");
        } catch (IOException ioe) {
            throw new PortletException(ioe);
        }
    }
}
```

A forrást fordítsuk le, közben figyeljünk oda, hogy a PDK `lib` könyvtárában található `provider.jar` benne legyen a `CLASSPATH`-ban.

Hozzuk létre a `provider.xml` konfigurációs állományt:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?providerDefinition version="2.0"?>
<provider class="oracle.portal.provider.v1.http.DefaultProvider">
   <session>true</session>

   <portlet class="oracle.portal.provider.v1.http.DefaultPortlet">
      <id>1</id>
```

```
      <name>DatumPortlet</name>
      <title>Datum Portlet</title>
      <description>
         Kiirja az aktualis datumot es az aktualis felhasznalo nevet
      </description>
      <timeout>10</timeout>
      <timeoutMessage>A dátum portlet lassú volt</timeoutMessage>
      <acceptContentType>text/html</acceptContentType>
      <renderer class="oracle.portal.provider.v1.RenderManager">
         <showPage class="DatumRenderer"/>
      </renderer>
   </portlet>

</provider>
```

Az XML-tagek jelentésének specifikációját az Oracle weboldaláról (otn.oracle.com) tölthetjük le.

A konfiguráció szempontjából lényeges, hogy a konfigurációs állomány és a lefordított Java-állomány (`.class`) hol található. Ez a könyvtár a mi esetünkben az `/oracle/oraweb/portlets` lesz, az Olvasó ezt mindig cserélje le a tényleges könyvtár nevére.

Most konfiguráljuk az Oracle HTTP Serverben a portletszolgáltató szervletet:

- A `zone.properties` JServ konfigurációs állományban a következő sorok hozzáadásával konfiguráljuk a szervletet:

  ```
  servlet.OrawebProvider.code=oracle.portal.provider.v1.http.HttpProvider
  servlet.OrawebProvider.initArgs=provider_root=/oracle/oraweb/portlets,
  sessiontimeout=1800000
  ```

  Ne felejtsük el lecserélni a `provider_root`-nál megadott könyvtárnevet.
- A `jserv.properties` JServ konfigurációs állományban adjuk meg, hogy hol található a lefordított Java-állomány (a PDK telepítésekor is itt kell megadni azt, hogy hol található a `providers.jar` állomány):

  ```
  wrapper.classpath=/oracle/oraweb/portlets
  ```

  A könyvtárnevet itt is cseréljük.
- Indítsuk újra az Oracle HTTP Server-t. A beállítások ellenőrzéséhez egy böngészőbe írjuk be: `http://`*<szervernév>*`:`*<port>*`/servlets/OrawebProvider`. Az oldalon a szolgáltató beállításairól kapunk információt.

Vegyük fel a szolgáltatót a Portal portletszolgáltatói közé:

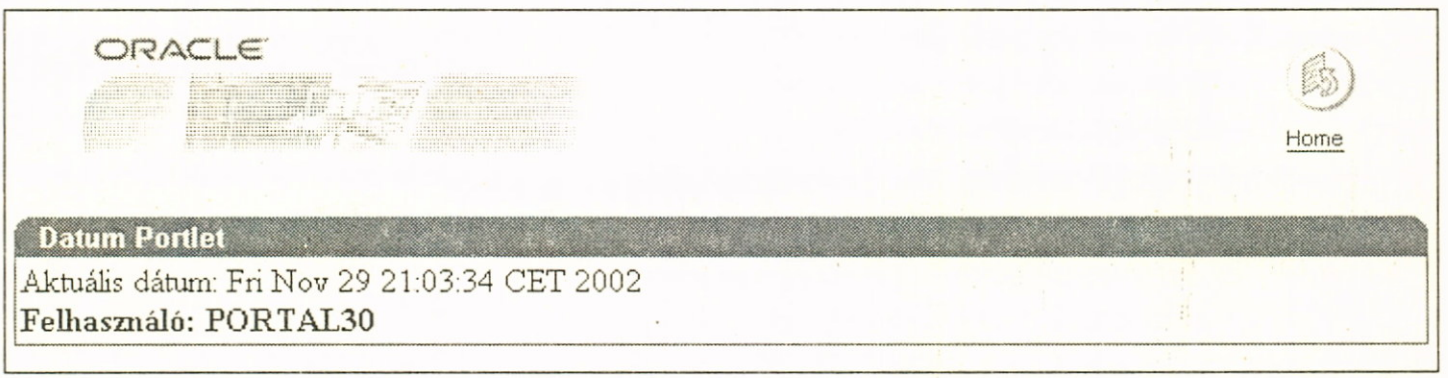
- Kattintsunk az Oracle Portal Home Page oldal adminisztrál lapján a `Provider` / *Szolgáltató* portletben az `Add a Portlet Provider` / *Portletszolgáltató hozzáadása* linkre.

- Töltsük ki az űrlapot:

Name / *Név*	OrawebProvider
Display Name / *Megjelenített név*	Oraweb Provider
Timeout / *Időkorlát*	100
Timeout message / *Időkorlát hibaüzenet*	Oraweb Provider timeout
Implementation Style / *Implementáció típusa*	Web
Provider Login Frequency / *Szolgáltatóba bejelentkezés gyakorisága*	Once Per User Session
URL	Http://<*szervernév*>:<*port*>/servlets/OrawebProvider

- Kattintsunk az OK gombra.

A szolgáltató teszteléséhez az Oracle Portal Home Page oldal adminisztrál lapján a Provider / *Szolgáltató* portletben kattintsunk a Display Portlet Repository / *Portlettár megjelenítése* linkre. Itt válasszuk az Oraweb Provider linket, majd a Datum Portlet linket (4.24. ábra). A portletet felhasználhatjuk a portáloldalakon.



4.24. ábra. *Dátum Portlet*

4.6.2. JSP-alkalmazás portletben

A portletek üzemmódjait különböző *forráskészítők* valósítják meg. Az előző portletben ez egy Java-osztály volt. Most létrehozunk egy újabb portletet, ahol a megjelenítés (show) módot egy JSP-oldal, a névjegy (about) módot egy statikus HTML-oldal valósítja meg.

Hozzuk létre az Oraweb/datum/ könyvtárat a HTTP-szerver dokumentumgyökerében. Például: /oracle/product/iAS/Apache/Apache/htdocs/Oraweb/datum. A könyvtárban hozzuk létre az index.jsp oldalt:

```
<%@page import="oracle.portal.provider.v1.*"%>
<%@page import="oracle.portal.provider.v1.http.*" %>
```

```
<center>
<h2>JSP D&#225;tum Portlet</h2>
<br>
Aktu&#225;lis d&#225;tum: <%= new java.util.Date() %><br>
<%
    PortletRenderRequest pr = (PortletRenderRequest)
        request.getAttribute(HttpProvider.PORTLET_RENDER_REQUEST);

    if (pr != null) {
%>
<b>Felhaszn&#225;l&#243;: <%= pr.getUser().getName() %></b><br>
<%
    }
%>
</center>
```

Próbáljuk ki az oldalt: `http://<szervernév>:<port>/Oraweb/datum/index.jsp`. Az oldalon nem jelenik meg felhasználónév, mivel a JSP-oldalt nem a Portalon keresztül értük el.

Hozzuk létre a könyvtárban `about.html` névvel a névjegy HTML-oldalát:

```
<HTML>
<BODY>
Ez a n&#233;vjegy oldal.
</BODY>
</HTML>
```

A `provider.xml` állományba vegyük fel az új portletet az első után:

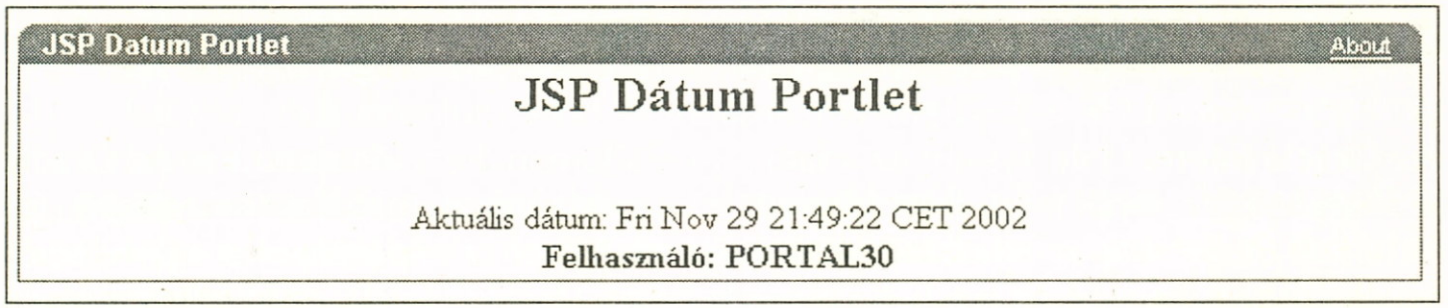
```
...
</portlet>

<portlet class="oracle.portal.provider.v1.http.DefaultPortlet">
   <id>2</id>
   <name>JSPDatumPortlet</name>
   <title>JSP Datum Portlet</title>
   <description>
      Kiirja az aktualis datumot es az aktualis felhasznalo nevet.
      JSP-t es statikus HTML-oldalt hasznal.
   </description>
   <timeout>10</timeout>
   <timeoutMessage>Timeout</timeoutMessage>
   <acceptContentType>text/html</acceptContentType>
   <hasAbout>true</hasAbout>
   <renderer class="oracle.portal.provider.v1.RenderManager">
      <appPath>/Oraweb/datum/</appPath>
      <appRoot>/oracle/product/iAS/Apache/Apache/htdocs/Oraweb/datum/
      </appRoot>
      <showPage>index.jsp</showPage>
      <aboutPage class="oracle.portal.provider.v1.http.FileRenderer">
         <name>about.html</name>
```

```
        </aboutPage>
      </renderer>
    </portlet>

</provider>
```

Indítsuk újra az Oracle HTTP Servert. Szerkesszük az Oracle Portal Home Page oldalon az `Oraweb Provider` szolgáltatót. A szerkesztés oldalon kattintsunk a `Refresh Provider` / *Szolgáltató frissítése* gombra. A portálszolgáltatók közt ellenőrizzük, hogy megjelent a `JSP Datum Portlet`. A portleten megjelenik a felhasználónév (4.25. ábra), a fejlécén látható az `About` / *Névjegy* link, erre kattintva megjelenik `about.html` tartalma.



4.25. ábra. *JSP Dátum Portlet*

Ebben a példában a JPDK API-nak csak egy töredékét használtuk fel. Bonyolultabb portletek tervezésekor tartsuk szem előtt a portletek testre szabhatóságát és a hozzáférés szabályozását.

5. FEJEZET

Az Oracle Internet File System

5.1. Általános áttekintés

Az Oracle Internet File System az adatbázisok állománykezelő rendszere. Az Oracle *i*FS ugyanúgy jelenik meg, mint bármely más csatlakoztatott hálózati meghajtó, így nem kell törődnünk az adatbázis struktúrájával. Mivel az állományokat egy relációs adatbázisban, nem pedig egy helyi merevlemezes tárolón tárolja, ezért számos olyan funkció áll rendelkezésünkre, melyek a szabványos állománykezelő rendszerekkel nem valósíthatók meg, mint például:

- Az Oracle *i*FS-ben tárolt állományok automatikusan indexelhetők, az állományokra és azok tartalmára komplex keresések végezhetők.
- Komplex biztonsági modellt alakíthatunk ki könnyedén, az állományokra és könyvtárakra egyéni és csoportos hozzáférési szabályokat alkalmazhatunk egyszerű parancsok kiadásával.
- Az Oracle *i*FS teljes funkcionalitása elérhető néhány ismerős felületen.
- Az Oracle *i*FS 150 állománytípust tart nyilván, mely lista tovább bővíthető, valamint kifinomult tartalomkezelő eszközöket támogat felhasználói testreszabás nélkül.

Az Oracle *i*FS lehetővé teszi, hogy vagy egy Windowsos, vagy egy webes felületen dolgozzunk, melyek kinézete bár eltérő, mégis ugyanazon alapműveletek végrehajtására alkalmasak.

Az Oracle *i*FS-t a következőképpen érhetjük el:

- E-mail kliensekről,
- FTP-kliensekről,
- hálózati mappákból – ezek a Microsoft Explorer bővítései, melyek lehetővé teszik állományok és könyvtárak elérését WebDAV-on keresztül. A WebDAV a HTTP-protokoll azon bővítéseit tartalmazza, melyekkel a felhasználók távoli webszervereken tárolt állományokat párhuzamosan érhetnek el és kezelhetnek.

Az Oracle *i*FS a következő komponensekből áll:

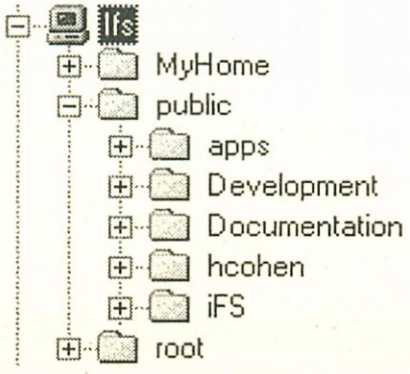
- Az Oracle *i*FS *adatszótára* egyszerű tárolást biztosít mind a hagyományos formátumú, mind a felhasználó által definiált típusú állományok számára. Minden állományt egységesen kezel, függetlenül a használt protokolltól.
- Az Oracle *i*FS-t különböző *kliens szoftvereken* keresztül érhetjük el (ilyen például a Windows Explorer), de saját ügyfélprogramot is alkalmazhatunk adataink elérésére. A kliens és az Oracle *i*FS szerver közötti kommunikációhoz használt protokoll a TCP/IP.
- Az Oracle *i*FS-hez szállított *protokollszerverek* (SMB, HTTP, FTP, IMAP4 és SMTP) a szabványos klienstől érkező parancsokat adatszótár-műveletekké konvertálják.
- Ha alkalmazásunkban szükséges, az Oracle *i*FS *dokumentumhierarchiája* saját dokumentumosztályokkal bővíthető. Ezen osztályok számára attribútumokat XML-dokumentumokkal definiálhatunk, míg viselkedést Java nyelven implementálhatunk.
- Az Oracle *i*FS eléréséhez és az állománykezelő műveletek végrehajtásához különféle osztályokat és metódusokat tartalmazó *Java alapú API* áll rendelkezésünkre.

5.1.1. Az Oracle9*i*FS elérése

A Windowsos kezelőfelület

A Windowsos kezelőfelületen keresztül az Oracle *i*FS állományokat és mappákat a Microsoft Windows 95, 98, 2000 és NT operációs rendszereknél megszokott módon láthatjuk. Az Oracle *i*FS szerver az asztalon található My Computer (Sajátgép) és a Network Neighborhood (Hálózati Helyek) ikonokon keresztül, valamint a Microsoft állománykezelő szoftveréből, a Windows Explorerből válik elérhetővé. Az Oracle *i*FS szerver ugyanúgy jelenik meg, mint bármely más meghajtó a Windows Explorerben (5.1. ábra).

További speciális tartalomkezelő eszközöket telepíthetünk gépünkre, mint például ki- és bejelentkeztetést, verziókezelést és az állományokban történő keresést.



5.1. ábra. *Az* Oracle *i*FS *a Windows Explorerben*

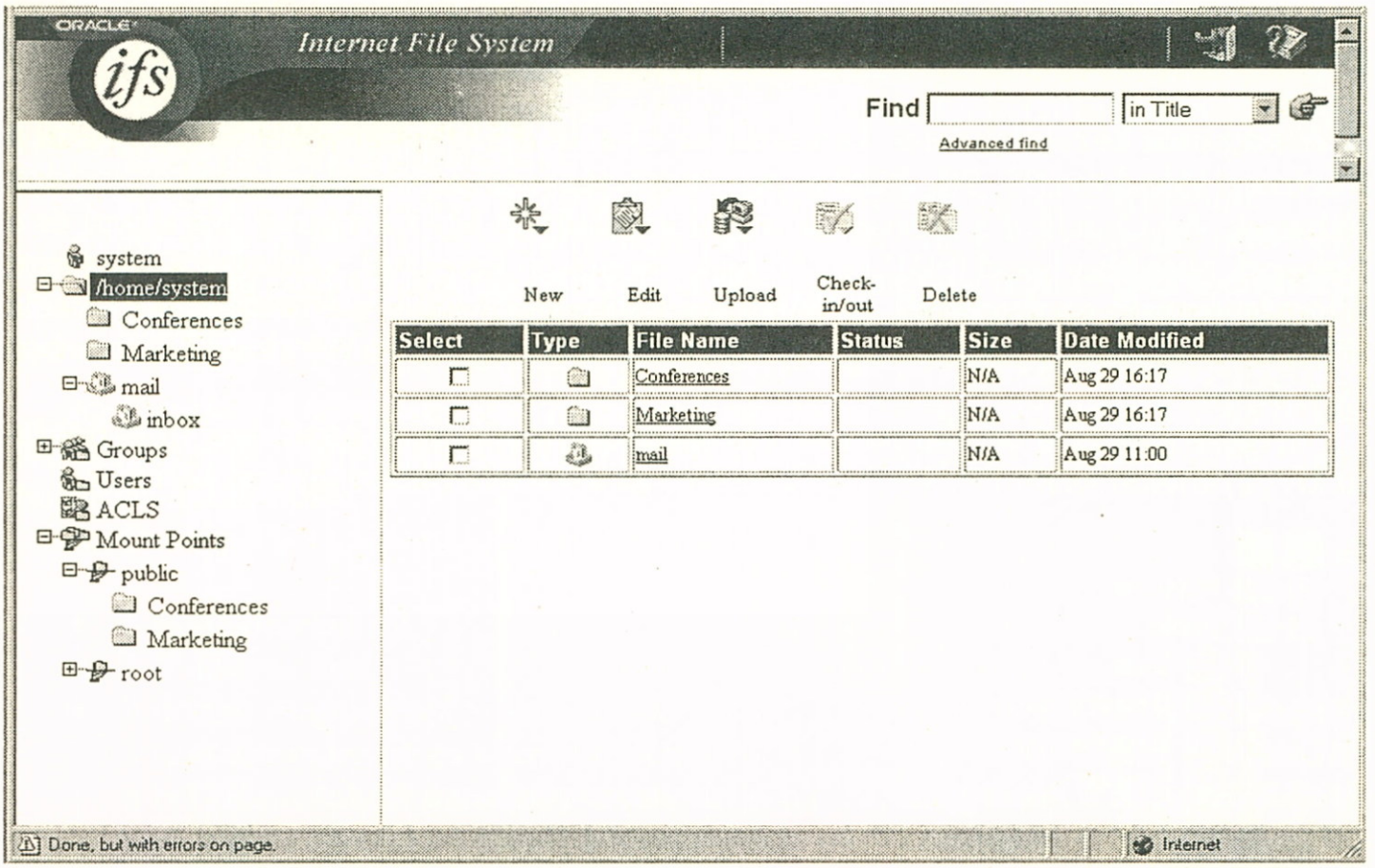
A webes kezelőfelület

A Windowsos kezelőfelület összes funkciója megtalálható a webes felületen is, továbbá elérhetőek az alapvető adminisztrációs eszközök is (amennyiben van adminisztrációs jogosultságunk). Az Oracle *i*FS webes interfészen keresztüli eléréséhez nincs szükségünk semmilyen szoftverre vagy segédeszközre. Mindössze a Netscape Navigator 4.7 vagy az Internet Explorer 5 (vagy ezek későbbi verziószámú) böngészőkbe be kell gépelni az *i*FS kiszolgáló címét, mely többnyire a szerver neve, például `http://omega.info2000.hu`.

A webes kezelőfelület három keretre van osztva (5.2. ábra). A legfelül található keret olyan vezérlőelemeket tartalmaz, melyekkel kijelentkezhetünk, igénybe vehetjük az on-line segítséget vagy kereshetünk meghatározott állományokat, illetve állományokban.

A bal oldali keretben egy könyvtárfa található, mely *i*FS mappáink hierarchikus struktúráját ábrázolja, beleértve a biztonsági és az elérhetőségi információkat tartalmazó mappákat is.

A jobb oldali keret a könyvtárfában kiválasztott aktuális könyvtár tartalmát mutatja, mely állományokból és könyvtárakból állhat. Az állományok listája felett található az eszköztár, mely Oracle *i*FS funkciókat tartalmazó menüket rejtő ikonokból áll.

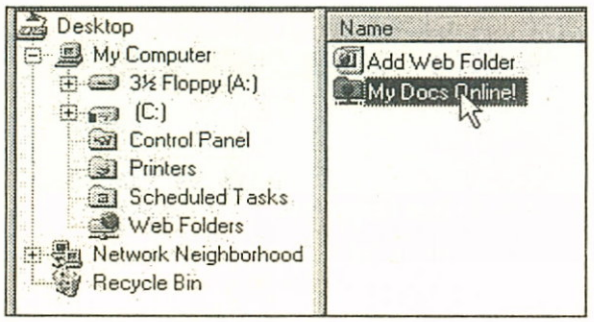


5.2. ábra. *A webes kezelőfelület a három kerettel*

A hálózati mappahierarchia

Hálózati mappák használata esetén azok tartalmát a Windows Explorerben használatos struktúrákhoz hasonlóan rendezzük el. Adataink hálózati mappákon keresztül történő elérésével képesek vagyunk a tartalom megváltoztatására, a verziókezelésre, valamint más Oracle *i*FS funkciók használatára. A mappahierarchiában ezenkívül böngészhetünk, kereshetünk és kezelhetjük hálózati Oracle *i*FS adatainkat is.

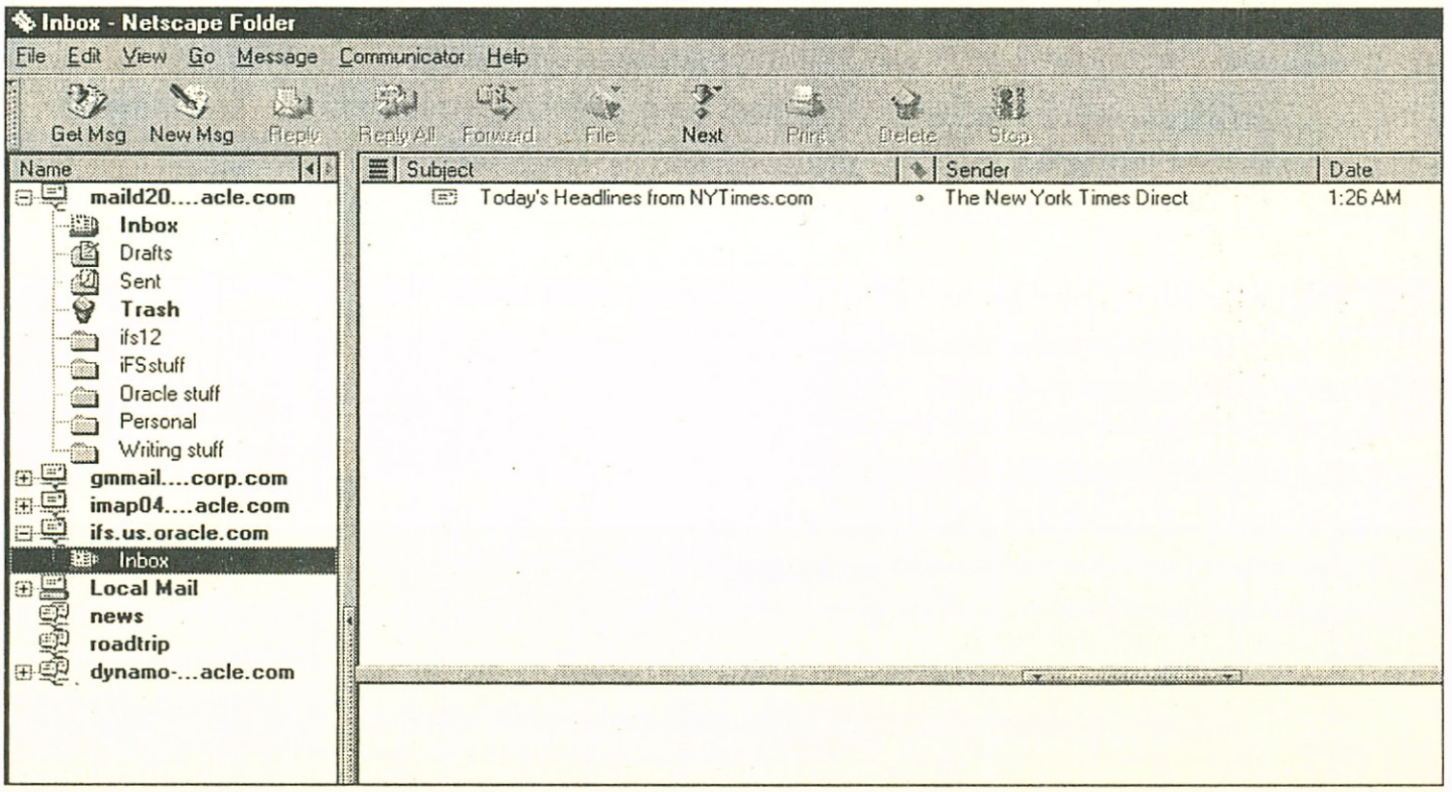
Az 5.3. ábrán láthatjuk, hogyan jelennek meg Oracle *i*FS állományaink, amikor a Windows Explorer segítségével a hálózati mappákban böngésszük őket. A hálózati mappák a Windows Explorer hierarchiájában csatlakoztatott erőforrásként jelennek meg.



5.3. ábra. *Hálózati mappák a Windows Explorer hierarchiájában*

E-mail

Az Oracle *i*FS gyökérkönyvtárának része egy levelezési mappa, mely tartalmaz egy postaládát is. Ez lehetővé teszi, hogy elektronikus leveleinket az Orcale *i*FS-ben ment-



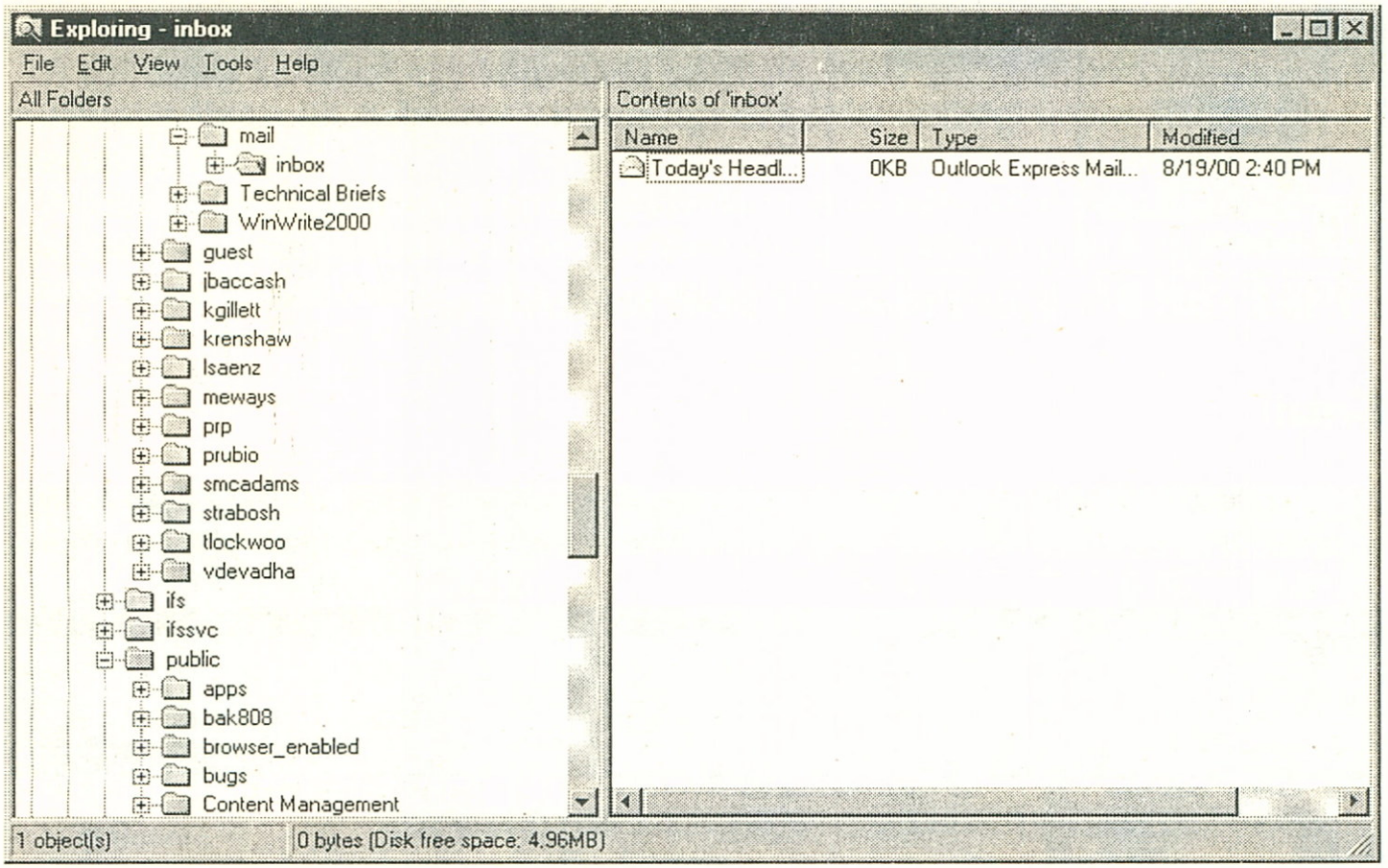
5.4. ábra. Oracle *i*FS *elérése e-mail kliensen keresztül*

sük el, és azokat a Windowsos vagy webes felületeken keresztül olvashassuk el. Így, amikor adataink között keresünk, a keresésbe bevonhatjuk elektronikus leveleinket és azok csatolt állományait is.

A rendszeradminisztrátor beállíthatja, hogy az elektronikus leveleket közvetlenül az Oracle *i*FS postaládába kapjuk, így anélkül olvashatjuk el üzeneteinket, hogy fel kellene jelentkeznünk egy e-mail kliensre (5.4. ábra). Sőt, levelező programunkban beállíthatunk egy IMAP könyvtárat az Oracle *i*FS szerverre.

Az e-mail állományok verziókezelése, valamint a ki- és bejelentkeztetés nem engedélyezett.

Ezután e-mail üzeneteinket bemásolhatjuk az Oracle *i*FS-be, így kezelhetjük azok tartalmát, és kereséseket is végrehajthatunk rajtuk (5.5. ábra).



5.5. ábra. Az Oracle *i*FS-be *másolt elektronikus levelek a levelezési mappában jelennek meg*

FTP

Sokan használnak állományaik másolására FTP-t (File Transfer Protocol), különösen az általuk készített honlapok interneten való publikálására, vagy pedig nagy méretű vagy nagy mennyiségű állományok feltöltésére. Az állományok le- és feltöltésénél az FTP-kliens az Oracle *i*FS meghajtót ugyanúgy kezeli, mint bármely más meghajtót a rendszerben.

Az FTP nem engedélyezi az állományok verziókezelését, továbbá a ki- és bejelentkeztetést.

5.1.2. Az *i*FS előnyei a szabványos állománykezelő rendszerekkel szemben

Az Oracle *i*FS használatával az állományrendszerek rugalmasságát és könnyű kezelhetőségét az adatbázisok megbízhatóságával kombináljuk. Az 5.1. táblázatban összehasonlítjuk az Oracle *i*FS-t a szabványos állománykezelő rendszerekkel.

5.1. táblázat. *A szabványos állományrendszerek és az* Oracle *i*FS *jellemzői*

Jellemző	*Szabványos állományrendszer*	Oracle *i*FS
Az állományok elérése	A tűzfaltól függően korlátozott lehet az állományok hálózaton keresztül történő elérése.	Több elérési mód áll rendelkezésünkre: Windows, WWW, FTP, e-mail, WebDAV.
Integrált tárolás	A különböző típusú állományokat gyakran speciális szerverek tárolják.	A különböző típusú állományokat ugyanaz a könyvtár tárolja: elektronikus levelek, webes állományok, szöveges dokumentumok stb. Egyetlen interfészt használunk mindegyik állománytípus kezelésére.
Verziókezelés	Amennyiben az állomány több változatát is szeretnénk elmenteni, különböző, egyedi neveket kell nekik adnunk, és saját mappájukban kell tárolnunk őket. Kézzel kell megszerveznünk ezen állományok tárolását.	Mikor elmentünk egy állományt, lehetőségünk van megtartani az előző változatokat. Az Oracle *i*FS szervezi ezek kezelését. Mindegyik változatot ugyanabban a könyvtárban tárolja.
Állományok konkurens elérése	Az elérés „mindent vagy semmit" típusú: ha mások elérhetik állományainkat, felülírhatják őket. Ha nem, saját másolatokkal kell dolgozniuk, és a végén kézzel kell összefésülniük őket.	A ki- és bejelentkeztetéssel lehetőségünk van zárolni állományainkat, hogy megvédjük a más felhasználók általi felülírástól. A biztonsági beállításoknál meghatározhatjuk, kinek van jogosultsága hozzáférni az egyes állományokhoz, valamint hogy a többi felhasználó felülírhatja-e őket.
Keresés	A különböző típusú állományok, mint az elektronikus levelek és a szöveges dokumentumok, különböző állományrendszerekben vannak tárolva, így több keresést is végre kell hajtanunk, hogy ugyanazon témakörhöz megtaláljuk az összes vonatkozó információt. A tartalom alapú keresés lassú és bonyolult.	Az Oracle *i*FS képes a különböző típusú állományokat együtt tárolni, így csupán egyetlen keresésre van szükségünk az egy témához tartozó információk megtalálására. A tartalom alapú keresés nemcsak gyors, hanem kifinomult is.

5.1.3. Az *i*FS előnyei a mindennapi munka során

Az Oracle *i*FS felhasználójaként érdemes tudnunk, hogyan segítenek ezen termék eszközei, hogy munkánkat gyorsabban és kényelmesebben végezhessük el. Ezt foglalja össze az 5.2. táblázat.

5.2. táblázat. *Hogyan segíti az* Oracle *i*FS *a felhasználók munkáját*

Követelmény	*Az iFS lehetőségei*	*Előny*
Állományok elérése és kezelése	Az állományok egységes elérése ismerős felületeken.	Nem kell megtanulnunk egy újabb kezelőfelületet az Oracle *i*FS-ben tárolt állományok kezeléséhez.
Különböző típusú állományok kezelése	A különböző állománytípusok egységes elérése. Egyetlen felületen az állományok sokféle típusával dolgozhatunk: szöveges dokumentumok, táblázatok, honlapok, audio- és videoállományok, elektronikus levelek, XML.	Egyetlen bejelentkezéssel az összes állományt elérhetjük, csak egyetlen felhasználói nevet és jelszót kell megjegyeznünk.
Ugyanazon állomány több változata szükséges	Verziókezelés.	A verziókezelés lehetővé teszi minden egyes állomány korábbi változatainak megtartását, így nyomon követhetjük az állomány fejlődését. Ha a jelenlegi változtatásokkal nem vagyunk elégedettek, előkereshetjük egy korábbi változatát, nem kell emlékezetből újraírnunk az egészet.
Konkurens elérés	Ki- és bejelentkeztetés, zárolás.	A ki- és bejelentkeztetéssel korlátozhatjuk az állományok elérhetőségét. Amíg az állományon dolgozunk, kijelentkeztetjük azt, így mások olvashatják, de nem írhatják felül.
Keresés különböző típusú állományokon, például honlapokon, elektronikus leveleken és szöveges dokumentumokon	Kombinált keresések.	Mivel az Oracle *i*FS különböző típusú állományokat is képes tárolni, egyetlen keresés is megadja a keresési feltételnek megfelelő állományok listáját.
Adatok háttértárra mentése	Adatbázis szintű mentés.	Mivel az Oracle *i*FS egy Oracle adatbázisban található, használhatjuk a mentést és visszaállítást munkaállományaink védelmének érdekében.

5.2. Az Oracle *i*FS elérése

Az Oracle *i*FS-t az alábbiakon keresztül érhetjük el:

- Microsoft Windows Explorer,
- Microsoft Network Neighborhood,
- böngésző,
- hálózati mappák,
- FTP-kliens,
- levelező kliensprogram.

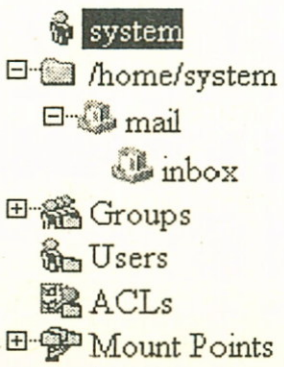
Az első lépés egy Oracle *i*FS felhasználói név beállítása, mellyel ezután be tudunk jelentkezni az Oracle *i*FS-be és használni tudjuk azt. Amennyiben a Windowsos kezelőfelületen keresztül is szeretnénk használni az Oracle *i*FS tartalomkezelő szolgáltatásait, telepítenünk kell az *Oracle iFS Utilities*-t. Ezek a segédeszközök az olyan szolgáltatásokhoz biztosítanak elérést, mint a verziókezelés, a ki- és bejelentkeztető funkciók, valamint az állományok zárolása. Amennyiben a webes felületen szeretnénk dolgozni, ezt a telepítést nem kell elvégeznünk.

Oracle *i*FS felhasználói fiókot a rendszeradminisztrátor tud létrehozni. Minden esetben az őáltala megadott felhasználói nevet és jelszót használjuk, függetlenül az elérés módjától.

A felhasználói fiók a következő három alapkönyvtárat tartalmazza (5.6. ábra):

- `/home/`*<felhasználói_név>* – Személyes könyvtárakat tartalmaz, valamint más könyvtárakra mutató hivatkozásokat. Más felhasználók nem láthatják ezt a könyvtárat, kivéve ha azt engedélyezzük.
- `mail` – E-mail üzeneteinket tárolja. Ez a könyvtár a gyökérkönyvtárunk egyik alkönyvtára.
- `inbox` – A `mail` könyvtár alkönyvtára, alapesetben ez tárolja bejövő üzeneteinket.

A rendszeradminisztrátor beállításaitól függően találhatunk egy `public` könyvtárat is, melyben a más felhasználók előtt is publikus állományainkat tárolhatjuk.



5.6. ábra. *A könyvtárfa, melyben minden felhasználó rendelkezik* `home` *könyvtárral*

5.2.1. Az Oracle *i*FS elérése a Windowsos kezelőfelületen keresztül

Először is létre kell hoznunk egy hálózati meghajtót, mely az Oracle *i*FS könyvtárra mutat. Ez azt jelenti, hogy az Oracle *i*FS könyvtárhoz egy betűt (például O:) rendelünk, mellyel ezután az Oracle *i*FS állományokra ugyanúgy hivatkozhatunk, mint bármely más, a helyi vagy hálózati háttértárakon tárolt állományokra.

Ha nem ismerjük a kiszolgáló nevét, rá is kereshetünk a Network Neighborhood ablakban, ahol [amennyiben a nézet részletes listára (`Details`) van állítva] mindegyik Oracle *i*FS meghajtó az „Oracle Internet File Server" címkét kapja. Ha megtaláltuk a kiszolgálót, jegyezzük meg az elérési útvonalat, és így már létre is hozhatjuk a hálózati meghajtónkat. De a rendszeradminisztrátort is segítségül hívhatjuk a szerver megtalálásához.

Második lépésben telepítenünk kell az *Oracle iFS Utilities* segédprogram-csomagot. Bár ez nem kötelező, enélkül is használható a Windowsos kezelőfelület, mégis érdemes, mert sok hasznos segédeszközt tartalmaz.

5.2.2. Az Oracle *i*FS elérése webes kezelőfelületen keresztül

Az Oracle *i*FS webes kezelőfelületével az Oracle *i*FS állományokat egy böngésző segítségével érhetjük el. A webes felület biztosítja mindazokat a funkciókat, melyeket a Windowsos felület, sőt további funkciók is rendelkezésünkre állnak.

A webes felület használatához nincs szükségünk speciális segédeszközökre vagy a böngésző bővítésére (csak engedélyeznünk kell a JavaScript és a cookie-k használatát). Felhívjuk a figyelmet arra, hogy a Java és az ActiveX nem követelmény a webes interfészhez. Jelenleg a következő böngészők alkalmasak:

- Netscape Navigator 4.7,
- Microsoft Internet Explorer 5 és 5.5.

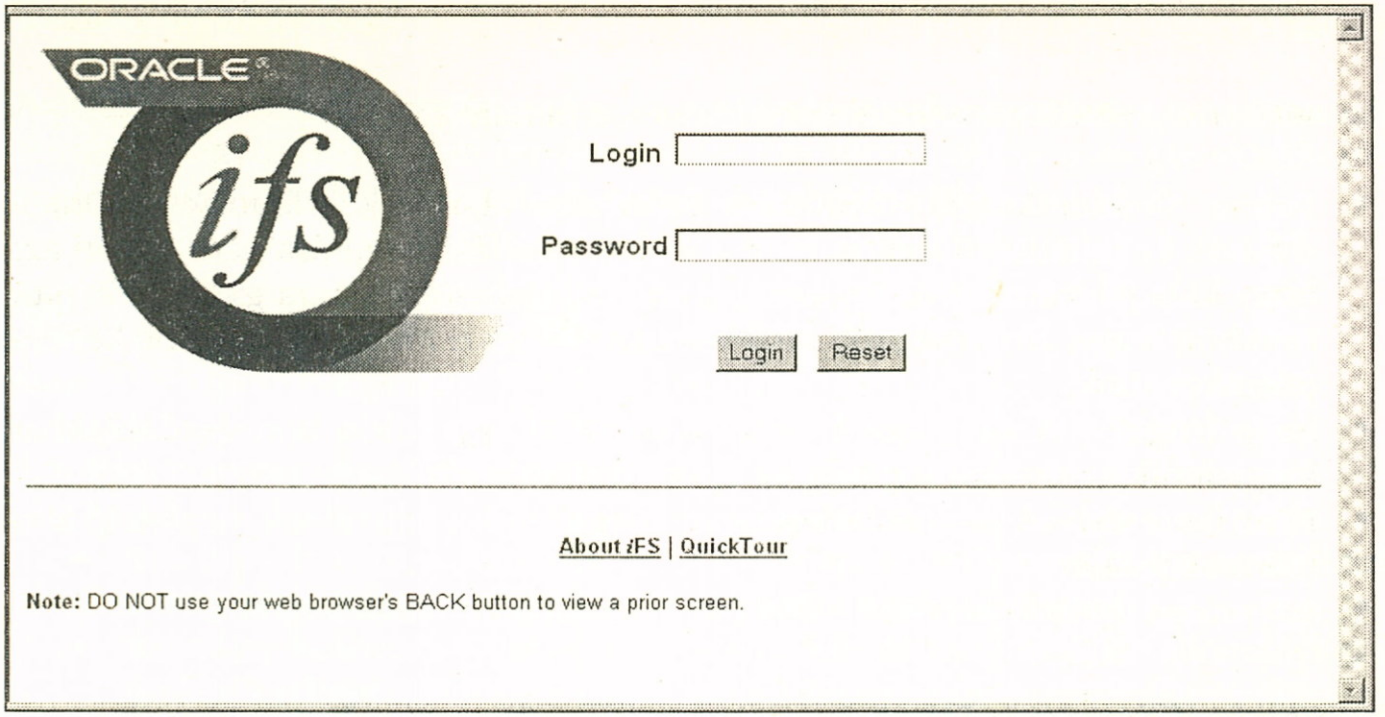
Az Oracle *i*FS-be való bejelentkezéshez először be kell gépelnünk az Oracle *i*FS kiszolgáló címét, például: `http://<szerver_neve>/ifs/webui/`.

A szerver neve vagy a rendszeradminisztrátor által a kiszolgálóhoz rendelt név, vagy annak IP-címe – ezeket az adminisztrátortól kell megkérdeznünk. Az `/ifs/webui/` könyvtárban található a bejelentkező lap.

A bejelentkező lap megjelenése után be kell gépelnünk felhasználói nevünket és azt a jelszót, melyet a rendszeradminisztrátor adott (5.7. ábra).

Az Oracle *i*FS lehetőségeinek megtekintéséhez válasszuk a `QuickTour` feliratot!

A webes felület bezárásával nem jelentkezünk ki az Oracle *i*FS-ből, így ha később valaki a böngésző `Vissza` gombjával újra megjeleníti a felületet, anélkül használhat-

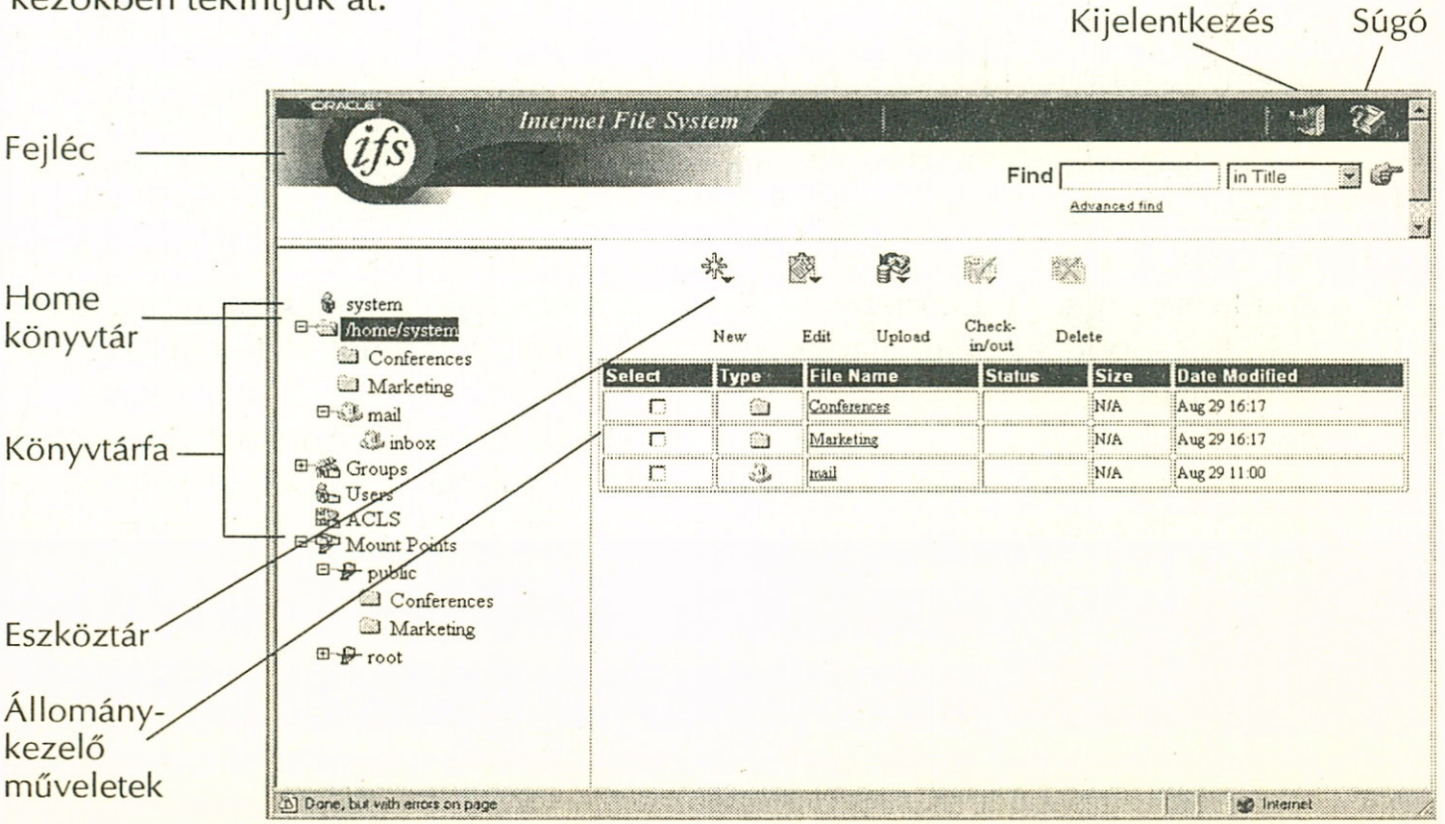


5.7. ábra. *A webes interfész bejelentkező oldala*

ja, hogy be kellene jelentkeznie. Ezért, ha befejeztük a munkát a webes felületen, mindenképpen jelentkezzünk ki és zárjuk be a böngészőt!

A webes felület részei

A bejelentkezés után megjelenik a webes felület (5.8. ábra), melynek részeit a következőkben tekintjük át.



5.8. ábra. *A webes kezelőfelület*

Banner

A Banner keret globális parancsok vezérlőelemeit tartalmazza:

- `Logout` (kijelentkezés) – Munkánk befejezése után ki kell lépnük az Oracle *i*FS-ből. Ez feltétlenül fontos, mert a webes interfész bezárása nem jelenti azt, hogy ki is jelentkeztünk, így később anélkül térhetünk vissza gyökérkönyvtárunkhoz (vagy, ami rosszabb, más felhasználó teszi meg ezt), hogy be kellene jelentkeznünk.
- `Help` (súgó) – A `Help` gomb megnyomásával egy külön böngészőablakban megjelenik az on-line segítség az egyes *i*FS funkciókhoz.
- `Find` (keresés) – A keresés befejeztével megjelenik a feltételnek megfelelő állományok listája.
- `Advanced` (haladó keresés) – Bővített keresést hajthatunk végre.

Könyvtárfa (Directory Tree)

A könyvtárfa az elérhető Oracle *i*FS könyvtárak hierarchikus listája. Alapesetben hat elem van ide telepítve:

- `<user>` (felhasználói nevünk) – Itt nézhetjük meg személyes beállításainkat, például megváltoztathatjuk a jelszót.
- `<userdirectory>` (gyökérkönyvtárunk) – Ha erre kattintunk, megjelennek az alkönyvtárak is. Amelyik könyvtár neve előtt + jel áll, az újabb alkönyvtárakat tartalmaz, melyeket rákattintva terjeszthetünk ki.
- `Groups` (felhasználói csoportok) – Akkor érdemes felhasználói csoportot létrehozni, ha több felhasználónak ugyanazokat a jogosultságokat szeretnénk adni, így nem kell mindegyiket külön-külön beállítani.
- `Users` (felhasználók) – Ez az ikon az aktuális Oracle *i*FS felhasználókat mutatja, akiket hozzáadhatunk egy általunk létrehozott csoporthoz.
- `ACLs` – Ez az ikon a jelenleg számunkra elérhető Access Control Lists (ACLs) listákat mutatja. Az állományainkhoz és a mappáinkhoz minden egyes felhasználó és csoport számára alapértelmezett, valamint általunk definiált ACL-eket rendelhetünk.
- `Mount Points` (csatolási pontok) – Ez az ikon az Oracle *i*FS információk eléréséhez létrehozott további belépési pontokat mutatja, melyeket a rendszeradminisztrátor definiált. Alapesetben csak a gyökérkönyvtárunkon keresztül érhetjük el az állományokat.

Eszköztár (Toolbar)

Az eszköztár a kiválasztott állományokon végrehajtható műveleteket tartalmazza (5.9. ábra). Ha egy menü le van árnyékolva (le van tiltva), akkor a megfelelő állományt vagy mappát ki kell választani a funkció végrehajtása előtt.

5.9. ábra. *A webes kezelőfelület eszköztára*

- `New` (új) – Ha erre kattintunk, új ACL-t, mappát, felhasználót (ha rendelkezünk a megfelelő jogosultsággal) vagy csoportot hozhatunk létre.
- `Edit` (szerkesztés) – Kiválasztása esetén a felbukkanó menüben állománykezelési funkciókat hajthatuk végre, ezek például állományok és mappák másolása, mozgatása, átnevezése, továbbá a verziók megtekintése vagy ACL-ek alkalmazása.
- `Upload` (feltöltés) – Ez az ikon egy vagy több állomány Oracle *i*FS-be történő feltöltésére szolgál egy speciális ablak segítségével.
- `Check in/out` (ki- és bejelentkeztetés) – Ezen ikon segítségével tartalomkezelő funkciókat érhetünk el. Az igényelt állományt kijelentkeztethetjük, hogy mások ne változtathassák meg addig, amíg újra be nem jelentkeztetjük. Az állományokat zárolhatjuk, hogy mások ne tudják felülírni vagy mozgatni. Itt állíthatjuk be azt is, hogy az Oracle *i*FS tárolja az állományok korábbi változatait is, és itt térhetünk vissza egy korábbi verzióhoz.
- `Delete` (törlés) – Törli a kiválasztott állományokat és mappákat.

Állománylista (File List)

Az állománylistában állománykezelő műveleteket hajthatunk végre (5.10. ábra).

Select	Type	File Name	Status	Size	Date Modified
☐		Chapters		N/A	August 11, 2000 11:39:55 AM PDT
☐		Conferences		N/A	August 9, 2000 2:08:22 PM PDT
☐		Marketing		N/A	August 9, 2000 2:08:06 PM PDT
☐		mail		N/A	August 2, 2000 10:36:47 AM PDT

5.10. ábra. *A webes felület állománylistája*

Az állománylista a következő oszlopokkal rendelkezik:

- `Select` (kiválasztás) – Itt jelölhetjük ki, hogy mely állományokra szeretnénk végrehajtatni a műveletet. Az eszköztár menü `Select all` pontja az összes állományt kiválasztja.

- `Type` (típus) – Ez mutatja az állomány típusát, mely mappa, állomány, grafika, postaláda, audio, film vagy videoklip lehet.
- `File name` (állomány neve) – Itt láthatjuk az állomány nevét. Ha az állomány hivatkozás, akkor a neve dőlt betűkkel jelenik meg, míg a verziókezelt állományoké vastagon szedve.
- `Status` (státus) – A mappa vagy az állomány elérhetőségét mutatja, mely elérhető, kijelentkeztetett, zárolt vagy csak olvasható lehet. (Zárolt állományok esetén külön jelöli, ha jogunk van a zárolást megszüntetni.)
- `Size` (méret) – Itt jelenik meg az állományok mérete KB vagy MB mértékben. A mappák méretét nem mutatja.
- `Data Modified` (utolsó módosítás) – Az utolsó módosítás dátuma és időpontja.

Navigáció a mappák között

Két módszer van a navigációra:

- Rákattintunk a kiterjesztendő mappára az állománylistában.
- Kiválasztjuk a megfelelő mappát a könyvtárfában. A + jellel kiterjeszthetjük az alkönyvtárakat tartalmazó könyvtárakat is, így azok láthatóvá válnak.

5.2.3. Hálózati mappák

A hálózati mappák a Microsoft Explorer olyan bővítései, melyekkel kiszolgálókhoz a WebDAV protokollon keresztül csatlakozhatunk. Bár a hálózati mappák ugyanúgy jelennek meg, mint bármely más hálózati erőforrás a Windows Explorerben, azonban más protokollt használ, mint azok. Ha a hálózati mappákon keresztül kapcsolódunk az Oracle *i*FS-hez, ugyanazt a tartalmat látjuk, mintha a hagyományos SMB protokollon, egy böngészőn, FTP- vagy e-mail kliensen keresztül látnánk azt.

Jelenleg a hálózati mappákat a Microsoft Internet Explorer 5 és 5.5 verziója támogatja Windows 95/98/NT operációs rendszerek alatt, melyekre telepítenünk kell ezeket. A Windows 2000-ben és az MS Office 2000-ben automatikusan telepítve vannak, és a `Save to the web` pont alatt találjuk meg. Telepítés után a hálózati mappák az operációs rendszer részévé válnak, és más mappákhoz hasonló módon érhetők el: Explorerben az `Open as Web Folder` menüpontban, vagy valamely Office 2000 alkalmazásban.

Office 2000-ben ugyanazon a helyen kezelhetjük a korábbi változatait megtartó állományokat, egyébként pedig egy helyi másolatot kell szerkesztenünk, aztán visszamásolnunk azt az eredeti helyére.

5.2.4. Az Oracle *i*FS elérése FTP-n keresztül

Az Oracle *i*FS elérésére és használatára az FTP is módot ad, amely különösen nagy számú állomány másolására hasznos. Mikor FTP-n keresztül töltünk fel állományokat az Oracle *i*FS-be, azokat közvetlenül egy adatbázis táblájába másolja, míg az FTP-kliens ezt úgy érzékeli, mintha egy hagyományos FTP-szerverrel kommunikálna. Lehetőségünk van állományok fel- és letöltésére, de nem tudjuk kezelni a korábbi verziókat, sőt a ki- és bejelentkeztetés sem lehetséges.

Az FTP-n keresztüli bejelentkezéshez ugyanazt a felhasználói nevet és jelszót kell használnunk, mint a webes vagy a Windowsos interfészhez, de bejelentkezhetünk név nélkül (anonymousként) is.

5.2.5. Az Oracle *i*FS elérése e-mailen keresztül

Amennyiben vállalatunk úgy dönt, hogy az Oracle *i*FS szervert levelező kiszolgálóként is felhasználja, levelezésünket továbbra is a megszokott módon intézhetjük. Az alapvető e-mail funkciókat ugyanúgy elérhetjük, mintha hagyományos kiszolgálóhoz kapcsolódtunk volna.

A különbség abban áll, hogy leveleinket nemcsak a szokásos levelezőprogramunkból érhetjük el, hanem bármely másik protokollon keresztül is, például a Windows állománymegosztásán, a weben, az FTP-n vagy a szabványos e-mailen keresztül. Az Oracle *i*FS használatának elsődleges előnye, hogy a kapott leveleket, valamint csatolt állományaikat egyetlen felületen később megtekinthetjük, kereshetünk bennük vagy szerkeszthetjük őket. Jó példa erre, ha a piacról statisztikát szeretnénk készíteni, akkor a keresés tartalmazhatja az erről szóló, elektronikus levélben megkapott jelentéseket is.

Természetesen a megszokott programot is használhatjuk az Oracle *i*FS-en keresztüli levelezés lebonyolítására, miután beállítottuk azt az Oracle *i*FS kiszolgálóra, ám ebben az esetben csak e-mail típusú állományainkat láthatjuk.

Rendelkezni fogunk egy `mail` mappával, amely tartalmazza a beérkező leveleket tároló mappát (`inbox`), valamint az általunk létrehozott könyvtárakat is.

A levelezőprogram konfigurálása az Oracle *i*FS szerver használatára

Mielőtt levelezőprogramunkat elkezdhetnénk használni az Oracle *i*FS szerveren keresztüli levelezés lebonyolítására, néhány beállítást el kell végeznünk rajta. Létre kell hoznunk egy új felhasználói fiókot, mely az Oracle *i*FS szerverre mutat. Ez a már megszokott módon történik, ám szükségünk van néhány speciális információra:

- `Username/Account` – Oracle *i*FS felhasználói nevünk.
- `Password` – Oracle *i*FS jelszavunk.
- `Organization` – Vállalatunk neve.
- `Identified as/Full name` – A `Feladó (From)` mezőben megjelenő nevünk.
- `Outgoing mail server` – Az Oracle *i*FS mint e-mail szerverhez rendelt név.
- `Incoming mail server type` – A beérkező leveleinket fogadó IMAP típusú szerver.
- `Incoming mail server` – Az Oracle *i*FS-hez mint e-mail szerverhez rendelt név.

5.3. Állomány- és tartalomkezelés

5.3.1. Állományok és mappák kezelése

A mappahierarchia áttekintése

A mappahierarchia az Oracle *i*FS mappák szervezett struktúrája. Ez rendszerezi az adatszótárt, így a felhasználók könnyedén képesek böngészni azt. Az Oracle *i*FS lehetővé teszi, hogy több hierarchia létrehozásával a felhasználók különbözőképpen strukturálhassák adataikat.

A kezdeti mappahierarchiát a rendszeradminisztrátor hozza létre, de az adminisztratív jogosultsággal rendelkező felhasználók is dolgozhatnak a hierarchiákkal. Ez különösen nagyvállalatoknál hasznos, ahol önszervező csoportokat lehet így alkotni.

A mappahierarchia, függetlenül az eléréshez használt kezelőfelülettől, mindig ugyanúgy néz ki. Három alapkönyvtár áll rendelkezésünkre:

- `Home` – Minden felhasználó rendelkezik egy saját, a felhasználói nevével megegyező nevű könyvtárral a `home` könyvtárban. Itt tárolhatjuk személyes mappáinkat és állományainkat. Ez egy olyan tárterület az Oracle *i*FS-ben, amely csak a miénk, nekünk van fenntartva. Teljesen a mi ellenőrzésünk alatt áll, és tetszőleges állományokat tölthetünk fel az adminisztrátor által megadott kvótáig. Az itt lévő állományokat más felhasználók nem érhetik el, csak ha külön engedélyezzük azt az állományokhoz rendelt ACL-lista módosításával.
- `Mail` - `home` könyvtárunk tartalmazni fog egy `mail` alkönyvtárat, mely a levelezés lebonyolítására szolgál.
- `Inbox` - A `mail` könyvtár azon alkönyvtára, mely a beérkező leveleket tárolja. Ha más levelező alkalmazást használunk, erre a könyvtárra kell mutatnia postafiókunknak.

A három alapvető mappán kívül még két közös mappa van:

- `Public` – A megosztott információk tárolására szolgál.
- `Root` – A legfelső szinten található mappa, ez alatt található az összes felhasználói és rendszerkönyvtár.

A könyvtárak elérésének szabályai

- Az állományok elérését nem a könyvtár, hanem az állomány ACL-listája szabályozza. Az állomány ACL-je különbözhet a mappáétól.
- A mappa ACL-je azt szabályozza, hogy a felhasználók törölhetik-e a könyvtárt, vagy másolhatnak-e oda vagy törölhetnek-e onnan elemeket.
- Az állomány ACL-je azt szabályozza, hogy ki módosíthatja vagy törölheti azt.
- Egy állomány eléréséhez el kell tudnunk érni a tartalmazó könyvtárat is. Előfordulhat, hogy egy keresés során egy állományt egy olyan könyvtárban találunk meg, amelyhez nincs hozzáférésünk.

Az XML és az Oracle *i*FS

Az Oracle *i*FS bármilyen típusú állományt képes tárolni, beleértve az XML-t is. Sok XML-állomány tartalmaz DTD-t vagy hivatkozik rá. Mivel az Oracle *i*FS beépített DTD-ellenőrzővel rendelkezik, ezért minden egyes XML-dokumentumot validál, valamint meghatározza, hogy jól formáltak-e. Ha egy XML-dokumentum nem felel meg, hibaüzenetet jelenít meg, és nem tölti fel azt. Alapesetben a DTD ellenőrzés nincs bekapcsolva.

Hivatkozások használata

A legtöbb esetben a szabványos állománykezelő rendszerek megfelelőek az információk tárolására és szervezésére. Ám vannak esetek, mikor egyetlen hierarchia nem felel meg igényeinknek. Mikor egy dokumentum több kategóriába is tartozik, jó lenne ugyanazt az állományt több mappában is megjeleníteni. Például ha kaptunk egy levelet Kovács Istvántól a budapesti vásárról, akkor azt három mappába is elhelyezhetjük: a `Levelek`, a `Kovács`, valamint a `Budapest` mappákban.

Egyik megoldás az, hogy az állományból másolatokat készítünk, de ebben az esetben a változások kezelése igen bonyolult. Ehelyett hivatkozásokat hozunk létre mindhárom helyről ugyanarra az állományra.

Az Oracle *i*FS-ben a hivatkozások ugyanazon állomány több mappában való megjelenítésére szolgálnak, áthidalva ezzel a több másolat kezelésének problémáját.

Hivatkozások működése

Ezeket a hivatkozásokat nem szabad összekeverni a Windows által használt hivatkozásokkal. Utóbbiak csak egy helyre mutatnak, nem reprezentálják magát az állományt. Ha töröljük az eredeti állományt, a hivatkozás megmarad, annak ellenére, hogy immáron nem mutat sehova.

Az Oracle *i*FS-ben minden egyes hivatkozás egyaránt reprezentálja az állományt. Amennyiben töröljük az egyiket, a többi még megmarad, de ha az összeset eltávolítjuk, azzal töröljük az eredeti állományt is az adatszótárból.

A hivatkozások létrehozása hasonlít ahhoz a műveletsorhoz, amikor a vágólapra másolunk egy állományt és beillesztjük egy könyvtárba. A Windowsos interfészen létrehozott hivatkozások látszanak a webesen is, és fordítva. A webes felületen a hivatkozások dőlt betűkkel vannak szedve.

A több mappában is tárolt állományokról azt mondjuk, hogy több szülővel rendelkeznek. Egy hivatkozás esetén lehetőségünk van megtudni, mely mappákból hivatkoznak még arra az állományra. Ez különösen hasznos akkor, ha szeretnénk megállapítani, kiknek van hozzáférésük ahhoz az állományhoz. Amennyiben megtaláltuk a szülőket, elvégezhetjük rajtuk a kívánt műveleteket – például megváltoztathatjuk a biztonsági beállításukat.

Az állományok attribútumai

A hagyományos állománykezelő rendszerek az állományokról kétféle információt tartanak nyilván:

- az állomány tartalma;
- egyéb információk az állományról (például a szerző vagy a létrehozás dátuma).

Ezeket az információkat az állományok *tulajdonságainak* vagy *attribútumainak* nevezik. Az Oracle *i*FS a hagyományos állományrendszereknél több információt tárol az állományokról, lehetővé téve, hogy pontosabb keresési feltételeket fogalmazzunk meg. Így keresésünk gyorsabb és precízebb lesz.

Az Oracle *i*FS-ben a rendszeradminisztrátor saját attribútumokat is képes létrehozni.

5.3.2. Tartalomkezelő eszközök

Az Oracle *i*FS-ben két fő lehetőségünk van adataink kezelésére:

- Használhatjuk a *verziókezelést*, mellyel az állomány aktuális változatán kívül a megelőző verzióit is tároljuk az ezen állományok ki- és bejelentkeztetésével.

Bejelentkeztetésnél az Oracle *i*FS automatikusan létrehozza az állomány előzményeit *(history),* mellyel annak bármely korábbi változatát elérhetjük. Az előzményeknél lehetőségünk van a változatokat megjegyzésekkel ellátni, melyekkel – magunk vagy mások számára – feltüntethetjük például a felmerült problémákat és annak megoldásait.

- Használhatjuk a *zárolást,* mellyel megakadályozhatjuk, hogy egy másik felhasználó megváltoztassa munkánkat. Zárolhatunk nem verziókezelt állományokat, valamint mappákat is. Zárolás esetén nem készül el az előzmények listája, mivel éppen a változtatást tiltjuk meg.

A zárolás akkor hasznos, amikor mi vagyunk egy mappa egyedüli tartalomkezelői, és a mappa készen áll a közzétételre. A könyvtár zárolásával megtiltjuk a benne lévő állományok további változtathatóságát.

A verziókezelést akkor érdemes alkalmazni, amikor állományaink egyedüli szerzőjeként a jövőben is felhasználható adatokat szeretnénk megtartani. Ám igazán akkor hasznos, amikor ugyanazon állományokon másokkal együttműködve dolgozunk.

Verziókezelés

A verziókezelés segítségével, amikor egy állományt átdolgozunk, megtarthatjuk a korábbi változatot is. A verziókezelt állományokat ki-, illetve be kell jelentkeztetni az adatbázisba. Az állomány kijelentkeztetése azt jelenti, hogy ezután senki sem tudja megváltoztatni, míg a bejelentkeztetés feloldja ezt a tilalmat. A verziókezelt állományok minden egyes bejelentkeztetésénél az Oracle *i*FS automatikusan eltárolja annak megelőző változatait is, és létrehozza az előzmények listáját. Például ha egy dokumentumot megváltoztatunk, és később úgy döntünk, hogy az előző változat jobb volt, elég visszatöltenünk egy korábbi verziót, nem kell emlékezetből újra megszerkesztenünk.

Az előzmények listáján látható korábbi változatok csupán olvashatók, de nem szerkeszthetők. A megfelelő jogosultságokkal rendelkező felhasználók letölthetik a korábbi változatokat, helyi háttértárukon szerkeszthetik azokat, majd feltöltve az adatbázisba, az lesz a jelenlegi verzió.

Az állományokat egyetlen parancs kiadásával verziókezeltté tehetjük. Ekkor automatikusan további attribútumokat kapnak, melyek az állomány *családjáról (family)* és a korábbi változatokról tárolnak információkat.

Hogy megvédjük az állományt a felülírástól, minden esetben, mikor elmentjük a verziókezelt állomány új változatát, a korábbi változatok is megmaradnak. Az állomány kijelentkeztetése után nem tárolja a korábbi verziókat addig, amíg újra be nem jelentkeztetjük.

A ki- és bejelentkeztetés

A ki- és bejelentkeztetés eszközével több felhasználó együttesen dolgozhat ugyanazokon az állományokon, ugyanakkor megvédhetjük a felülírástól azt a dokumentumot, amelyen éppen dolgozunk.

A ki- és bejelentkeztetés használatához az állománynak verziókezeltnek kell lennie.

A ki- és bejelentkeztetés hasonló a zárolás műveletéhez, azzal a különbséggel, hogy bejelentkeztetéskor az állományról új verzió készül. Ennek alapján az állományról elkészül az előzmények listája. A zárolt állományok hasonló módon védettek, de a zárolás feloldásáig nem változtathatók és nem készül el az előzmények listája. A zárolt állományon elvégzett bármilyen mentés az aktuális (és egyetlen) változatot írja felül.

A zárolás

Amennyiben az állományunk nem verziókezelt, a felülírástól az Oracle *i*FS zároló eszközével védhetjük meg. Mind állományok, mind mappák zárolhatók. A zárolás akkor hasznos, amikor szeretnénk megőrizni egy állományt, de nem akarjuk, hogy megváltoztassák, és emiatt nincs is szükség az előzmények listájára.

Egy állomány vagy mappa zárolása megvédi azt a törléstől, a mozgatástól és a felülírástól addig, amíg a zárolást elvégző személy fel nem oldja azt. A zárolás időtartama alatt senki, még a záró felhasználó sem képes véletlenül megváltoztatni vagy törölni a zárolt állományt. A záró felhasználó meg tudja nyitni és szerkeszteni az állományt, de a változtatásokat nem tudja elmenteni.

A zárolás felülbírálja a felhasználók írási és törlési jogosultságait (kivéve a zárolást elvégző felhasználóét). Ezek a személyek továbbra is meg tudják nyitni, vagy le tudják tölteni a zárolt állományt, amennyiben rendelkeznek a megfelelő jogosultságokkal. A zárolást egyedül a záró felhasználó tudja feloldani.

Nem engedélyezett azon könyvtárak törlése, amelyekben van legalább egy zárolt állomány.

5.3.3. Információ keresése az Oracle *i*FS adatszótárában

Az Oracle *i*FS `Find` (`Keresés`) eszközével hatékonyabban és pontosabb keresési feltételekkel kereshetünk az állományokon és a szövegeken, mint a Windows keresőjével. Sőt, mivel az Oracle adatbázisok hatékony *inter*Media Text opciója is be van építve az Oracle *i*FS-be, a webes felületen lehetőségünk nyílik az állományok tartalma alapján is keresni, nem csupán az állományok attribútumai alapján.

A nagy mennyiségű adatokon végzett hatékony keresés érdekében az Oracle *i*FS lehetőséget biztosít kulcsszó, dátum és egyéb feltételek alapján történő keresésre, sőt, ezeket a feltételeket kombináltan is alkalmazhatjuk.

A keresést az állományok attribútumai alapján is végezhetjük, például az állomány neve, mérete, nyelve vagy a létrehozás dátuma alapján. Webes interfészen keresztül ezeken kívül kereshetünk az állományok tartalmára is. Ha elektronikus leveleinket az Oracle *i*FS-ben tároljuk, a kereső megvizsgálja magát az üzenetet és a levél csatolt állományait is. Kereshetünk az állományok meghatározott elemei, például fejléce vagy tag-ek alapján is. Kereshetünk az állományok témájára is. A téma az állomány tartalmát írja le akkor is, ha maga a dokumentum nem tartalmazza a keresésben megadott kifejezéseket.

A kereséshez beírt kifejezések többségét a rendszer tárolja, így egy munkamenet alatt újra felhasználhatjuk őket anélkül, hogy újra be kellene gépelni.

A keresés eredményeképpen megjelenik azon állományok listája, melyek egyrészt megfelelnek a keresési feltételeknek, másrészt pedig van hozzájuk jogosultságunk. A korábbi változatait megtartó állományok esetén csak a legfrissebb verzió jelenik meg, de lehetőségünk van beállítani, hogy az összes verziót megtalálja.

Az Oracle *i*FS-ben a következő típusú keresések állnak rendelkezésünkre:

- Az állományok attribútumai alapján történő keresésnél az állományokat leíró metaadatokat vizsgáljuk, például az állományok nevét, a szerzőjét vagy a létrehozás dátumát.
- Az állományok tartalma alapján történő keresésnél az állományok törzsében szereplő szavakat és kifejezéseket vizsgáljuk. Csak akkor áll módunkban, ha az Oracle *i*FS szerverre telepítve van az Oracle *inter*Media Text.
- Az állományok témaköre alapján történő keresésnél olyan kifejezések után kutatunk, melyek nem feltétlenül jelennek meg az állományok címében vagy törzsében. Csak akkor érhető el, ha telepítve van az Oracle *inter*Media Text.

Az attribútumok alapján történő keresés

Az állományokon a következő attribútumok vagy metaadatok alapján kereshetünk:

- az állományt létrehozó felhasználó neve;
- az állomány leírása;
- a létrehozás dátuma;
- az utolsó módosítás dátuma;
- az állomány lejárati dátuma;
- az állomány mérete;
- az állomány státusa (ki- vagy bejelentkeztetett, zárolt vagy hozzáférhető);
- a tartalmának kódolásához használt karakterkészlet.

Az előre definiált attribútumokon kívül a keresést a rendszeradminisztrátor által definiált felhasználói attribútumokon is elvégezhetjük.

A tartalom alapján történő keresés az Oracle *inter*Media Text segítségével

Amennyiben az Oracle *inter*Media Text telepítve van az *i*FS szerverre, hatékony kereséseket végezhetünk nemcsak a metaadatok, hanem a dokumentumok törzsében szereplő elemek alapján is. Például megkereshetjük azokat az állományokat, melyekben szerepel a *piacelemzés* szó.

Mind a Windowsos, mind a webes kezelőfelület képes a tartalom alapján keresni, ehhez mindössze be kell gépelni a keresett kulcsszavakat és megjelölni, hogy tartalom alapján szeretnénk keresni.

A keresett fogalom lehet egy vagy több szó, kifejezés, és ezek tetszőleges kombinációja. A kifejezéseket aposztrófok közé kell tenni, ellenkező esetben a keresés eredményében szerepelnek olyan állományok is, ahol a kifejezésben szereplő szavak úgy fordulnak elő, hogy nem állnak egymással kapcsolatban.

Természetesen a tartalom alapján történő keresést más típusú keresésekkel is lehet kombinálni.

A téma alapján történő keresés az Oracle *inter*Media Text segítségével

Az állomány témája egyszerűen az, amiről az állomány szól. Amikor keresési feltételként egy témát adunk meg, akkor olyan kifejezések után kutat, amelyek nem feltétlenül szerepelnek a dokumentumok törzsében. Téma alapján történő keresésre csak a webes interfész képes, és az ABOUT operátort használja, melyet logikai operátorokkal lehet pontosítani.

Az Oracle *i*FS a téma alapján történő keresést olyan szabályok segítségével hajtja végre, melyek lehetővé teszik azon dokumentumok megtalálását, melyek kapcsolódhatnak a keresett témához.

Haladó keresés az Oracle *i*FS-ben

A webes felületen pontosabb keresési feltételeket fogalmazhatunk meg logikai operátorok segítségével. Az interfész `Advanced find` pontja alatt a következő kereséseket indíthatjuk el:

- Téma alapján történő keresések, ahol a megadott fogalmakat nem feltétlenül tartalmazza a dokumentum. Ez a keresés az ABOUT operátort alkalmazza, mely után szavak, illetve aposztrófok között kifejezések is állhatnak.
- A dokumentumok speciális elemei (például fejléc, tag-ek, mondatok, paragrafusok) alapján történő keresések. Ez a keresés a WITHIN operátort használja.

Általános keresési operátorok

'A' feltétel ÉS 'B' feltétel, ahol a ritkábban előforduló feltételt tekinti fontosabbnak	AND
'A' feltétel VAGY 'B' feltétel	OR
'A' kifejezés egyenlő-e a 'B' kifejezéssel	EQUIV
'A' kifejezés, de 'B' kifejezés NEM	NOT
'A' kifejezés, de ahol 'B' előfordul, kisebb súllyal	MINUS
'A' kifejezés, ahol közel áll 'B' kifejezéshez	NEAR

Az operátorokat mindig nagybetűkkel kell írni.

5.4. A biztonság kérdése az Oracle *i*FS-ben

Az állományok és mappák hozzáférhetőségének szabályozása alapvető fontosságú az adatkezelésben. Az Oracle *i*FS átfogó és rugalmas biztonsági modellt biztosít a jogosulatlan hozzáférések megakadályozására. Mivel a biztonság a protokollszerverek helyett az adatszótár hatáskörébe tartozik, az *i*FS egységes hozzáférési mechanizmust alkalmaz, függetlenül a felhasználó személyétől, illetve attól, hogy milyen protokollt és eszközt használ. A Java API-t használó saját alkalmazásokat is ugyanazok a szabályok kötik, mint a két hagyományos felületet, a Windowsos és a webes interfészt.

Az Oracle *i*FS kétszintű biztonsági rendszert biztosít:

1. A felhasználói fiókok szintjén az Oracle *i*FS jelszót kér a felhasználó azonosítására.
2. A tartalom szintjén az Oracle *i*FS a felhasználó vagy csoport jogosultságait összeveti az állomány vagy mappa hozzáféréséhez megkövetelt jogosultságokkal.

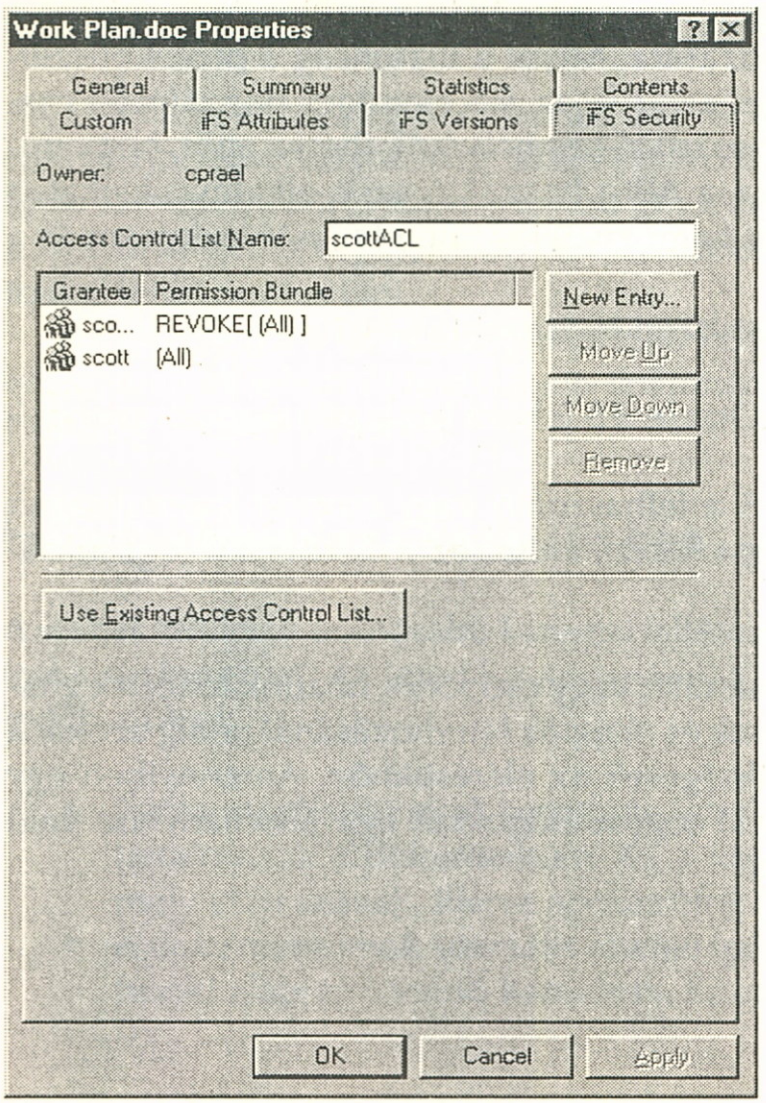
Minden esetben, amikor egy új felhasználót vagy csoportot hozunk létre, meghatározzuk annak jogosultságait. Állományok és mappák létrehozásánál pedig azokat a jogosultságokat definiáljuk, melyek szükségesek az elérésükhöz. Amennyiben egy felhasználó nem rendelkezik a megfelelő jogosultságokkal egy állományhoz való hozzáféréshez, nem tudja megnyitni azt.

5.4.1. Hozzáférési listák

Az Oracle *i*FS-ben egy felhasználó vagy csoport a hozzárendelt jogosultságokkal egy ACE-t (Access Control Entry – Elérést Szabályozó Bejegyzés) alkot. Több ACE egy ACL-t (Access Control List – Elérést Szabályozó Lista) alkot, amely megad vagy megtagad jogosultságokat a felhasználótól vagy csoporttól.

Egy ACL-lista (5.11. ábra) a következőket tartalmazza:

- Jogosult (grantee) – Meghatározott felhasználó vagy csoport. Csoport létrehozása akkor hasznos, amikor több felhasználónak ugyanazokat a jogosultságokat szeretnénk adni, így nem kell mindegyiknek külön beállítani, és a módosítása is egyszerűbb.
- Jogosultságok (set of permissions) – A jogosultságok a rendszeradminisztrátor által definiált műveletekhez kötődnek, például mappák elemeinek törlése, módosítása és mozgatása, vagy a tartalom olvasása. Mindegyik műveletet lehet engedélyezni vagy letiltani. A jogosultságok ugyanazon ACL-en belül az egyes ACE-knek különbözőek lehetnek.



5.11. ábra. *Az ACL-lista*

A létrehozott mappák és állományok tulajdonosaiként szabályozhatjuk, kinek van hozzáférése, valamint megadhatjuk a hozzáférés szintjét. Előre definiált ACL-ek állnak rendelkezésünkre, de saját, testreszabott ACL-eket is alkalmazhatunk. Az ACL-ek hozzárendelése két szinten történik: állományok és mappák szintjén. Mindkettő esetén szabályozhatjuk az elvégezhető műveletek mennyiségét, továbbá a mappák szintjén könyvtárműveleteket is ellenőrizhetünk.

Felhasználóként rendelkezünk egy, a rendszeradminisztrátor által definiált ACL-listával. Amikor létrehozunk egy állományt vagy mappát, ezt az ACL-t fogja kapni az is. Alapesetben a hozzáférés az új elemhez privát, ami azt jelenti, hogy csak mi érhetjük el. Ezen ACL megváltoztatásához adminisztratív jogosultsággal kell rendelkeznünk.

5.4.2. Az ACL-listák típusai

Az ACL-listáknak két típusa van:

- *Előre definiált* ACL-ek, melyek alapvető biztonsági szolgáltatásokat nyújtanak, és az Oracle *i*FS-sel szállítják őket.
- *Felhasználó által definiált* ACL-ek, melyeket a felhasználók hoznak létre a beépített ACL-ek által nem biztosított speciális körülmények megoldására.

A beépített ACL-ek biztonsági szintjeit az 5.3. táblázat tartalmazza.

5.3. táblázat. *A beépített ACL-ek biztonsági szintjei*

ACL	*Leírása*	*Megfelelő jogosultságok*
Privát (private)	Az általunk létrehozott állományok és mappák alapértelmezett ACL-je. Kizárólag a tulajdonosnak és az adminisztrátoroknak enged hozzáférést.	A tulajdonos és az adminisztrátorok az összes jogosultsággal rendelkeznek, míg a többiek egyetlen jogosultsággal sem.
Védett (protected)	Lehetővé teszi a felhasználóknak, hogy lássák az állományokat és mappákat, mozgathatják az általuk létrehozott elemeket, de nem törölhetik a többi elemet és nem változtathatják meg a tulajdonságait.	Védett
Nyilvános (public)	Teljes elérést biztosít. Bármelyik változtatás végrehajtható.	Minden jogosultság (olvasás, módosítás, törlés)
Publikált (published)	Láthatóvá teszi a mappákat és az állományokat, valamint azok tartalmát.	Olvasás

5.4.3. A jogosultságok

A jogosultságok olyan műveletek, melyeket egy felhasználó vagy csoport megtehet egy állománnyal vagy mappával. Minden egyes felhasználóhoz és csoporthoz meg kell adnunk, hogy a következő öt művelet közül melyek engedélyezettek:

- `All` – Mindegyik jogosultságot megadjuk vagy megtagadjuk.
- `Delete` – Elemek törlése.
- `Modify` – Állomány vagy mappa tartalmának módosítása.
- `Protected` – Mappák változtatás elleni védelme. Az ezen jogosultsággal rendelkező felhasználók másolhatnak állományokat a mappába vagy törölhetnek onnan, de magát a könyvtárat nem nevezhetik át és nem törölhetik. A védett jogosultságot használják az alapértelmezett védett ACL definiálására.
- `Read` – Állomány vagy mappa tartalmának olvasása.

Egy felhasználó vagy csoport számára mindegyik jogosultság beállításával (megadásával vagy megtagadásával) az ACL-listán belül egy ACE-t hozunk létre. Lehetőségünk van arra, hogy minden általunk kezelt állományra mindegyik *i*FS felhasználónak különböző ACE-t definiáljunk, de legtöbbször az Oracle *i*FS beépített ACL-listái megfelelők, a maradék esetekben pedig létrehozhatunk testreszabott, rövid ACL-eket.

ACE definiálásakor először azt kell eldöntenünk, hogy az ACE-ben többségében megadjuk-e a jogosultságokat, vagy inkább megtagadjuk őket. Ha egy felhasználónak vagy csoportnak csak kevés jogosultságot szeretnénk megadni, válasszuk az ACE típusának a `Grant`-et *(Megad)*, és válasszuk ki a megadandó jogokat. Ha viszont olyan ACE-t szeretnénk létrehozni, mely kevés megszorítással általános hozzáférést biztosít, legyen az ACE típusa `Revoke` *(Megtagad)*, így csak a néhány megszorító jogot kell bejelölnünk. Például, ha egy könyvtárhoz a teljes elérést megadjuk a törlési jog kivételével, akkor érdemes `Revoke` típusú ACE-t létrehozni, és csak a `Delete` jogosultságot kiválasztani.

Az ACE-bejegyzések az ACL-listában meghatározott sorrendben találhatóak – általában a létrehozás sorrendjében. A listán lentebb szereplő ACE-k felülbírálják a felette elhelyezkedő ACE-ket, ebben az esetben a később definiáltak erősebbek a korábbiaknál. Figyelem, a megfelelő biztonság eléréséhez az ACE-bejegyzések sorrendje nagyon fontos!

5.5. XML-adatállományok használata az Oracle *i*FS-ben

Az Oracle *i*FS támogatja XML-dokumentumok tárolását – elemző és leképező eszközeivel a felhasználók XML-állományaikat strukturált vagy strukturálatlan adatokként tárolhatják az adatbázisban. Az Oracle az első olyan adatbázis-kezelő, amely XML-dokumentumok tárolásához alapvető eszközrendszerrel rendelkezik, képes automatikusan feltölteni, generálni és transzformálni XML-állományokat magában az adatbázis motorjában.

Az Oracle *i*FS kibővíti az adatbázisok alapvető XML funkcionalitását, és egy olyan felhasználóbarát felületet biztosít, mely kinézetében hasonlít egy szabványos állománykezelő rendszerhez, így a felhasználók könnyedén tudnak létrehozni, szerkeszteni és megtekinteni az adatbázisban tárolt XML-állományokat.

5.5.1. Az XML-dokumentumok tárolásának módjai

Az alábbiakban öt alapvető módszert ismertetünk arra, hogyan használhatunk fel XML-állományokat az Oracle *i*FS-ben. Ezekben a meghatározásokban a perzisztens XML-dokumentumok valóban XML-dokumentumokként jelennek meg a könyvtárhierarchiában, míg a tranziens XML-dokumentumokat a rendszer csak egy-egy tranzakció végrehajtásához tölti fel: ennek eredményeképpen más típusú, perzisztens dokumentumok jöhetnek létre, vagy megváltoztathatják az Oracle *i*FS konfigurációját, de magát az XML-dokumentumot az *i*FS nem tárolja dokumentumként.

Perzisztens XML-dokumentumok tárolása

Az Oracle *i*FS-ben az XML-dokumentumokat ugyanúgy tárolhatjuk, mint bármely más állománytípust. Ezeket tetszőleges protokollon keresztül feltölthetjük, tartalmukat tetszőleges szerkesztővel módosíthatjuk, illetve tetszőleges XML-alkalmazásban használhatjuk fel őket.

Az XML-dokumentumok tárolásához semmilyen testreszabás nem szükséges, hacsak a gyökértag nem egy olyan elem, mely megegyezik egy Oracle *i*FS objektum foglalt gyökérelemével. Ha mégis, akkor az ütközések elkerüléséhez XML-névteret kell használnunk. A foglalt gyökérelemek a következők: az Oracle *i*FS osztályhierarchiájában szereplő bármely osztály neve (például `Document`, `Folder`, `PublicObject`), két speciális osztály, a `SimpleUser` és az `ObjectList`, illetve az általunk létrehozott alosztályok nevei.

Tárolhatunk literális XML-dokumentumokat is (abban az esetben is, amikor a gyökértag megegyezik egy Oracle *i*FS elem nevével), ekkor az elemzést le is tilthatjuk. Ez utóbbi a CUP és az FTP protokollokban és a webes felületen is lehetséges.

Példa literális XML-dokumentumra:

```
<?xml version="1.0" standalone="yes"?>
<NemFelismertTag>
<Nev>Felhasználó Károly</Nev>
<Leiras>Ő egy felhasználó.</Leiras>
</NemFelismertTag>
```

A `literal.xml` dokumentumot a fenti tartalommal, elemzés nélkül, XML-dokumentumként tároljuk. A `Name` és a `Description` név sok Oracle *i*FS objektumban szerepel, de mivel a gyökértag (`NemFelismertTag`) nem egyezik az előzőkkel, az XML-dokumentumot a rendszer nem elemzi.

DTD érvényességellenőrzés

Az Oracle *i*FS elemzője képes DTD érvényességellenőrzésre. A DTD-t tárolhatjuk az XML-dokumentumon belül vagy egy külső állományban is.

Az Oracle DOM elemzi az XML-állományokat, összeveti tartalmukat a DTD érvényességi szabályai által előírtakkal, és meghatározza, hogy az állomány érvényes-e vagy sem. Alapesetben az elemző nem ellenőrzi az XML-állományokat. Mivel az érvényesség ellenőrzésének a beállítása az egész rendszerre vonatkozik, a DTD ellenőrzés be-, illetve kikapcsolása egy rendszertulajdonság beállításával lehetséges az Oracle9*i*FS Managerben.

Az elemzés alatt az Oracle *i*FS ellenőrzi a `Validate` paramétert (melyet a `Parse()` metódus kapott meg). Amennyiben nem kapott ilyet, úgy megvizsgálja az ennek megfelelő rendszertulajdonságot és ennek alapján beállítja a paramétert. Mindez az előellenőrzés alatt zajlik le. Később, az elemzés során az `IfsSimpleXmlParser` figyelmen kívül hagyja a `Validate` paraméter értékét, de a `LiteralDocumentParser` ennek alapján dönti el, hogy kell-e ellenőrizni az állományt. Saját elemző is használhatja a `Validate` paramétert: kiolvashatja értékét és átadhatja azt az `IfsXmlParser.createDOM()` metódusnak, vagy bármi más módon felhasználhatja azt.

Amennyiben az XML-elemző egy állományt érvénytelennek talált, az Oracle *i*FS kivételt dob, amelyet az elemző visszagörgető mechanizmusában, vagy bármely más Oracle *i*FS alkalmazásban elkaphatunk. Ezenkívül a feltöltésre használt szerver megjelenít egy hibaüzenetet.

Ha az állomány érvénytelen, az Oracle *i*FS nem tárolja, tehát a felhasználónak ki kell javítania a hibá(ka)t és a feltöltést újra meg kell próbálnia.

Elemzett attribútumokkal rendelkező perzisztens XML-dokumentumok tárolása

Lehetőségünk van a perzisztens XML-dokumentumban levő adatelemek elemzésére, és ezek közül néhánynak vagy akár az összesnek a dokumentum attribútumaiként való tárolására. Ezek az elemek a dokumentum `Property` tulajdonságlapján jelennek meg (mind a Windowsos, mind a webes felületen), és ezekre saját keresést is kiadhatunk. Számos lehetőség áll rendelkezésünkre arra, miként kezelje az Oracle *i*FS ezeket az attribútumokat.

Az Oracle *i*FS ugyan egy állományrendszer, a lelke mégis az Oracle9*i* adatbázis. Az Oracle *i*FS egyik egyedi képessége, hogy állományainkat mind állományokként, mind relációs adatokként képes tárolni. Lehetőségünk van egy olyan, saját elemző készítésére, mely az XML-dokumentumokból kiemeli a kiválasztott adatokat anélkül, hogy a dokumentumot magát módosítanunk kellene. Továbbra is képesek vagyunk ezen dokumentumok megtekintésére és szerkesztésére, de kiadhatunk olyan kereséseket is, melyek ezekre az elemzett adatelemekre irányulnak.

Ezen lehetőség kiaknázásához az XML-dokumentumunkat a `Document` tag, vagy a `Document` valamely leszármazott alosztályát reprezentáló tag felhasználásával kell létrehoznunk. A felismert tag-ek az állományok attribútumaiként tárolódnak, a nem felismert tag-ekben levő adatok pedig figyelmen kívül lesznek hagyva.

XML-dokumentum elemzett adatelemekkel:

```
<?xml version="1.0" standalone="yes"?>
<Document>
<Nev>Felhasználó Károly</Nev>
<Leiras>Ő egy felhasználó.</Leiras>
</Document>
```

Amikor a következő XML-dokumentumot beszúrjuk az Oracle *i*FS-be, adatelemei az állomány attribútumaiként kerülnek elemzésre (például a fenti kód az XML-állomány attribútuma lesz). Ezekre az attribútumokra kereshetünk, és értelmezhetjük a dokumentumot XML-ként. Hogy a dokumentumnak szöveges tartalma legyen (például mint egy HTML-állománynak), a `Content` tag-et használhatjuk:

```
<?xml version="1.0" standalone="yes"?>
<Document>
<Nev>Felhasználó Károly</Nev>
<Leiras>Ő egy felhasználó.</Leiras>
<Content>Íme a dokumentum tényleges tartalma!</Content>
</Document>
```

Amikor ezt a dokumentumot beszúrjuk az Oracle *i*FS-be, adatelemei az állomány attribútumaiként vannak elemezve, ezenkívül a `Content` elemek által határolt adatok a dokumentum tartalma lesznek. Feltöltés után, megnyitva ezt az állományt a webes felületen, az „Íme a dokumentum tényleges tartalma!" olvasható.

Néhány alkalmazás esetén előfordulhat, hogy XML-állományok helyett saját dokumentumtípust szeretnénk definiálni. Ekkor lehetőségünk van a dokumentum alosztályt olyan XML-állomány használatával azonosítani, amelyben XML-tag-ek segítségével definiáljuk az általunk választott dokumentumtípus attribútumait. Amikor feltöltjük az XML-állományt az Oracle *i*FS-be, a dokumentum elemzésre kerül és létrejön a választott típusú dokumentum. A létrehozáshoz felhasznált XML-dokumentum nem jelenik meg az Oracle *i*FS-ben.

Adatátvitelre használhatunk XML-állományokat is. Több olyan lehetőségünk is van, melyek csak kevés, illetve semmilyen testreszabást nem igényelnek.

Roundtrip XML-dokumentumok tárolása

A Roundtrip XML az Oracle9*i*FS egy új eszköze, mellyel XML-állományokat perzisztens dokumentumokként tárolhatunk, majd később ugyanúgy képezhetjük le őket, ahogy beérkeztek. Ez azon felhasználók számára előnyös, akik információk tárolására megjegyzés elemeket használnak, vagy definiálatlan elemeket olyan információk tárolására, melyeket az Oracle *i*FS-nek nem kell elemeznie.

Alapesetben, mikor egy `Document` osztállyal vagy annak egy alosztályával rendelkező dokumentumot feltöltünk, maga a dokumentum az állomány `Content` attribútumában lesz tárolva. A dokumentum leképezésénél a kimenet az eredeti XML-dokumentum – ahogy az megérkezett, tartalmazva a megjegyzést és az Oracle *i*FS-ben definiálatlan elemeket.

Roundtrip XML használatakor két fontos megállapodást kell figyelembe vennünk:

1. A dokumentum az Oracle *i*FS-ből pontosan ugyanúgy lesz leképezve, ahogyan az be lett szúrva. Az Oracle *i*FS-ben történő változtatások a dokumentumban nem lesznek hatással a kimenetre, hacsak nem írunk egy olyan interfészt, mely szinkronban tartja a dokumentum attribútumait a `Content` attribútumban tárolt információkkal.
2. A teljes dokumentum a szabványos Oracle9*i*FS `Content` attribútumában lesz tárolva. Ez azt jelenti, hogy egy roundtrip dokumentum nem definiálhatja saját `Content` elemét. Ha ezt mégis megtesszük, a `Content` tag-ben lévő szöveg a dokumentum tartalmaként lesz tárolva.

5.5.2. XML-dokumentumok használata az Oracle *i*FS-ben

Dinamikus tartalom megvalósítása

Az e-business alkalmazások készítőinek és felhasználóinak alapvető fontosságú eszköz a dinamikus tartalomgenerálás lehetősége. Akár a legfrissebb híreket közzétevő honlapot írunk, akár egy használati útmutatót szeretnénk több nyelvre lefordítva pub-

likálni az interneten, a dinamikus tartalom megvalósítása az egyik legfontosabb követelmény az alkalmazásfejlesztők számára.

Az Oracle *i*FS leképező eszközei lehetőséget biztosítanak a dinamikus tartalomgenerálásra. A leképezőn keresztül ugyanazt a dokumentumot tetszőleges módon képezhetjük le, és a kimeneti dokumentumot a szükséges formában állíthatjuk össze. A leképezést az állomány elemzésével fel lehet gyorsítani. Az XML segítségével történő dinamikus tartalomgenerálás során például az alábbi lehetőségeink vannak:

- Bizonyos részleteket elrejthetünk a felhasználók elől azzal, hogy egyszerűen kihagyjuk azokat a leképezés során.
- Egy XML-PDF leképező beillesztésével XML-dokumentumokat PDF formátumúvá tudunk konvertálni.
- Létre tudunk hozni olyan honlapot, mely megjeleníti a legfrissebb híreket. Ehhez mindössze be kell illeszteni egy, a híreket tartalmazó XML-dokumentumot, és a megfelelő attribútumot rá kell irányítani. A honlap megtekintésekor automatikusan megkeresi a legfrissebb hírt, és beszúrja a honlapba.

Üzleti logika

XML-dokumentumok tárolására az Oracle *i*FS használatának a szabványos állománykezelő rendszerekkel szembeni egyik előnye, hogy az XML alapú tartalmon üzleti logikát alkalmazhatunk. A legtöbb szervezet mára megalkotta adattárházas rendszerét, hogy abban elemezze és értékelje forgalmi adatait, melyeket relációs adatbázisokban tárolnak. Azonban számos információ található még mindig a régi állománykezelőkben tárolva. Hogyan tudunk üzleti logikát alkalmazni ezekre a többnyire XML-dokumentumokban tárolt információkra?

Ennek megoldására jelenleg a leggyakoribb módszer Perl-szkriptek írása az XML-dokumentumok feldolgozására. Ez azonban igen lassú futást eredményez, ami megengedhetetlen a vállalat döntéshozói számára. Jobban járunk, ha ezekre az XML-állományokra saját elemzőket hozunk létre, majd a dokumentumok komponenseit eltároljuk egy relációs adatbázisban. Ezután már használhatjuk megszokott relációs adatbányászati eszközeinket. Ezek az eszközök az időbeli változások elemzésére is képesek.

Saját típusú dokumentumpéldányok létrehozása XML-lel

Saját típusú (felhasználó által definiált alosztályba tartozó) példányok létrehozása ugyanúgy történik, ahogyan külső objektumokat példányosítunk XML-lel.

A következőkben bemutatjuk, hogyan kell egy olyan XML-dokumentumot létrehozni, amely saját típusú dokumentumot példányosít:

1. Gyökérelemként hozzunk létre egy saját típusú XML-állományt.

2. Illeszünk be bejegyzéseket (belépési pontokat) a kötelező, valamint a kívánt mennyiségű opcionális attribútumok számára.
3. Töltsük fel az állományt az Oracle *i*FS-be, ezzel példányosítunk. A lehetséges, illetve a kötelező attribútumok száma az állomány osztályától, valamint annak ősosztályaitól függ (beleértve azok lehetséges értékeit is).

Amennyiben erre XML-állományt használunk fel, az soha nem tárolódik az Oracle *i*FS-ben, csak a létrehozott dokumentumpéldány jelenik meg.

Oracle *i*FS objektumok használata attribútumokként

Állományok attribútumaként lehetőségünk van Oracle *i*FS objektumreferenciákat tárolni. Például, ha szeretnénk egy, az Oracle *i*FS-nek elküldött dokumentum szerkesztőjeként egy bizonyos kezelőt társítani, akkor használhatjuk a `DirectoryUser` objektumot attribútumként. Használhatunk egyetlen értéket, vagy tömbben több objektumot.

Egy Oracle9*i*FS objektum attribútumként való felhasználásához azonosítanunk kell a referencia típusát az attribútum elem paramétereként. Az 5.4. táblázatban bemutatjuk az attribútumok referenciatípusait.

5.4. táblázat. *Attribútumok referenciatípusai*

Referenciatípus (`RefType`)	*Használata*	*Példa*

`ID`	A hivatkozott objektum azonosítóját specifikálja. Az `IfsSimpleXmlParser` ezen azonosító alapján keresi az objektumot az adatszótárban.	`<CLASSOBJECT>`
		`<Name>TypeDef</Name>`
		`<Superclass RefType='ID'> 263`
		`</Superclass>`
		`<Description>`
		`XML tesztosztaly`
		`</Description>`
		`</CLASSOBJECT>`

`Path`	Beágyazott `PublicObject` típusú objektumra elérési útjával lehet hivatkozni. Ezen paraméter értéke egy elérési útvonal (minősítés), mely lehet relatív vagy abszolút. Előbbi esetén az `IfsXmlParser`-t hívó alkalmazásnak be kell állítania a `Path` opciót az opciótáblázatban.	`<PublicObject>`
		`<Name> Tesztobjektum </Name>`
		`<Document RefType='Path'>`
		`/docs/test01.doc`
		`</Document>`
		`</PublicObject>`
		
		Mivel ez az útvonal / jellel kezdődik, ezért abszolút. Figyeljünk arra, hogy az elválasztó karakter egy munkamenet-függő konfigurációs paramétertől függ.

5.4. táblázat. *Attribútumok referenciatípusai (folytatás)*

Referenciatípus (RefType)	*Használata*	*Példa*
ValueDefault	Beágyazott objektumokat egy ValueDefault (alapértelmezett érték) típusú objektumon keresztül lehet elérni, mely az objektumra hivatkozik.	<PropertyBundle> <Update RefType='valuedefault'> ParserLookupByFileExtension </Update> <Properties> <Property action ='add'> <Name>xls</Name> <Value DataType='String'> MyApplication.Parsers.XLSParser </Value> </Property> </Properties> </PropertyBundle >
Search	Beágyazott objektumokra ezen típusú referenciával tudunk keresést kiadni. Annak az attribútumértéknek, mely a beágyazott objektum, rendelkeznie kell egy osztálytartománnyal (Class Domain), mely egyetlen típusra, vagy annak leszármazottainak valamelyikére szűkíti le, vagy egy másik attribútumnak (Classname) kell leírnia az objektum típusát. A RefType attribútum értékének azon attribútum nevének kell lennie, amelyre a keresést kiadjuk. Az elem értékének a keresett értéknek kell lennie. Amennyiben a keresés sikertelen, kivételt dob.	<PublicObject> <Name> Test Object </Name> <Owner RefType='DistinguishedName'> jdoe@us.oracle.com </Owner> </PublicObject> Mivel az Owner attribútumot az osztálytartománya leszűkíti a DirectoryUser típusra, ezért a keresés nem állítja be a Classname attribútumot.

Az XmlParserLookup tulajdonságcsoport

Az Oracle *i*FS-ben a tulajdonságcsoport összetartozó név–érték párok együttese. Az Oracle *i*FS képes a rendszer metaadatokat a tulajdonságlistában nevük alapján megkeresni, és így kiolvasni a tulajdonság értékét. Ezzel testreszabhatjuk a rendszert, de emellett meghagyjuk az Oracle *i*FS-nek, hogy előírja egyes tulajdonságok nevét és elhelyezkedését.

Az `XmlParserLookup` tulajdonságcsoportot arra használjuk, hogy meghatározza, mely elemző dolgozza fel a beérkező XML-dokumentumokat. Az 5.5. táblázatban az ebben a csoportban használatos négy szabványos értéket mutatjuk be.

5.5. táblázat. *Az* `XmlParserLookup` *tulajdonságcsoport szabványos értékei*

Tulajdonság neve	*Értéke*	*Leírás*
`IfsDefaultNamespace`	http://xmlns.oracle.com/ifs	Ez az alapértelmezett névtér, a külön névtérrel nem ellátott XML-állományok ezt használják. Megváltoztatásához csupán létre kell hoznunk egy névteret az XML-dokumentum számára, így saját elemző fogja feldolgozni az `IfsSimpleXmlParser` helyett.
	`Oracle.ifs.beans.parsers.IfsSimpleXmlParser`	Ez az Oracle *i*FS alapértelmezett XML-elemzőjének a teljes csomagneve.
`LiteralDocumentParser`	`Oracle.ifs.beans.parsers.LiteralDocumentParser`	Ez az elemző dolgozza fel azokat az XML-dokumentumokat, melyeknek a gyökéreleme nem az Oracle *i*FS osztályhierarchiájának a része, és sem a `Document` (vagy más származtatható osztály), sem a `SimpleUser`, sem az `ObjectList` osztálynak nem alosztálya.
`IfsDefaultDTDValidation`	`False`	Logikai érték, mely meghatározza, hogy az XML-állományt kell-e érvényesíteni DTD-vel.
`IfsRoundTripEnabled`	`True`	Logikai érték, mely meghatározza, hogy az XML-állomány tartalma az állománynak megfelelő objektum tartalmaként legyen-e tárolva.

Amennyiben az `IfsSimpleXmlParser` helyett saját elemzőnket szeretnénk használni, a tulajdonságcsoportban az elemzőt az alapértelmezett Oracle *i*FS névtérhez társító tulajdonságot megváltoztatva beállíthatjuk saját elemzőnk teljes csomagnevét. Ha többnyire az `IfsSimpleXmlParser`-t szeretnénk használni, de bizonyos névterekhez speciális elemzőt szeretnénk hozzárendelni, új névteret kell létrehoz-

nunk. Saját viselkedés implementálásához a legjobb módszer, ha az Oracle *i*FS névteret változatlanul hagyjuk, definiálunk magunknak egy új névteret, majd az `IfsDefaultNamespace` tulajdonságot beállítjuk úgy, hogy az a saját névterünkre mutasson.

XML-adatállományok leképezése

Az XML-állományokat nemcsak az Oracle *i*FS-be történő információbevitelre használják, hanem természetesen az onnan való olvasásra is. Az Oracle *i*FS-ben bármelyik objektum leképezhető XML-állománnyá, függetlenül a létrehozásának módjától.

Az Oracle *i*FS-ben az általános leképező *(renderer)* az elemző ellentéte, vagyis az elemzett XML-dokumentumból újra megszerkeszt egy XML-állományt. Így amikor egy felhasználó lekéri ezt az állományt (például szeretné megtekinteni egy böngészőben vagy rákattint a Windows Explorerben), akkor az Oracle *i*FS az elemzett komponensekből felépíti a dokumentumot.

A fejlesztőknek lehetőségük van saját leképező létrehozására. Például, ugyanazt a dokumentumot többféle formában is meg lehet jeleníteni, vagy megszűrhetjük a dokumentum tartalmát.

Az Oracle *i*FS-ben tárolt állományok számos metaadattal rendelkeznek. Ezek az Oracle *i*FS környezetén kívül értéktelenek (mint például a jogosultsági lista), így leképezéskor beállíthatjuk, mely attribútumok, milyen szintű részletességgel kerüljenek be az XML-dokumentumba.

Osztályobjektumok leképezési lehetőségei

Az Oracle *i*FS-ben minden osztályobjektum (`ClassObject`) rendelkezik egy társított tulajdonságcsoport (`PropertyBundle`) objektummal. Ebben a következő három tulajdonság határozza meg, hogy leképezéskor mely attribútumokat dolgozza fel:

- `IFS.RENDERER.RenderedAttributes`
- `IFS.RENDERER.InheritSuperClassPolicy`
- `IFS.RENDERER.InlineRendering`

Nézzük sorra ezeket.

`IFS.RENDERER.RenderedAttributes`

Ez az attribútum sztringek listája, melyek a leképezendő attribútumokat tartalmazzák. Amennyiben ez a tulajdonság nincs meghatározva, az osztály bővített attribútumai feldolgozásra kerülnek, és az örökölt attribútumok további leképezését az `IFS.RENDERER.InheritSuperClassPolicy` szabályozza.

A három alaposztály (`PublicObject`, `SchemaObject` és `SystemObject`) külön definiált attribútumlistával rendelkezik. A lista `PublicObject` esetén `Name` és `Description` attribútumokat, `SchemaObject`-nél csupán a `Name` attribútumot tartalmazza, míg

SystemObject és Document esetében az értékek null-ra vannak állítva. Ezzel lehetővé vált a leghasznosabb információk leképezése (például a Document esetén a bővített attribútumok teljesen értéktelenek az Oracle *i*FS kontextusán kívül). Lehetőségünk van megváltoztatni ezeket az értékeket, de a legtöbb felhasználó számára megfelelőek ezek az információk, így nem kell átrágni magunkat egy sor rendszerfüggő tulajdonságon.

IFS.RENDERER.InheritSuperClassPolicy
Logikai érték; amennyiben értéke igaz, a szülő osztály attribútumai annak IFS.RENDERER beállításaitól függően leképezésre kerülnek. Ha a szülőosztály IFS.RENDERER.InheritSuperClassPolicy értéke szintén igaz, akkor szülő osztályának az attribútumait is leképezi, és így tovább, fel a hierarchián.

Hamis beállítás esetén csupán az objektum attribútumai kerülnek feldolgozásra. Ha egy alosztályának IFS.RENDERER.InheritSuperClassPolicy értéke igazra van állítva, akkor az objektum attribútumait leképezi, de nem lép visszább a hierarchiában.

Alapértelmezett értéke az igaz.

IFS.RENDERER.InlineRendering
Ez a tulajdonság három értéket vehet fel: Name, ID és Deep (név, azonosító és mélység). Általa definiálható, hogy az objektum mely attribútumait képezze le, abban az esetben, amikor az objektum belsőleg, egy másik osztály attribútumaként kerül leképezésre.

Például, létrehozunk egy saját dokumentumtípust, melynek egyik attribútuma a Manager, azon személy neve, aki a publikálás előtt ellenőrzi a dokumentumot. A Manager típusa DirectoryUser. Az egyetlen információ, mely mindebből megjelenik a dokumentum adattáblájában, a Manager-ként megjelölt DirectoryUser azonosítója (ID). Ez az Oracle *i*FS környezetén kívül haszontalan információ. Az IFS.RENDERER.InlineRendering három típust biztosít a kimenetre olyan objektumok számára, melyeket más objektumok használnak attribútumként.

- A Name a beágyazott objektum Name attribútumát képezi le.
- Az ID a beágyazott objektum azonosítóját (ID attribútumát) képezi le.
- A Deep a „mélységi leképezés"-re (deep rendering) utal, amely azt jelenti, hogy a beágyazott objektum teljes attribútumkészletét leképezi, de figyelembe veszi a beágyazott objektum IFS.RENDERER.RenderedAttributes beállítását, és csak a meghatározott attribútumokat képezi le.

Amennyiben az IFS.RENDERER.InlineRendering nincs explicit módon beállítva, alapértelemzésként a Name attribútumot próbálja meg leképezni. Amennyiben a Name hiányzik, az azonosítót képezi le.

Speciális kezelést igénylő osztályok

Alapértelmezés szerint az osztályok minden adateleme leképezésre kerül. Az alábbi esetekben azonban mindegyik osztály rendelkezik egy logikai tulajdonsággal, amellyel bizonyos részeleteket elrejthetünk. Például, ha egy ACL-t leképezünk, akkor alapesetben az abban lévő mindegyik ACE is leképezésre kerül. Azonban ha csak az ACL nevét szeretnénk látni, akkor az `IFS.RENDERER.ACCESSCONTROLLIST.ShowAces` tulajdonságot hamisra kell állítanunk.

Ezekben az esetekben nincs öröklődés. Például ha származtatunk a `PropertyBundle` osztályból, és nem szeretnénk látni a tulajdonságokat, akkor a megfelelő tulajdonságot (`ShowProperties`) a gyermek osztályban is be kell állítanunk hamisra. Ezen tulajdonság alapértelmezés szerint igaz.

Az 5.6. táblázat a speciális kezelést igénylő osztályokat sorolja fel. Mindegyik tulajdonság logikai, alapértelmezés szerint igazra állítva.

5.6. táblázat. *Speciális kezelést igénylő osztályok*

Tulajdonság	*Leírás*
`IFS.RENDERER.ACCESSCONTROLLIST.ShowAces`	ACL-objektumok számára. Igaz esetén az összes benne lévő ACE leképezésre kerül.
`IFS.RENDERER.PUBLICOBJECT.ShowCategories`	Publikus objektumok számára. Igaz esetén az objektumhoz rendelt összes kategória leképezésre kerül.
`IFS.RENDERER.CLASSOBJECT.ShowAttributes`	Osztályobjektumok számára. Igaz esetén az osztályobjektum összes attribútuma leképezésre kerül.
`IFS.RENDERER.ACCESSCONTROLENTRY.ShowAccessLevels`	ACE-k számára. Igaz esetén mindegyik ACE elérési szintje leképezésre kerül.
`IFS.RENDERER.PROPERTYBUNDLE.ShowProperties`	Igaz esetén a tulajdonságcsoport összes taulajdonsága leképezésre kerül.

Speciális eset még az állományok tartalma. Belsőleg kerül leképezésre, amennyiben az `IFS.RENDERER.RenderedAttributes` tulajdonság attribútumlistájában szerepel a `CONTENTOBJECT` attribútum.

5.5.3. Az Oracle *i*FS konfigurálása XML-állományokkal

Az XML-állományokkal mindeddig a tartalom és az attribútumok szemszögéből foglalkoztunk. XML-lel adatelemeket azonosíthatunk, az `IfsSimpleXmlParser` pedig ezen értékek alapján képes létrehozni és közzétenni dokumentumokat. Az állományokat az Oracle9*i* adatbázisban tároljuk mint az Oracle9*i*FS séma megfelelő rekordjait.

Azonban az Oracle *i*FS-ben maga a séma önleíró. A séma elemeit sémaobjektumokként, táblák rekordjaiként tárolja. Ahogyan `PublicObject`-eket tudunk létrehozni felismert adatelemekkel rendelkező dokumentumok feltöltésével, úgy `SchemaObject`-eket is létre tudunk hozni felismert adatelemekkel rendelkező XML-dokumentumok feltöltésével.

A feltöltés folyamata hasonlatos ahhoz, amikor tranziens XML-állományokat töltünk fel saját perzisztens dokumentumok létrehozásához. Ebben az esetben egy olyan tranziens XML-állományt töltünk fel, amely egy `SchemaObject`-et ír le; ennek megalkotásához a paraméterei elemzésre kerülnek. A rendszer az XML-állományt nem tárolja. A módosítás végrehajtása után az XML-állomány példánya törlődik.

Az Oracle *i*FS osztályhierarchiájának számos objektuma létrehozható XML konfigurációs állományok használatával.

A `Document` osztály egy alosztályának létrehozása XML használatával

A `Document` osztály egy alosztályának létrehozásához szükségünk van egy új `ClassObject`-re. Az egyetlen kötelező mező a `ClassObject`, valamint szülő osztályának a neve. Például a `Folder` osztály nem rendelkezik saját attribútummal, hanem az ősosztályától, a `PublicObject`-től örökli azokat. Azonban a `Folder` osztálynak szüksége van egy saját alosztályra ahhoz, hogy lehetőségünk nyíljon a `Folder` objektumok egyedi feldolgozásához.

Az alosztályok létrehozásának az elsődleges oka mindazonáltal saját attribútumok hozzáadása. Az attribútum definíciók a `ClassObject` definíciójától különállón vannak tárolva az `Attribute` táblában. Ezzel lehetővé válik hozzáadni vagy elvenni attribútumokat a `ClassObject` osztálytól anélkül, hogy alosztályunk definícióját megváltoztatnánk. A `ClassObject` és az `Attribute` táblákban lévő bejegyzések felhasználásával az Oracle *i*FS automatikusan létrehozza az ODM_*alosztálynév* táblát, amelyben saját alosztályunk dokumentumainak példányait tárolja.

Tulajdonságcsoportok létrehozása XML konfigurációs állományok felhasználásával

Tulajdonságcsoportokat a `<PropertyBundle>` tag segítségével tudunk definiálni. A `<PropertyBundle>` tag tartalmazhat elemeket minden egyes attribútumhoz az Oracle *i*FS osztályhierarchiájának `PropertyBundle` osztályában.

A név/érték párok, avagy a tulajdonságok listája a `<Properties>` elemben van definiálva. A `<Properties>` tag `<Property>` tag-eket fog közre, amelyek a tulajdonságcsoportban szereplő összes tulajdonságobjektumot definiálják.

Mindegyik `<Property>` tag tartalmaz egy `<Name>` és egy `<Value>` tag-et. Ez utóbbi egy attribútuma a `DataType`, amely a tulajdonságobjektum adattípusát határozza meg (például `String`, `String_Array`, `PublicObejct_Array`). A `<Value>` tag nem egyezik meg közvetlenül a `Property` osztály egyik attribútumával sem. Ehelyett annak értéke, típusától függően, az egyik `Property` attribútumban lesz eltárolva.

A `<Property>` elemek használhatnak egy `Action` attribútumot, mellyel meghatározhatjuk, hogy az adott tulajdonságot hozzá szeretnénk adni, vagy törölni akarjuk a csoportból. Ennek alapján az attribútum értéke lehet `Add` vagy `Remove` (alapértelmezés az `Add`). Csoport létrehozásánál a `Remove`-nak természetesen nincs hatása. Tulajdonság cseréjénél, ugyanazzal a névvel először töröljünk, majd adjuk hozzá újból a csoporthoz.

Több objektum egyidejű létrehozása egyetlen XML konfigurációs állománnyal

Az Oracle *i*FS sémanyelve támogatja azon XML konfigurációs állományokat, melyek több objektumot hoznak létre. Ahhoz, hogy a konfigurációs állomány több objektumot is kezeljen, az elemek deklarációját össze kell fogni az `<ObjectList>` elemmel.

Több objektum létrehozásakor figyelnünk kell arra, hogy az objektumok deklarációja pontos legyen. Az XML-kódban hiba esetén az `IfsSimpleXmlParser` abbahagyja az objektumok elemzését, és egyetlen objektumot sem hoz létre. Ez alól a kivételt a `ClassObject` és a `DirectoryUser` létrehozása jelenti. Ezen típusú objektumok létrehozását explicit módon hagyja jóvá, így ha ezalatt lép fel hiba, módosítanunk kell az XML-állományt és törölnünk kell a már létrehozott objektumok definícióját.

Irodalomjegyzék

[1] Abbey, Michael–Corey, Michael C.–Abramson, Ian: ORACLE8*i* Kézikönyv kezdőknek. Panem, 2001, Budapest.

[2] Bhamidipati, Kishore: SQL programozói referenciakönyv. Panem, 1999, Budapest.

[3] Database Uri-references, Oracle9*i* Application Developer's Guide – XML.

[4] Gábor András–Juhász István: PL/SQL-programozás. Alkalmazásfejlesztés ORACLE9*i*-ben. Panem, 2002, Budapest.

[5] Bergsten, Hans: JavaServer Pages, Kossuth Kiadó, 2001, Budapest.

[6] Kende Mária–Kotsis Domokos–Nagy István: Adatbázis-kezelés az Oracle-rendszerben. Panem, 2002, Budapest.

[7] Loney, Kevin–Koch, George: Oracle8*i* Teljes referencia. Panem, 2001, Budapest.

[8] Melton, Jim–Simon, Alan R.: SQL:1999. Morgan Kaufmann Publishers, 2001, San Francisco.

[9] Bradley, Neil: Az XML kézikönyv, Szak Kiadó, 2000, Budapest.

[10] Nyékiné Gaizler Judit (szerk.): JAVA 2. Útikalauz programozóknak. ELTE TTK Hallgatói Alapítvány, 2000, Budapest.

[11] Oracle Internet File System Developer Reference.

[12] Oracle Internet File System User's Guide.

[13] Oracle JDeveloper on-line dokumentáció, http://otn.oracle.com/docs/products/jdev/content.html.

[14] Oracle Portal Building Portals.

[15] Oracle Portal Configuration Guide.

[16] Oracle Portal Tutorial.

[17] Oracle Single Sign-On Application Developer's Guide.

[18] Oracle Technology Network, http://otn.oracle.com/.

[19] Oracle9*i* Application Developer's Guide – XML.

[20] Oracle9*i* Application Developer's Guide – Large Objects(LOB's).

[21] Oracle9*i* Application Developer's Guide – Object-Relational Features.

[22] Oracle9*i* Application Server Documentation Library.

[23] Oracle9*i* Application Server PL/SQL Web Toolkit Reference, http://sun-doku.htw-aarland.de:8888/ora9doc/9iASRel2/doc/web.902/a90101/toc.htm.

[24] Oracle9*i* CORBA Developer's Guide and Reference.
[25] Oracle9*i* Database Concepts.
[26] Oracle9*i* Enterprise JavaBeans Developer's Guide and Reference.
[27] Oracle9*i* JDBC Developer's Guide and Reference.
[28] Oracle9*i* JSP Support for JavaServer Pages Developer's Guide and Reference.
[29] Oracle9*i* PL/SQL User's Guide and Reference.
[30] Oracle9*i* Servlet Engine Developer's Guide.
[31] Oracle9*i* SQL Reference.
[32] Oracle9*i* SQL*Plus User's Guide and Reference.
[33] Oracle9*i* SQLJ Developer's Guide and Reference.
[34] Oracle9*i*AS Portal Center, http://portalstudio.oracle.com/.
[35] Security and the Oracle Internet File System.
[36] Vég Csaba–Juhász István: Java – Start! Logos 2000, 1999, Debrecen.
[37] XML in the Oracle Internet File System.

Tárgymutató

Nyomtatta és kötötte a Kaposvári Nyomda Kft. – 230266
Felelős vezető: Pogány Zoltán igazgató

Adatbázis-*programozás*

Az ORACLE és a web

Haladó ORACLE 9*i* ismeretek

Az Oracle9*i* Release 1 az Oracle olyan terméke, amely alkalmas nagy megbízhatóságú, skálázható internetes és intranetes vállalati alkalmazások fejlesztésére és üzemeltetésére.

Az Oracle és a web – Haladó Oracle9*i* ismeretek című könyv az Oracle9i néhány eszközének lehetőségeit tárgyalja, amelyek segítségével olyan webes alkalmazások hozhatók létre, amelyeknél alapvető a kommunikáció milyensége és hangsúlyos a dokumentumkezelés. A könyv az alábbiakat tárgyalja:

- Java az Oracle-ben, Java tárolt alprogramok,
- az XML használatának lehetőségei,
- az Oracle Portal használata weboldalak kialakításánál,
- az Oracle Internet File System.

A könyv ismeretanyagának jó része receptszerűen, lépésekre bontva áll össze. A megértést számos példa, kódrészlet, szkript segíti.

ISBN 963-545-385-X

9 789635 453856

www.panem.hu

3900 Ft

PANEM