**OE NIK 2013**

**Táborhely Üzemeltető segédszoftver**

**Teljes dokumentáció**

**Lovas István K971UM**

# Tartalom Jegyzék:

Táborhely Üzemeltető segédszoftver dokumentáció.

# A Szoftver céljai és megszorításai

A web-es adatbázis-alkalmazás célja az, hogy egy tábor személyzetének megkönnyítse a tábor lakóival kapcsolatos adminisztrációt. A nyilvántartásban tárolja a lakók, a tábori csoportok, illetve a tábori férőhelyekkel kapcsolatos legfontosabb adatokat. A rendszer adminisztrátorának, az SQL futtatását megkönnyítő szakasszal is rendelkezik, illetve egy olyan részegységgel, mely információt biztosít a rendszer környezetének OpenCL képességeiről.

# A Szoftverrel kapcsolatos technikai elvárások és megszorítások

Az elkészítés során a következő nyelveket és technológiákat alkalmaztam:

* Az alkalmazás az Oracle11g APEX környezetének felhasználásával készült.
  + Oracle11g:
    - Egy nagyvállalati környezetbe készült, és ilyen környezetben közkedvelt adatbázis kezelő eszköz.
  + APEX (Application Express):
    - Az Oracle RAD eszköze, mely adatbázis intenzív webes alkalmazások egyszerű elkészítésére hivatott.
* SQL - Structured Query Language (strukturált lekérdezőnyelv):
  + Mely egy olyan deklaratív nyelv mely az adatbázis kezelés megkönnyítésére hivatott.
* PL/SQL - Procedural Language/Structured Query Language:
  + Az Oracle által kifejlesztett programozási nyelv, mely az SQL procedurális eszközökkel való kiegészítése.
* Java:
  + Egy olyan általános célú, objektumorientált programozási nyelv, aminek tervezése és megalkotása során nagy figyelmet fordítottak a robosztus, hibatűrő, biztonságos működésre, és hogy lehetőleg kevés ráfordítással lehessen, könnyen hordozható, platform független, vállalati szintű programokat előállítani a segítségével.
  + A fenti előnyei, és a tény, hogy jól együttműködik az Oracle adatbázis kezelő rendszerrel, kiváló választássá tette az adott feladatra.
* Aparapi:
  + Egy olyan Java nyelvi programozási felület (API), mely segíti a külső gyorsító eszközök programozását.
  + Tervezésénél ügyeltek arra, hogy használata ne legyen idegen egy Java programozó számára. Ezt olyannyira sikeresen teljesítették, hogy egy egyszerű kliens megírása nem sokkal bonyolultabb egy Java szál indításánál.
  + Az Oracle bejelentette, hogy a Java 9 OpenCl képességeit az Aparapi mintájára, annak eredményeit felhasználva kívánják implementálni.
  + Az Aparapi ezért egy nagyon előremutató és jó eszköz a kívánt cél elérésére.
* NetBeans:
  + Egy nyílt forráskódú, de az Oracle által is koordinált fejlesztésű IDE, a Java kódok előállítását segítette.

# A Szoftver beüzemelése és tesztadatokkal való feltöltése

Az összes felhasználandó script, és importálandó fájl megtalálható a   
CD\_MELLÉKLET\_GYÖKÉR\Scriptek\_Programok\_Fájlok\ mappában.

Illetve az összes Java külső eljáráshoz kapcsolódó forrás, illetve lefordított program megtalálható a  
CD\_MELLÉKLET\_GYÖKÉR\Java mappában.

A program beüzemelésének lépései:

## A program vázának elkészítése

A beüzemelés az f102.sql APEX-be való importálásával kezdődik.

## A táblák elkészítése:

### Csak scriptek segítségével:

A következő SQL fájlokat kell sorban meghívni SEMA1 nevében.

* 1a\_create\_tables.sql
* 1b\_alter\_constraints.sql
* 2a\_init\_test\_Házak.sql
* 2b\_init\_test\_Csoportok.sql
* 2c\_init\_test\_Lakók.sql

A fent látható scripteket fel lehet tölteni az APEX-be, és onnan egyenként meghívni (SQL Workshop > SQL Scripts), de egyszerűbb, ha lefuttatjuk az install\_tables.bat fájlt, mely csatlakozik sqlplus segítségével sema1-hez, és lefuttatja a nevében a fenti scripteket.

### Az APEX-be való importálás szövegfájlba kimentett táblák segítségével:

* + Az APEX SQL Workshop > Utilities > Data Workshop eszközével
    1. Data Load > Text Data eszközzel importálandó fájlok:
       - csoportok.txt
       - házak.txt
       - lakók.txt
    2. 1b\_alter\_constraints.sql script futtatása, a megszorítások létrehozása végett

## A Java nyelvű külső rutin beüzemelése:

A beüzemelés kezdő lépése a CD\_MELLÉKLET\_GYÖKÉR\Java\LJava könyvtár másolása a C:\LJava címre.

A

# Jogosultsági szintek a rendszerben

A rendszerben két jogosultsági szintet, azaz jogosultsági csoportot különböztetek meg, az „adminok”, illetve a „dolgozok” szintjét. A rendszer későbbi használhatóságát nagyban növeli, ha az egyes lapok elérhetőségét nem személyekhez, hanem csoportokhoz kötöm, így elég az új személy elhelyezni egy megfelelő csoportba, és a megfelelő jogok birtokában már használhatja is a rendszert.

Az adminok jogosultak elérni az SQL Parancs, a Paraméteres SQL, illetve az OpenCL részeket.

A dolgozok jogosultak elérni a Táblák Megjelenítése, és az Űrlap részeket.

A kezdőoldal bejelentkezés nélkül bárki számára elérhető.

Az összes lap és fül kapott engedélyezési mintát.

## A rendszerben megjelenő jogosultsági csoportok



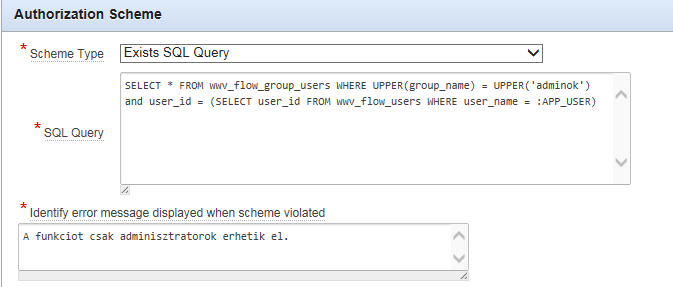
## A rendszerbe felvett felhasználók

## A rendszerbe felvett engedélyezési minták (Authorization Shemes)



### Adminoknak megvalósítása

SELECT \* FROM wwv\_flow\_group\_users WHERE UPPER(group\_name) = UPPER('adminok') and user\_id = (SELECT user\_id FROM wwv\_flow\_users WHERE user\_name = :APP\_USER)



### Dolgozóknak megvalósítása

SELECT \* FROM wwv\_flow\_group\_users WHERE UPPER(group\_name) = UPPER('dolgozok') and user\_id = (SELECT user\_id FROM wwv\_flow\_users WHERE user\_name = :APP\_USER)

# Betétlap

BETÉTLAP

Lovas István Féléves Feladatához

1. Autentikációs adatok:

- adminisztrátor / adminisztrátor

- dolgozó / dolgozó

2. A Féléves Feladat kiemelkedő képességei:

- Gyorsító eszközök Java nyelvből történő elérése

- Paraméteres SQL (kiemelve az új sor beszúrását)

# Táblaszerkezet tervezése:

## A feladat rövid leírása

* Egy ifjúsági tábor lakóival kapcsolatos fontosabb adatok tárolása.

## A tárolandó tulajdonságok leírása

|  |  |
| --- | --- |
| **A tárolandó adat** | **Rövid leírása** |
| név | a lakó neve |
| SzületésiDátum | a lakó születési dátuma |
| nem | a lakó neme |
| egészségbiztosítási\_szám | a lakó egészségbiztosítási száma |
| személyigazolvány\_szám | a lakó személyigazolvány száma |
| tartózkodas\_tól | az a dátum amikortól az adott lakó a táborban tartózkodik |
| tartózkodas\_ig | az a dátum ameddig az adott lakó a táborban tartózkodik |
| ország | a lakó származási országa |
| ház\_szám | a ház száma,a táboron belül a házak egytől számozódnak |
| ház\_megnevezés | a ház neve, a táboron belül lehet több azonos nevű ház is. |
| férőhely\_szám | egy szám, mely megmondja, hogy az adott ház hány személyt képes befogadni |
| csoport\_név | a tábor lakói csoportokat képeznek, táboron belül több csoportnak is lehet azonos a neve |
| csoportvezető | minden csoportnak csak egy vezetője van, de egy vezetőnek lehet több csoportja |
| csoport\_pontszám | egy-egy csoport által, a csapatversenyeken elért, összpontszámát tárolja |

## Az ősmodell:

* Lakók< név, SzületésiDátum, nem, egészségbiztosítási\_szám, személyigazolvány\_szám, tartózkodas\_tól, tartózkodas\_ig, ország, ház\_szám, ház\_megnevezés, férőhely\_szám, csoport\_név, csoportvezető, csoport\_pontszám >

## Függőségek:

1. Egy lakó pontos meghatározásához szükséges a neve és az egészségbiztosítási száma. **flakók** ={név, egészségbiztosítási\_szám } -> { SzületésiDátum, nem, személyigazolvány\_szám, tartózkodas\_tól, tartózkodas\_ig, ország, ház\_szám, ház\_megnevezés, férőhely\_szám, csoport\_név, csoportvezető, csoport\_pontszám }
2. Egy Csoport egyértelmű meghatározásához szükséges, a csoport neve, illetve a vezetője.  
   **fcsoportok** = { csoport\_név, csoportvezető } -> { csoport\_pontszám }
3. A házszámok különbözőek, ezért a házszámból egyértelműen képes megállapítani a rendszer, hogy melyik házról van szó. Megengedjük viszont, hogy több ház viselje ugyanazt a nevet.  
   **fházak** = { ház\_szám } -> { ház\_megnevezés, férőhely\_szám }

## A függőségek normalizálása

1. **flakók** normalizálása:
   1. 1NF-ben van, mert nincs összetett attribútuma
   2. 2NF-ben van, nincs olyan másodlagos attribútum, aminek a meghatározásához elegendő lenne az elsődleges attribútumok egy része.
   3. Nincs 3NF-ben, hisz **fcsoportok** és **fházak** belső függések, ezeket kiemelve:  
      **flakók1** = {név, egészségbiztosítási\_szám } -> { SzületésiDátum, nem, személyigazolvány\_szám, tartózkodas\_tól, tartózkodas\_ig, ország, ház\_szám, csoport\_név, csoportvezető }
   4. BCNF-ben van, hisz 3NF-ben van, és a baloldal egyetlen valódi részhalmaza sem függ másik elsődleges, vagy másodlagos attribútumtól.
2. **fcsoportok** normalizálása:
   1. 1NF-ben van, hisz nincs összetett attribútuma.
   2. 2NF-ben van, hisz a jobb oldal egy attribútumból áll, és az függ mindkét elsődleges attribútumtól.
   3. 3NF-ben van, hisz a jobb oldal csak egy elemből áll.
   4. BCNF-ben van, hisz az elsődleges attribútumok nem függnek egymástól, vagy a másodlagos attribútumtól.
3. **fházak** normalizálása:
   1. 1NF-ben van, hisz nincs összetett attribútuma.
   2. 2NF-ben van, hisz a baloldal csak egyetlen attribútumból áll.
   3. 3NF-ben van, hisz a másodlagos attribútumok között nincs összefüggés.
   4. BCNF-ben van, hisz 3NF-ben van, és a bal oldalán csak egyetlen attribútum áll.

## Az eredményfüggőségek összefoglalása

* **flakók1** = { név, egészségbiztosítási\_szám } -> { SzületésiDátum, nem, személyigazolvány\_szám, tartózkodas\_tól, tartózkodas\_ig, ország, ház\_szám, csoport\_név, csoportvezető }
* **fcsoportok** = { csoport\_név, csoportvezető } -> { csoport\_pontszám }
* **fházak** = { ház\_szám } -> { ház\_megnevezés, férőhely\_szám }

## Az eredmény adatmodell felírása

* Lakók < név, egészségbiztosítási\_szám, SzületésiDátum, nem, személyigazolvány\_szám, tartózkodas\_tól, tartózkodas\_ig, ország, *ház\_szám*, *csoport\_név, csoportvezető* >
  + Ahol a név és egészségbiztosítási\_szám attribútumok,   
    tehát a **Klakók** = { név, egészségbiztosítási\_szám } attribútum halmaz a Lakók relációra vonatkozó elsődleges kulcs.
  + A *ház\_szám attribútum a házak relációra vonatkozó idegen kulcs.*
  + *A* *csoport\_név és csoportvezető attribútumok pedig a csoportok relációra vonatkozó idegen kulcsok.*
* Csoportok *<*csoport\_név, csoportvezető, csoport\_pontszám>
  + A **Kcsoportok** = { csoport\_név, csoportvezető } attribútum halmaz a Csoportok relációra vonatkozó elsődleges kulcs.
* Házak < ház\_szám, ház\_megnevezés, férőhely\_szám >

A **Kházak** = { ház\_szám } attribútum halmaz a Házak relációra vonatkozó elsődleges kulcs.

# Táblák Implementációja

## A táblák szerkezete

A táblák tervben foglaltak szerint készültek.

A kívánt szöveges és szám mezők meghatározásakor, ahol tudtam, a lehető legnagyobb, még ésszerű foglalási méretet választani. Ezt néhol kutatás is megelőzte, például az ország név mező hosszának meghatározásakor számításba vettem ilyeneket, mint: The United Kingdom of Great Britain and Northern Ireland, mely 48 karakter hosszú. Az ország mezőt 50 hosszú VARCHAR2-ként vettem fel. A VARCHAR2 típusról tudni kell, hogy nem foglalódik le alapból a teljes meghatározott karakternyi méret (előző példában 50), ezért úgy láttam, érdemesebb túltervezni, ahol csak lehet. Az Egészségbiztosítási\_szám, illetve a Személyigazolvány\_szám fix hosszúságú szövegek, ennek megfelelő CHAR típusnak határoztam meg mind a kettőt. A szám típusoknál, mint a ház\_szám, illetve a férőhely\_szám három helyértéken tárol, a csoport pontszám egy tizedes jegy pontossággal, és öt egész helyértékkel tárol, ami a táborban minden bizonnyal elég lesz.

### A megvalósító script: 1a\_create\_tables.sql

ALTER SESSION SET NLS\_LANGUAGE= 'HUNGARIAN';

ALTER SESSION SET NLS\_TERRITORY= 'HUNGARY';

ALTER SESSION SET NLS\_DATE\_FORMAT= 'YYYY-MM-DD';

ALTER SESSION SET NLS\_DATE\_LANGUAGE = 'HUNGARIAN';

ALTER SESSION SET NLS\_ISO\_CURRENCY = 'HUNGARY';

ALTER SESSION SET NLS\_NUMERIC\_CHARACTERS = ',.';

ALTER SESSION SET NLS\_SORT = 'HUNGARIAN';

--SELECT \* FROM nls\_session\_parameters;

DROP TABLE Lakók;

CREATE TABLE Lakók (

Név VARCHAR2(30),

Egészségbiztosítási\_szám CHAR(9) UNIQUE,

SzületésiDátum DATE,

Nem CHAR(1) CHECK( Nem IN ('f','n') ),

Személyigazolvány\_szám CHAR(11) UNIQUE,

Tartózkodas\_tól DATE,

Tartózkodas\_ig DATE,

Ország VARCHAR2(50),

ház\_szám NUMBER(4),

Csoport\_név VARCHAR2(50),

Csoportvezető VARCHAR2(30)

);

DROP TABLE Csoportok;

CREATE TABLE Csoportok(

csoport\_név VARCHAR2(50),

csoportvezető VARCHAR2(30),

csoport\_pontszám NUMBER(5,1)

);

DROP TABLE Házak;

CREATE TABLE Házak (

ház\_szám NUMBER(3),

ház\_megnevezés VARCHAR2(25) NOT NULL,

férőhely\_szám NUMBER(3) NOT NULL

);

A fentiekben látható, hogy a Lakók táblában az Egészségbiztosítási\_szám, amellett, hogy az elsődleges kulcs része, még egyedinek is jelöltem. Egyedi továbbá a Személyigazolvány\_szám mező is. Az egészségbiztosítási szám, mivelhogy az elsődleges kulcs része, kötelező a kitöltése is. A Személyigazolvány\_szám kitöltése ellenben nem kötelező, hisz lehetséges, hogy kiskorú személy is tartózkodik a tábor területén.

## A táblák megszorításai

A terv alapján megalkottam az elsődleges, illetve az idegen kulcsokat.

### A megvalósító script: 1b\_alter\_constraints.sql

ALTER TABLE Lakók ADD CONSTRAINTS p\_Lakók PRIMARY KEY (Név,Egészségbiztosítási\_szám);

ALTER TABLE Csoportok ADD CONSTRAINTS p\_Csoportok PRIMARY KEY (csoport\_név,csoportvezető);

ALTER TABLE Házak ADD CONSTRAINTS p\_Házak PRIMARY KEY (ház\_szám);

ALTER TABLE Lakók ADD CONSTRAINTS f\_Lakók\_Csoportok FOREIGN KEY (Csoport\_név,Csoportvezető)

REFERENCES Csoportok (csoport\_név,csoportvezető);

ALTER TABLE Lakók ADD CONSTRAINTS f\_Lakók\_Házak FOREIGN KEY (ház\_szám)

REFERENCES Házak (ház\_szám);

# Implementáció részleteinek leírása

Általános megjegyzések

## Kezdőlap

### A komponens célja

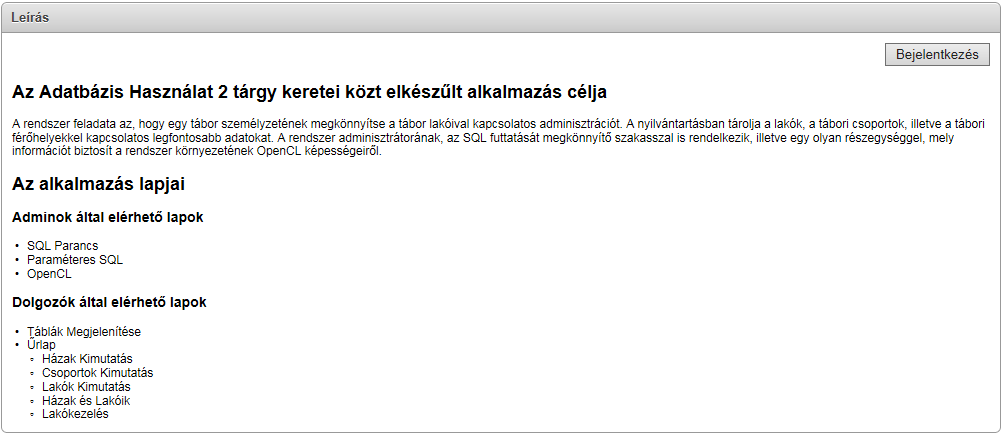
A kezdőlap általános leírást nyújtani a webalkalmazás céljairól, és használatáról. Innen lehet továbbá bejelentkezni, és elérni az adott felhasználó jogosultsági szintjének megfelelő lapokat.

A kezdőoldal bárki számára látható.

### A megvalósításának fontosabb részletei

Az információt egyszerű HTML régióban tárolja.  
Azt, hogy az oldalt bárki láthassa a Kezdőlap > Page Attributes > Security > Authentication választómezőjének Page Is Public értékre állításával értem el.

### Futási eredménye



## Táblák Megtekintése

### A komponens célja

Könnyen kezelhető eszköz a rendszerben szereplő táblák megjelenítésére.

Táblaválasztó súgószövege:  
Válasszon táblát, majd jelölje be a megjelenítendő oszlopokat, és ha szükséges, megadhat egy oszlopot, mely szerint az adatokat rendezi a rendszer.

Csak dolgozók láthatják.

### A megvalósításának fontosabb részletei

* A táblát egy legördülő listából lehet választani
  + értéklista lekérdezése:  
    select distinct table\_name, table\_name as "a" from user\_tables  
    where table\_name not like 'APEX%' AND table\_name not like 'DEMO%';
* A tábla kiválasztása után az adott tábla oszlopait checkbox segítségével lehet kiválasztani:
  + értéklista lekérdezése:  
    select column\_name as val, column\_name from all\_tab\_columns where table\_name = :P2\_TV
* A rendezést rádiógomb segítségével lehet kiválasztani:
  + értéklistája:  
    STATIC:Növekvő;ASC,Csökkenő;DESC,Nincs;NO
* A rendező oszlopot egy automatikus szövegkiegészítést nyújtó szövegdobozban lehet megadni
  + értéklista lekérdezése: megegyezik a checkbox lekérdezésével.
* Az oldal elküldése után két validáció futhat le, az egyik a kiválasztott tábla változtatásánál nulláz minden attól függő mezőt, míg a másik lekérdez gomb hatására ellenőrzi, hogy minden szükséges mezőt kitöltött-e a felhasználó.
* A lekérdezés szövegét egy folyamat állítja össze egy rejtett mezőbe.

DECLARE

tabla VARCHAR2(50);

oszlopok VARCHAR2(200);

sql\_str VARCHAR2(300);

rendez VARCHAR2(50);

BEGIN

:P2\_HID\_SQL\_CONTAINER := '';

tabla := :P2\_TV;

oszlopok:= apex\_util.table\_to\_string(apex\_util.string\_to\_table(:P2\_CHK\_OSZLOPOK,':'),',');

if oszlopok is null then

oszlopok := '\*';

end if;

if :P2\_RDO\_SORT != 'NO' then

rendez:='ORDER BY' || ' ' || :P2\_TB\_SORT\_FIELD || ' ' || :P2\_RDO\_SORT;

end if;

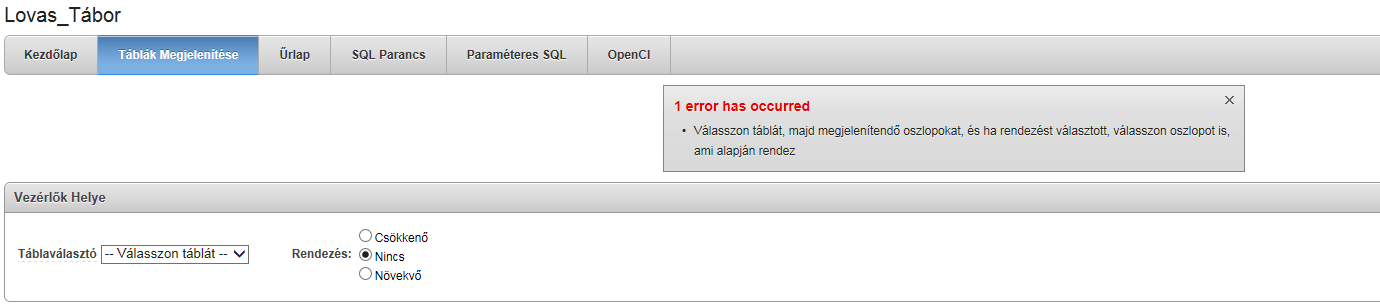
sql\_str := 'SELECT '|| oszlopok ||' FROM '|| tabla || ' ' || rendez;

:P2\_HID\_SQL\_CONTAINER := sql\_str;

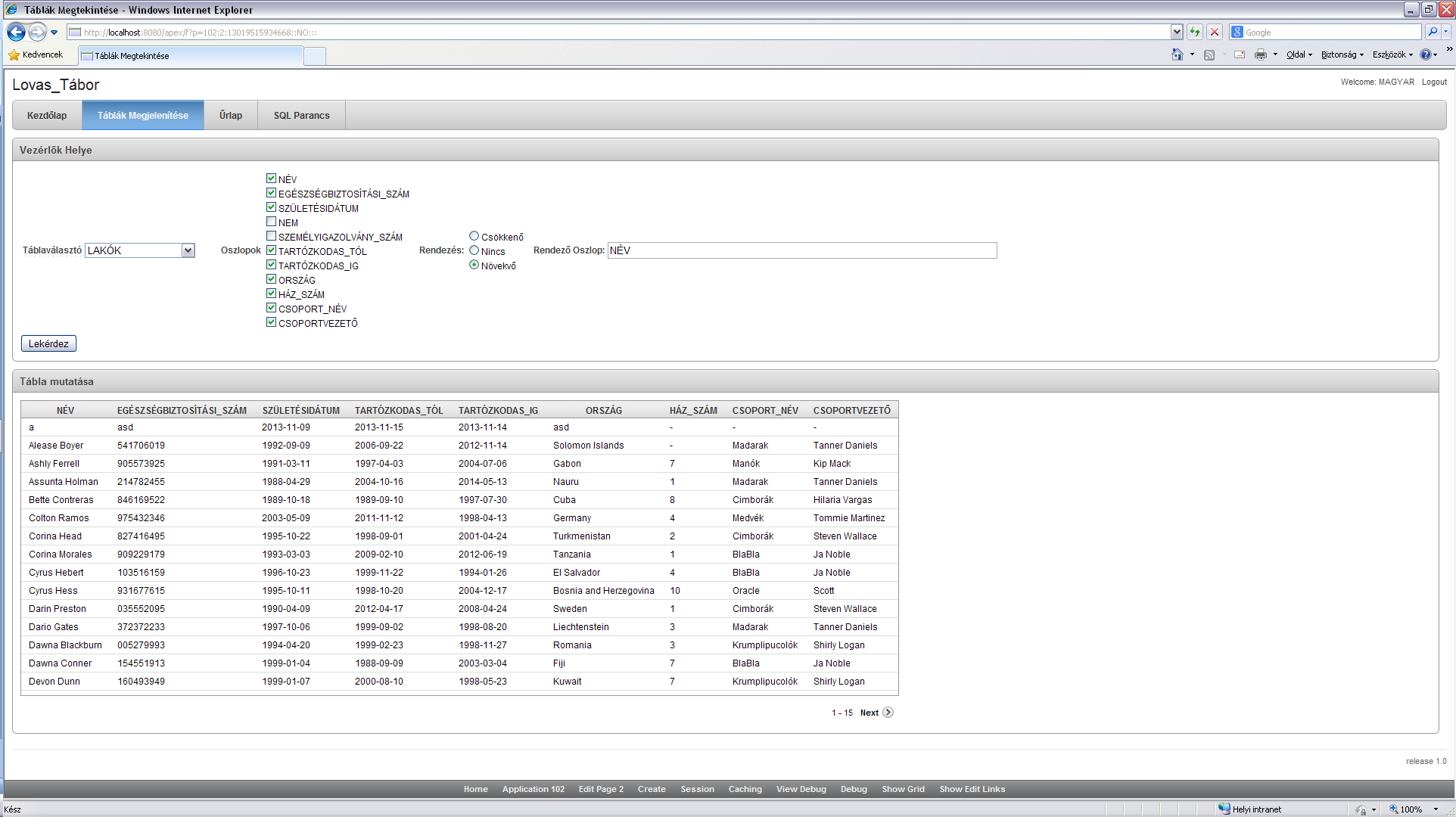
END;

### Futási eredménye

Az alábbi kép bemutatja azt az esetet, amikor a felhasználó nem tölti ki a megfelelő mezőket, látható a magyar nyelvű figyelmeztető szöveg.



Az alábbi kép a helyes működést ábrázolja.



## Űrlap

### A komponens célja

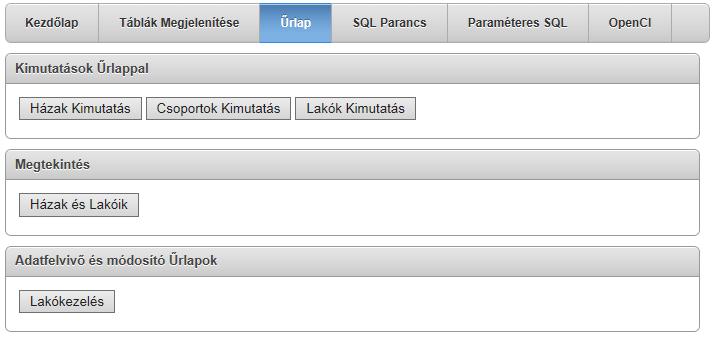
Innen érhetik el a dolgozók a rendszer által nyújtott űrlapokat (form) és kimutatásokat (report)

### A megvalósításának fontosabb részletei

A különböző alegységeket régiók különítik el, és a régiókban gombok vannak, melyek segítségével el lehet jutni a konkrét űrlaphoz vagy kimutatáshoz.

Az alegységeket külön-külön részletezem.

### Futási eredménye



## Kimutatások Űrlappal

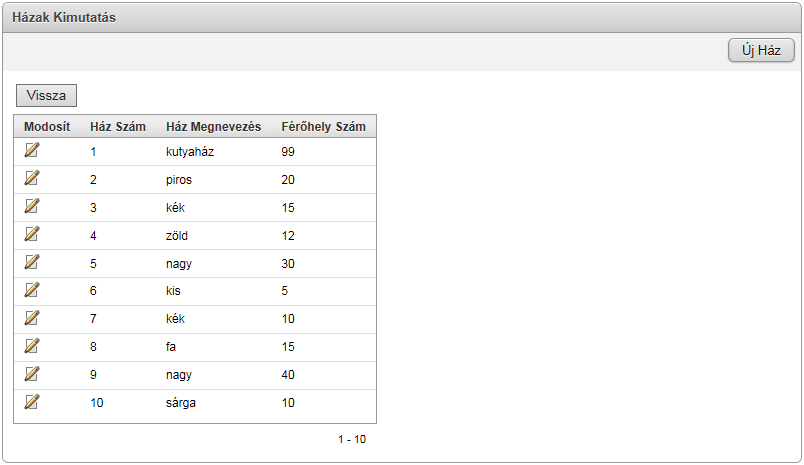
### A komponens célja

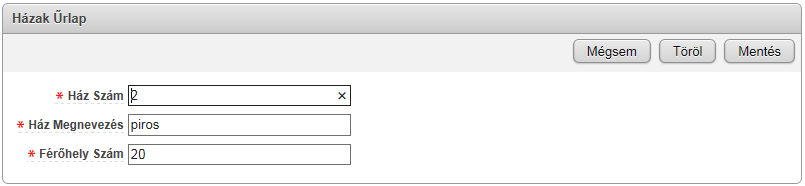
Az alegység egyszerű felületet biztosít a házak, csoportok, és lakók kezelésére.

### A megvalósításának fontosabb részletei

Az alegység összes lapja az APEX Form on Table with Report eszközével készült. A kimutatási oldal és az űrlap oldal kapcsolatát mindenhol a rowid segítségével teremtem meg, mely kényelmesebb kezelést biztosít összetett kulcsok, esetén, továbbá kiküszöböli a környezet szöveges kulcsok getes paraméterként való küldésénél fellépő karakterkódolási problémáját. Az űrlap oldalakon, amennyiben dátum mező nevében ékezet volt, a generált item nevét átírtam ékezetmentesre, a DateTime picker ilyenkor fellépő hibájának kiküszöbölése végett. A generált szövegdobozokat ahol lehetett, és célszerű volt, legördülő listára cseréltem.

### Futási eredménye





A fenti képeken a Ház Kimutatás látható, a Csoportok illetve a Lakók Kimutatás is hasonlóképp működik.

## Megtekintés -> Házak és lakóik

### A komponens célja

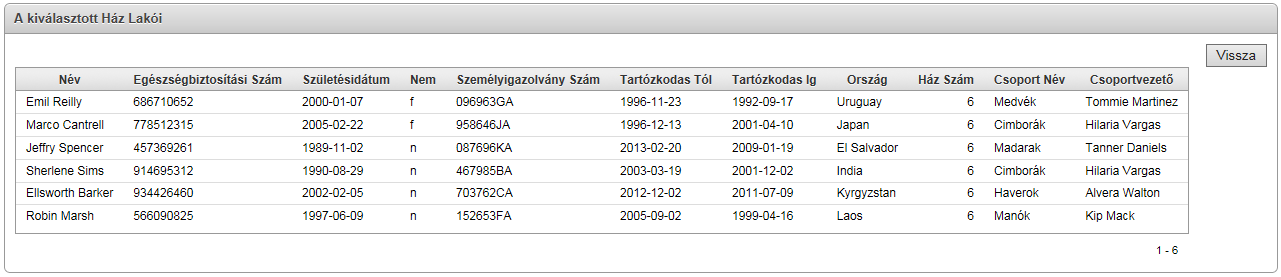
Az alegység házak és a kiválasztott ház lakóinak egyszerű megjelenítését szolgálja.

### A megvalósításának fontosabb részletei

Droll Down módszert használtam, az APEX Wizard Report eszközének segítségével. A kapcsolatot megteremtő házszám mező kiválasztott értékét égy rejtett mezőben tároltam.

### Futási eredménye





## Adatfelvivő és módosító Űrlapok -> Lakókezelés

### A komponens célja

A lakókról tárolt nagy mennyiségű információra és a lakók nagy számára való tekintettel, a lakók kezelésére, ez az alrendszer, egy a Lakók Kimutatásnál látottnál hatékonyabb eszközt biztosít a dolgozók részére.

### A megvalósításának fontosabb részletei

A megvalósítás során az APEX Tabular Form eszközét használtam. A generált szövegdobozokat ahol lehetett, és célszerű volt, legördülő listára cseréltem. A rendszerben egy csoportot vezetője és neve együttesen határoz meg, ez a szakasz viszont a felhasználók már meglévő csoportokhoz rendelésére is használható. Ezért úgy döntöttem, hogy a már meglévő csoportnév - csoportvezető kombinációkat egyetlen legördülő listában jelenítem meg a felhasználó munkájának megkönnyítésére.

A Csoportok legördülő lista értéklistáját az alábbi lekérdezés adja:

SELECT "CSOPORT\_NÉV" || ' - ' || "CSOPORTVEZETŐ" AS "Csoportnev", "CSOPORT\_NÉV" || ' - ' || "CSOPORTVEZETŐ" AS "Csoportval" FROM csoportok

Ennek megfelelően a feldolgozást is át kellett írni:

Declare

csap\_n Varchar(500);

csap\_v Varchar(500);

idx NUMBER(4);

len NUMBER(4);

BEGIN

csap\_n := '';

csap\_v := '';

for i in 1..apex\_application.g\_f02.count loop

--Coportnév - Csopotrvezető

csap\_n := '';

csap\_v := '';

idx := instr(apex\_application.g\_f11(i),'-');

csap\_n := SUBSTR(apex\_application.g\_f11(i),1,idx-2);

len := length(apex\_application.g\_f11(i));

len := len - (idx+1);

csap\_v := SUBSTR(apex\_application.g\_f11(i),idx+2,len);

if apex\_application.g\_frowid(i) is not null then

--Update

UPDATE Lakók

set

NÉV = apex\_application.g\_f02(i),

EGÉSZSÉGBIZTOSÍTÁSI\_SZÁM = apex\_application.g\_f03(i),

SZÜLETÉSIDÁTUM = TO\_DATE(apex\_application.g\_f04(i),'YYYY-MM-DD'),

NEM = apex\_application.g\_f05(i),

SZEMÉLYIGAZOLVÁNY\_SZÁM = apex\_application.g\_f06(i),

TARTÓZKODAS\_TÓL = TO\_DATE(apex\_application.g\_f07(i),'YYYY-MM-DD'),

TARTÓZKODAS\_IG = TO\_DATE(apex\_application.g\_f08(i),'YYYY-MM-DD'),

ORSZÁG = apex\_application.g\_f09(i),

HÁZ\_SZÁM = apex\_application.g\_f10(i),

CSOPORT\_NÉV = csap\_n,

CSOPORTVEZETŐ = csap\_v

WHERE rowid = apex\_application.g\_frowid(i);

else

--insert

INSERT INTO Lakók

(

Név,

Egészségbiztosítási\_szám,

SzületésiDátum,

Nem,

Személyigazolvány\_szám,

Tartózkodas\_tól,

Tartózkodas\_ig,

Ország,

ház\_szám,

Csoport\_név,

Csoportvezető

)

VALUES

(

apex\_application.g\_f02(i),

apex\_application.g\_f03(i),

TO\_DATE(apex\_application.g\_f04(i),'YYYY-MM-DD'),

apex\_application.g\_f05(i),

apex\_application.g\_f06(i),

TO\_DATE(apex\_application.g\_f07(i),'YYYY-MM-DD'),

TO\_DATE(apex\_application.g\_f08(i),'YYYY-MM-DD'),

apex\_application.g\_f09(i),

apex\_application.g\_f10(i),

csap\_n,

csap\_v

)

;

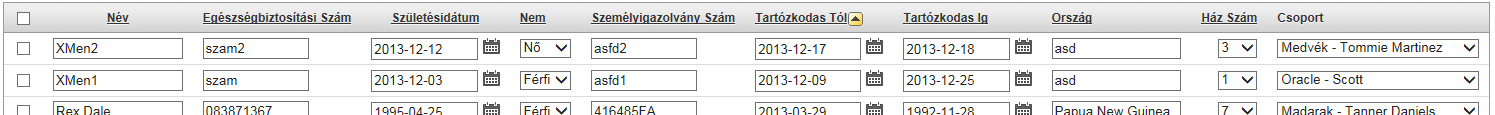
end if;

end loop;

END;

### Futási eredménye





Az fenti képek bemutatják, két sor egyszerre történő sikeres felvételét.

## SQL Parancs

### A komponens célja

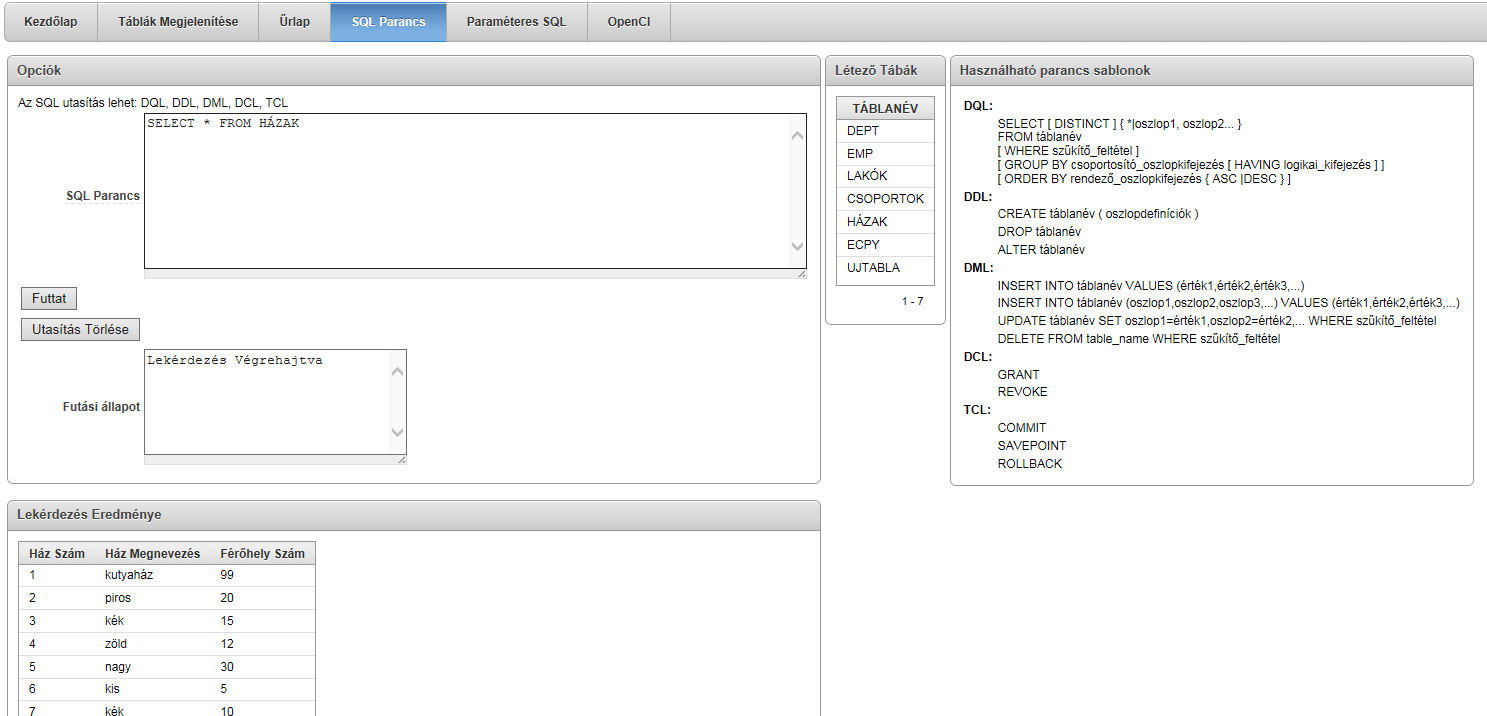
Az adminisztrátoroknak biztosít SQL parancsfuttatási lehetőséget. A komponens a parancs megírását parancssablonokkal és a felhasználó tábláinak a megjelenítésével segíti.

### A megvalósításának fontosabb részletei

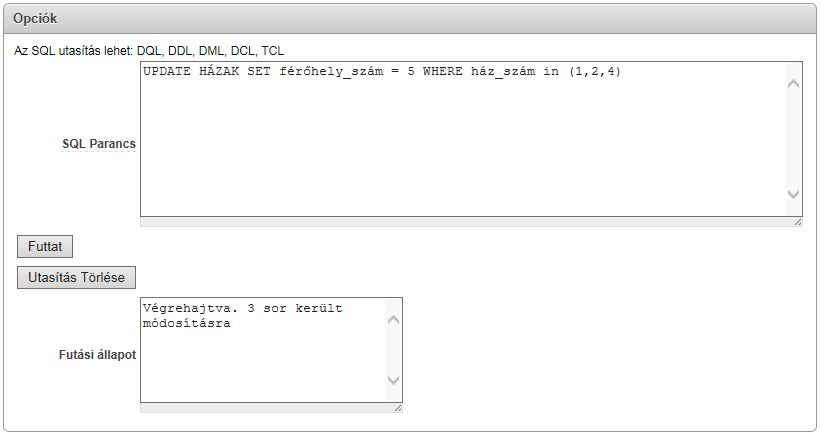
A felhasználói parancs végrehajtására az EXECUTE IMMEDIATE PL/SQL parancsot használtam. Amennyiben a parancs eleje, nagybetűssé alakítva, SELECT, akkor a lekérdezést átadom egy Riportnak, ellenkező esetben az SQL%ROWNUM segítségével megadom a módosított sorok számát.

### Futási eredménye

Egy lekérdezés eredménye:



Egy módosító SQL szkript végrehajtása:



## Paraméteres SQL

### A komponens célja

Az adminisztrátoroknak biztosít dinamikus SQL összeállítási, és futtatási módot.

### A megvalósításának fontosabb részletei

Az SQL Típusa legördülő lista alapján jelennek meg, az adott parancs összeállítására szolgáló beviteli elemek. A kötelezően kitöltendő beviteli mezők kitöltését ellenőrzöm, az összetett parancs helyességét viszont az adatbázis által kiváltott kivételek segítségével ellenőrzöm. Ezeket a kivételeket elkapás után minden esetben továbbdobom, hogy a felületen is megjelenjen, hisz ez az alegység adminisztrátoroknak szól, kik így minden információt megkaphatnak, mely segítheti feladatuk hibamentes megoldását. Az összes rész az SQL dinamikus összeállításával és annak EXECUTE IMMEDIATE paranccsal való futtatásával végzi a dolgát, kivéve az alább részletezett „Új sor felvétele a kiválasztott táblába” nevű részfunkció.

#### Új sor felvétele a kiválasztott táblába

##### Tábla oszlopainak bekérése

Egy kiválasztott tábla hatására dinamikusan listázásra kerül minden oszlopnak megfelelően egy beviteli mező:

FOR i in (SELECT DISTINCT column\_name from USER\_TAB\_COLUMNS where table\_name = :P15\_ACT\_TABLE)

LOOP

htp.p('<div style="min-width:200px; display: inline-block;">' || i.column\_name ||' </div> '||APEX\_ITEM.TEXT(50) || '<br />' );

END LOOP;

htp.p('<br />');

##### A feldolgozásnál felhasználom az Oracle DBMS\_SQL csomagját:

DECLARE

str VARCHAR2(1000);

vals VARCHAR2(500);

TYPE idxt IS TABLE OF NUMBER INDEX BY PLS\_INTEGER;

j NUMBER(2);

idx\_index NUMBER(2);

inx\_es idxt;

curid INTEGER;

ret INTEGER;

BEGIN

if :P15\_ACT\_TABLE is null

then

raise\_application\_error(-20001, 'Adjon meg egy kezelendő táblát (Kijelőlt tábla)');

end if;

str := 'INSERT INTO ' || :P15\_ACT\_TABLE || '( ';

vals := 'VALUES ( ';

j := 0;

idx\_index := 0;

FOR i in (SELECT DISTINCT column\_name from USER\_TAB\_COLUMNS where table\_name = :P15\_ACT\_TABLE)

LOOP

j := j + 1;

if apex\_application.g\_f50(j) is not null

then

if idx\_index > 0

then

str := str || ', ';

vals := vals || ', ';

end if;

idx\_index := idx\_index + 1;

str := str || i.column\_name;

vals := vals || ':p'|| idx\_index;

inx\_es(idx\_index) := j;

end if;

END LOOP;

str := str || ' ) ';

str := str || vals || ' )';

curid := DBMS\_SQL.OPEN\_CURSOR;

DBMS\_SQL.PARSE(curid, str, DBMS\_SQL.NATIVE);

for i in inx\_es.FIRST..inx\_es.LAST

loop

DBMS\_SQL.BIND\_VARIABLE(curid, 'p' || i, apex\_application.g\_f50(inx\_es(i)));

end loop;

ret := DBMS\_SQL.EXECUTE(curid);

DBMS\_SQL.CLOSE\_CURSOR(curid);

:P15\_GEN := str;

:P15\_EREDMENY := 'Sikeres végrehajtás';

EXCEPTION WHEN OTHERS THEN

IF DBMS\_SQL.IS\_OPEN(curid) THEN

DBMS\_SQL.CLOSE\_CURSOR(curid);

END IF;

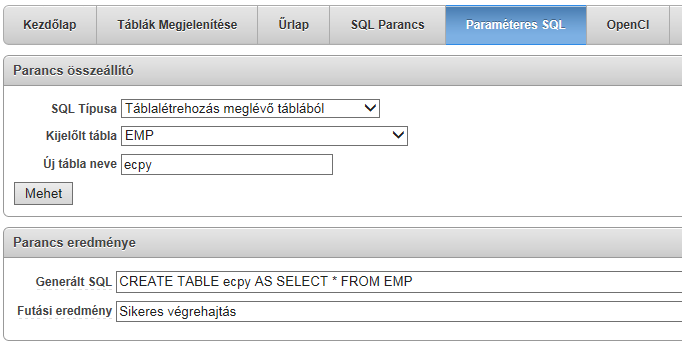
:P15\_EREDMENY := 'A végrehajtás ismeretlen okból megszakadt, ellenőrizze a lekérdezést és próbálja újra';

RAISE;

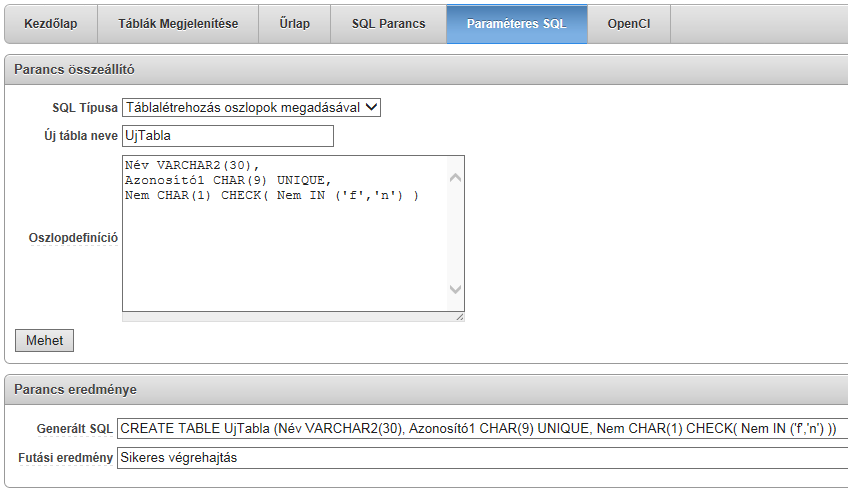
END;

### Futási eredménye

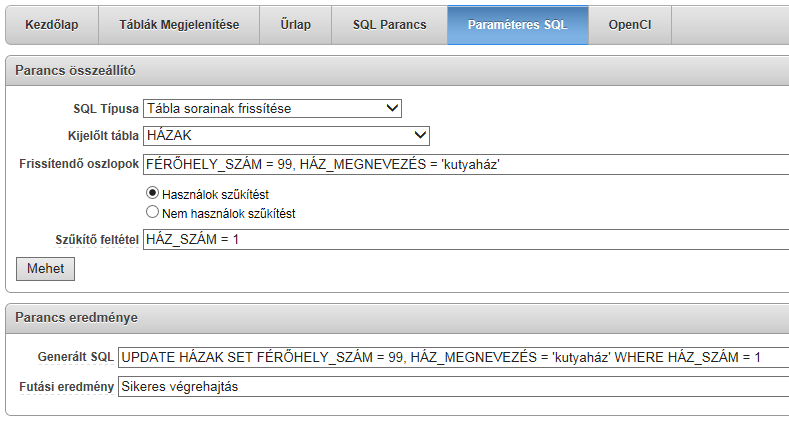
#### Táblalétrehozás meglévő táblából rész futása



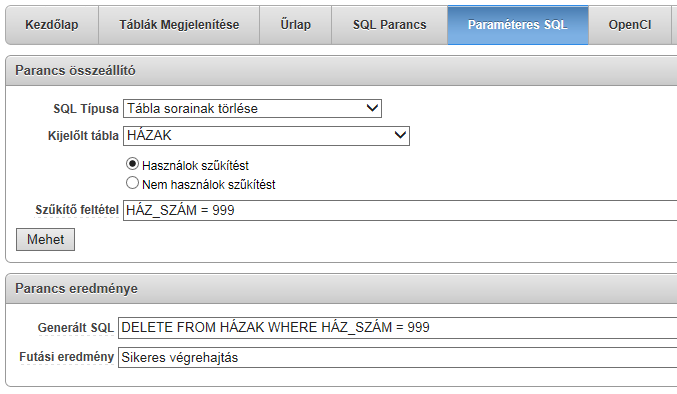
#### Táblalétrehozás oszlopok megadásával rész futása



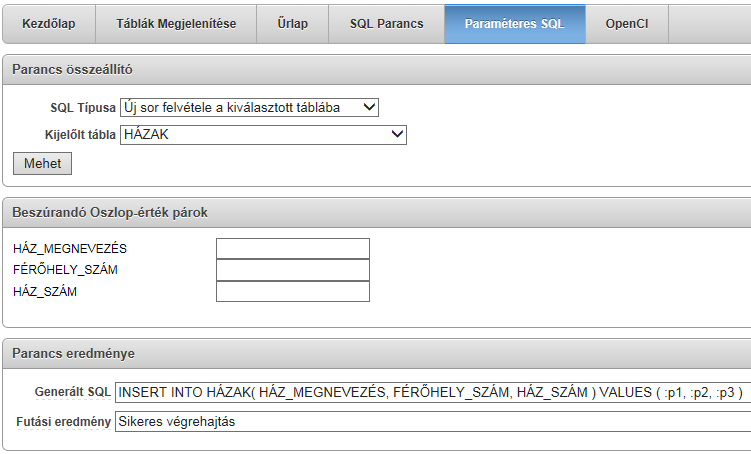
#### Tábla sorainak frissítése rész futása



#### Tábla sorainak törlése



#### Új sor felvétele a kiválasztott táblába



## OpenCL

### A komponens célja

Adminisztrátoroknak nyújt információt az adatbázist futtató rendszer Aparapi – OpenCL képességeiről.

### A megvalósításának fontosabb részletei

Az eddigiektől eltérően nem csak a végeredményt írom le, mert úgy gondoltam, a megoldás kutatása közben bejárt zsákutcák is tanulságosak lehetnek, egy másik esetben.

Az Oracle számos megoldást nyújt külső Java nyelvű rutinok meghívására. Melyeket a saját Oracle Aurora JVM környezetében futtat.

#### Java Stored Procedures

Az első tárgyalt módszer a java nyelvű tárolt eljárás. Mely elkészítése során először a szükséges java forrás vagy lefordított class fájlokat, illetve a különböző erőforrás fájlokat feltöltjük az adatbázisba egy séma objektumként, majd egy PL/SQL burkolót írunk, annak a meghívandó public static elérhetőségű metódusaira.

##### A java futtatásának képességét le lehet ellenőrizni a következő módon:

SQL> SELECT COUNT(\*) FROM all\_objects WHERE object\_type LIKE 'JAVA%';

amennyiben a fent látható lekérdezés eredménye 0, az adatbázis adminisztrátornak le kell futtatnia a nitjvm.sql nevű scriptet, mely az $ORACLE\_HOME/javavm/install/ könyvtárban található.

##### A feltöltés történhet:

A loadjava parancssoros segédprogrammal, a segédprogram adatbázisbeli dbms\_java.loadjava nevű elérőjével, vagy a CREATE JAVA SQL parancs segítségével. Ezek közül a lodajava segédprogram irodalma a legnagyobb, és elég egyszerű is a kezelése, csak a feltöltendő file mappájából, egy parancsablakban ki kell adni egy  
loadjava –user felhasználónév/jelszó fájl.kiterjesztés formájú parancsot.  
A támogatott feltölthető kiterjesztések: .class, .java, .properties, .sqlj, .ser, .jar, illetve .zip.  
A loadjava segédprogram legfontosabb kapcsolói:

* user: megadja azt a felhasználót, akinek a sémájához feltölti az adott fájlt, futtatási jogot is ad az adott felhasználónak a fájlra.
* grant: megadható olyan felhasználó, aki úgyszintén futtatási jogot kap a fájlra.  
  ugyanez SQL-ből: grant execute on myclass to scott;  
  Amint látható, hogy itt a jogosultságot fájlonként kell kiadni, nincs egységes java futtatási grant opció (~8g környékén még volt biztonsági okokból tértek át erre a változatra)
* resolve: hasznos kapcsoló, ami azt ellenőrzi, hogy a feltöltött osztályok közti függőségi viszonyok rendben vannak-e. Az alap resolver, amit az Oracle biztosít a szükséges osztályokat a public területen (ami a szabványos java könyvtárakat is tárolja), illetve az adott felhasználó területén keresi.
* resolver: ha az alap resolver nem elég, egyszerűen megadhatunk sajátot.   
  pl: resolver ( (\* SCOTT) (\* PUBLIC) (my/gui/\* -) )  
  a fent említett minta a keresést a SCOTT felhasználó, területén, majd a PUBLIC területen végzi, de a my/gui csomag összes függőségi hibáját figyelmen kívül hagyja. A \*, helyet használhatunk, java import-szerű eléréseket, csak itt a csomagokat elválasztó pontot a perjel váltja fel.
* a ? illetve a help kapcsolók számos további kapcsoló leírását biztosítják angol nyelven.

A feltöltés sikerességének ellenőrzése:  
Lekérdezhető az ALL\_OBJECTS illetve a USER\_OBJECTS nevű adatbázis nézet mely tárolja a feltöltött objektumokat.

Hasznos lekérdezések lehetnek például:

* SELECT \* FROM all\_objects WHERE object\_type='JAVA CLASS' and owner='SCOTT';
* SELECT object\_name, object\_type, status  
  FROM user\_objects  
  WHERE object\_type IN ('JAVA SOURCE', 'JAVA CLASS', 'JAVA RESOURCE')  
  ORDER BY object\_type, object\_name;

##### A PL/SQL burkoló

CREATE OR REPLACE FUNCTION helloworld RETURN VARCHAR2 AS

LANGUAGE JAVA NAME 'Hello.world () return java.lang.String';

Hívás SQL-ből:

SQL> CALL helloworld() INTO :myString;

HÍVÁS PL/SQL-ből:

ret := helloworld();

##### Módszer értékelése:

A módszer fő előnye, hogy bármelyik publikus statikus metódus meghívható a segítségével, kényelmesen PL/SQL kódból, és a visszatérés is könnyen változóba menthető. Előnye továbbá, hogy az Oracle Aurora java virtuális gépen fut a kód, mely különböző eljárásoknak hála képes a megszokottnál gyorsabb java kódfuttatásra, vagy akár képes natív kódra fordítani a java byte kódot. Nézőpontomból viszont pont ez tette a módszert számomra alkalmatlanná, hisz az Aurora verziója lassabban növekszik, mint a hagyományos, piacra szánt java verziója. Az Aparapi, viszont megkövetel egy minimális java verziót, mindamellett natív részei is vannak, mely ütközik az Aurora biztonsági előírásaival.

#### Command-Line Interface

Ez a módszer, egy osztály, vagy jar elérési útjának megadásával, séma objektumnak való feltöltés nélkül képes meghívni annak main metódusát.

##### A módszer kulcsa

A java hívást a dbms\_java csomag runjava, illetve runjava\_in\_current\_session nevű függvényeivel lehet megvalósítani. A runjava függvény, a runjava\_in\_current\_session függvénnyel ellentétben meghívásával töröl minden esetlegesen még feldolgozás alatt álló kérelmet. A külső java rutinoknak van alapból egy 30 perces futási korlátja, melyet természetesen megnövelhetünk.

##### Egy PL/SQL példa a runjava használatára

set serveroutput on

DECLARE

l\_output dbms\_output.chararr;

l\_lines number;

ret varchar2(50);

BEGIN

dbms\_output.enable(1000000);

dbms\_java.set\_output(1000000);

ret := dbms\_java.RUNJAVA('-cp C:\LJava JDelete korte.txt');

l\_lines:=3;

dbms\_output.get\_lines(l\_output, l\_lines);

for i in 1 .. l\_lines loop

dbms\_output.put\_line( i || '. sor: ' ||l\_output(i));

end loop;

END;

##### A runjava legfontosabb kapcsolói

* cp, vagy classpath a megszokott java classpath, mely kettőspontokkal elválasztva megadja az összes olyan elérési utat, ahol a meghívandó osztályok lehetnek. Az Aurora JVM itt keresi az osztályokat.

##### A szabványos kimenettel való munka:

* System.out.println("szöveg");
  + A java oldali kiírást szolgáló kódsor
* dbms\_output.put\_line(’szöveg’);
  + Adatbázis oldali kiíratás
* dbms\_output.enable(1000000);
  + PL/SQL-ben engedélyez egy megadott méretű kimenet-puffert
* dbms\_java.set\_output(1000000);
  + PL/SQL-ben a java kimenetét átirányítja az adatbázis kimenetére
  + Ezután minden java println, úgy fog viselkedni, mintha egy dbms\_output.put\_line() hívást adtunk volna ki.
* dbms\_output.get\_lines(:puffer, :sorok);
  + Egy adott pufferbe betölt megadott sornyi kimenetet.

##### Módszer értékelése:

Az előző tárolt eljárásoknál, egyszerűbb elkészíteni, ellenben csak a main metódus hívható, és annak csak a hagyományos módon lehet argumentumokat átadni. Mivel a java nyelvben a main metódus csak void visszatérésű lehet, ezért csak a szabványos kimenetre való írással képes a java kódunk visszaüzenni. A tárolt eljárások esetén a szükséges statikus metódust lehetett meghívni, itt viszont a java main metódust kell egy bemenő paraméterrel koordinálni, például az első bemenő paraméter lehet a main által meghívandó metódus neve (a mainben egy switch-case szerkezet választ az arg[0] alapján), a második pedig annak a paraméterei. A módszer használatát ugyancsak az Aurora JVM megkötései miatt, és a kimenet feldolgozásának nehézségei miatt vetettem el.

#### Végül egy kerülő megoldást választottam:

Miután világossá vált, hogy a beépített Aurora JVM nem felel meg a céljaimnak, módot kellett találnom az elkészült jar fájl futtatására, egy hagyományos JVM segítségével.

Ezt a feladatot úgy oldottam meg, hogy készítettem egy bat fájlt, ami kiadja egy   
java –jar elérés/file.jar formájú utasítást.

Ilyenkor paramétert lehet ugyan átadni a meghívandó programnak, de az csak adatbázis kapcsolaton (JDBC), vagy fájlba írás segítségével képes az adatbázis irányába kommunikálni. Jól látható, hogy ez a megoldás a legkevésbé sem elegáns, de a kényszerfeltételek mégis azt diktálták, hogy ezt a megoldást válasszam.

##### Először megközelítés

A runjava segítségével próbáltam meghívni egy olyan java kódot, mely kiad egy Runtime.getRuntime().exec() utasítást.

Az utasítás formája:

* Runtime.getRuntime().exec("cmd /c " + command);
  + Egy egyszerű parancs meghívása egy új parancsablak segítségével, mely ezután be is zárul.
* Runtime.getRuntime().exec("cmd /c start " + file + ".bat”);
  + Az előzőhöz hasonló, de bat fájl Java exec segítségével való futtatásához kell a start parancs is.

Probléma a megközelítéssel:

* java.io.FilePermission <<ALL FILES>> execute…
* Jogosultság problémája.  
  (a hiba a java oldalon váltódott ki, az Aurora biztonsági házirendje alapján)

Megoldási kísérlet:

* call dbms\_java.grant\_permission('SCOTT', 'SYS:java.io.FilePermission','<<ALL FILES>>', 'read, write, execute, delete');
* SELECT \* FROM USER\_JAVA\_POLICY
* Kísérletet tettem Java oldalú megoldás elkészítésére is, sikertelenül

A problémát nem sikerült hosszú kutatás során sem orvosolni, így másik megoldás után kellett néznem. Végül a DBMS\_SCHEDULER csomagot alkalmasnak találtam a céljaim elérésére.

#### A DBMS\_SCHEDULER csomag használata

Az Oracle ütemező képes külső parancsfájlok meghívására. Alaphelyzetben ütemezhető, esetleg esemény által kiváltott feladatok készíthetők vele, de szerencsére van mód egy feladat szükség szerinti kódból történő meghívására is.

##### A csomag fontosabb függvényei:

* DBMS\_SCHEDULER.create\_job( params );
  + egy parancsra végrehajtható feladat készíthető vele
* DBMS\_SCHEDULER.drop\_job('feladat\_név');
  + segítségével törölhető egy feladat (ha változtatunk egy adott feladaton, előbb törölni kell, majd újra létrehozni)
* DBMS\_SCHEDULER.run\_job('feladat\_név',az\_adott\_session\_használata);
  + meghív, és aktivál egy feladatot
* DBMS\_SCHEDULER.set\_job\_argument\_value('feladat\_név',1,'arg1');
  + egy létező feladatnak ad argumentumot, a második szám paraméter az argumentum pozícióját határozza meg, így lehet több argumentumot is megadni
* DBMS\_SCHEDULER.stop\_job('feladat\_név',erőltetett)
  + Egy adott nevű feladatot állít le, ha a második paraméter TRUE, akkor kényszeríti a leállást.

##### Nehézségek a csomag használata közben

* Alapból a hibavisszajelzés nem elég részletes (később bemutatok egy lekérdezést mellyel többek között a pontos hiba is kiolvasható)
* A csomag a feladat futása idejére a teljes PATH környezeti változót üríti az aktuális folyamat számára. Mindent abszolút elérési úttal kellett megadni.

##### A hibakeresést nagyban megkönnyítő lekérdezések:

* SELECT \* FROM all\_scheduler\_job\_run\_details where job\_name='feladat\_név';
* SELECT \* FROM DBA\_SCHEDULER\_JOBS WHERE JOB\_NAME = 'feladat\_név';
* select \* from V$SCHEDULER\_RUNNING\_JOBS;

##### A Felhasznált PL/SQL kódok a feladat ütemezésére

Szükséges jogok megadása SEMA1-nek:

GRANT MANAGE SCHEDULER TO SEMA1;

GRANT SELECT ON DBA\_SCHEDULER\_JOBS TO SEMA1;

###### Feladat elkészítése:

BEGIN

DBMS\_SCHEDULER.create\_job (job\_name => 'CTRLJOB',

job\_type => 'EXECUTABLE',

job\_action => 'c:\windows\system32\cmd.exe /c C:\LJava\control.bat',

enabled => FALSE,

comments => 'runs controll.bat'

);

END;

###### A feladat futtatásának elindítása:

BEGIN

DBMS\_SCHEDULER.run\_job ('CTRLJOB', TRUE);

END;

###### A végleges kód, ami egy before heather folyamatban kap helyet:

DECLARE

jnum number(1) := 0;

rnum number(1) := 0;

jobname varchar2(15) := 'CTRLJOB\_S1';

credname varchar2(15) := 'SYS.SDBA\_CRED';

BEGIN

SELECT count(\*)

INTO rnum

FROM V$SCHEDULER\_RUNNING\_JOBS;

if rnum>0 then

DBMS\_SCHEDULER.stop\_job(jobname,true);

end if;

SELECT count(\*)

INTO jnum

FROM DBA\_SCHEDULER\_JOBS

WHERE JOB\_NAME = jobname;

if jnum>0 then

DBMS\_SCHEDULER.drop\_job(jobname);

end if;

--,credential\_name => 'SYS.SDBA\_CRED'

--'c:\windows\system32\cmd.exe /c C:\LJava\control.bat'

DBMS\_SCHEDULER.create\_job (job\_name => jobname,

job\_type => 'EXECUTABLE',

job\_action => 'c:\windows\system32\cmd.exe /c C:\LJava\control.bat',

enabled => FALSE,

comments => 'runs controll.bat'

--,NUMBER\_OF\_ARGUMENTS => 2

);

--DBMS\_SCHEDULER.set\_job\_argument\_value(jobname,1,'/c');

--DBMS\_SCHEDULER.set\_job\_argument\_value(jobname,2,'C:\LJava\control.bat');

DBMS\_SCHEDULER.set\_attribute(jobname, 'credential\_name', credname);

DBMS\_SCHEDULER.set\_attribute(jobname, 'destination', 'marge.localdomain:61850');

DBMS\_SCHEDULER.run\_job (jobname, FALSE);

END;

###### Automatikusan másodpercenként lefutó verzió:

#### A control.bat fájl tartalma

"C:\Program Files\Java\jre7\bin\java" -jar C:\LJava\AparapiOpenCLInfo.jar

Jól látható, hogy a Java JVM elérési útját külön meg kellett adni, a PATH változó hiánya miatt. A macskaköröm az elérési útban lévő szünet karakter miatt volt szükséges.

#### Az Aparapi Device osztályának fontosabb statikus metódusai

* best()
  + Egy olyan Device osztállyal tér vissza, mely tárolja a környezet által legjobbnak ítélt eszköz paramétereit.
* firstGPU()
  + Egy olyan Device osztállyal tér vissza, mely tárolja a környezet által felismert első GPU-t, jelenleg a környezet még nem képes több eszközt is megtalálni
* firstCPU()
  + Egy olyan Device osztállyal tér vissza, mely tárolja a környezet által felismert első CPU-t, jelenleg a környezet még nem képes több eszközt is megtalálni

A Device osztály számos tulajdonságát tárolja az adott osztálynak, én egyszerűen csak az osztály toString() metódusát hívtam, mely kiírja ezeket.

Megjegyzendő, hogy amennyiben nincs megfelelően feltelepítve a környezetben az OpenCL driver, a fent említett metódusok, üres, kitöltetlen Device példányokkal térnek vissza.

#### A Kernel osztály főbb jellemzői

Amennyiben olyan kódot szándékozunk elkészíteni, amely esetleg egy gyorsító eszközön fut, akkor a Kernel absztrakt osztályból kell származtatnunk egy osztályt, mely megvalósítja az egyetlen run() metódust, amit a Kernel absztraktnak jelölt. A Java absztrakt Thread osztályának a használata is így zajlik.

Amire még oda kell figyelni, hogy azokat az adattömböket, amiket az eszköz fog írni illetve olvasni, lehetőleg ne módosítsuk a kernel futása közben, ezért ajánlatos final kulcsszóval való megjelölésük, ami kikényszeríti, hogy a Java kódunkban csak a konstruktor írhassa a változókat.

Megjegyzendő, hogy a Kernel alapbeállításként, ha mi másképp nem rendelkezünk a legjobbnak ítélt eszközön fut, amennyiben nincs megfelelően feltelepítve a környezetben az OpenCL driver, a Kernel automatikusan a Java Thread Poolban fut le.

##### A Kernel osztály főbb metódusai:

* execute(méret)
  + Egy megadott számnyi elemet dolgoz fel (én ezt használtam, és a bemenő tömb méretét adtam meg méretként)
* execute(méret, kihagyás)
  + Elképzelhető, hogy a tömb mérete túl nagy az eszköz számára, így megoldható, hogy több részletben dolgozzuk fel az adatot.
* execute(range)
  + Egy Range típusú objektumot vár
* getConversionTime()
  + Az a konvertálási idő, amire a JVM-nek szüksége volt, hogy a Java byte kódból OpenCL kódot készítsen.
* getExecutionTime()
  + A kernel futási ideje.
* getAccumulatedExecutionTime()
  + Összesített futási idő.

#### A Range osztály főbb jellemzői

Ha az egyszerű egydimenziós tömbnél komolyabb feldolgozást kívánunk végrehajtani, vagy meg szeretnénk határozni, hogy melyik eszközön fusson le a Kernel, akkor a Range osztályra van szükség.

##### Egy új példány inicializálásának módjai:

* new Range(Device.best(),2);
  + Az egyetlen konstruktora vár egy eszközt, és egy dimenzió számot.
* Range.create(width, height);
  + A creat gyártófüggvény, mely egy szélességet és egy magasságot vár.
  + Ez a metódus számos túlterhelt változatban létezik.

##### Egy példakód a Rage használatára

Device device = Device.firstGPU();

Kernel kernel = new Kernel(){

@Override public void run(){

// input[] feldolgozása;

}

};

range = device.createRange2D(1024, 1024);

kernel.execute(range);

Megfigyelhető, hogy maga a Device is képes visszatérni egy rá mutató Range objektummal.

#### Az Aparapit felhasználó kód fontosabb részei

Alább a négyzetre emelést végrehajtó kernel látható.

public class SquareKernel extends Kernel implements IRunInfo {

final private float in[];

final private float out[];

public SquareKernel(float in[]) {

this.in = in;

out = new float[in.length];

}

public float[] getIn() {

return in;

}

public float[] getOut() {

return out;

}

@Override

public void run() {

int i = getGlobalId();

out[i] = in[i] \* in[i];

}

@Override

public String getRunMode() {

EXECUTION\_MODE mo = this.getExecutionMode();

switch (mo) {

case CPU:

return "CPU";

case GPU:

return "GPU";

case JTP:

return "JTP: Java Thread Pool";

case NONE:

default:

return "Nincs információ a futásról";

}

}

public String InputAsString() {

return FloatArrayAsString(in);

}

public String OutputAsString() {

return FloatArrayAsString(out);

}

private String FloatArrayAsString(float[] input) {

if (input == null || input.length == 0) {

return "";

}

Float first = new Float(input[0]);

String ret = first.toString();

for (int i = 1; i < in.length; i++) {

Float fi = new Float(input[i]);

ret += ", " + fi.toString();

}

return ret;

}

@Override

public String getRunTask() {

return "Négyzetre emelés: " + in.length + " float típusú számon";

}

public static String GetTestRunHTML(IRunInfo testrun) {

String html = "";

html += "<div>";

html += "<h2>Aparapi kernel feladata: " + testrun.getRunTask() + "</h2>";

html += "<p><h3>Futtató eszközről</h3>" + testrun.getRunMode() + "</p>";

html += "</div>";

html += String.format("%n");

return html;

}

}

Amint látható ez az osztály felelős a

#### A végeredmény fájlt generáló Java osztály

Az létrehozni kívánt fájl karakterkódolására az ISO-8859-2-t (közép európai latin) választottam, ez azért szükséges, mert a PL/SQL fájlolvasó függvénye jobban boldogul az ASCII alapú karakterkódolással, és ez az a karaktertábla, amely képes a magyar karaktereket is megjeleníteni.

##### Az OCLFileWriter osztály

public class OCLFileWriter implements Closeable {

Charset ASCII\_HU = Charset.forName("ISO-8859-2");

OutputStream os = null;

OutputStreamWriter ow = null;

public OCLFileWriter(String path) throws IOException {

File f = new File(path);

if (!f.exists()) {

f.createNewFile();

}

os = new FileOutputStream(f);

ow = new OutputStreamWriter(os, ASCII\_HU);

}

public void append(String str) throws IOException {

ow.append(str);

}

public void write(String str) throws IOException {

ow.write(str);

}

@Override

public void close() throws IOException {

if (ow != null) {

ow.close();

}

if (os != null) {

os.close();

}

}

}

##### Az OCLFileWriter osztályt felhasználó writeFile metódus.

public static void writeFile(String str){

OCLFileWriter infoFileWriter = null;

String infoFilePath = "C:\\LJava\\OCLInfo.txt";

try {

infoFileWriter = new OCLFileWriter(infoFilePath);

infoFileWriter.write(str);

} catch (IOException e) {

System.out.println("e: " + e.getMessage());

} finally {

if (infoFileWriter != null) {

try {

infoFileWriter.close();

} catch (IOException ex) {

System.out.println("Problémás bezárás");

System.out.println("ex: " + ex.getMessage());

}

}

}

}

#### Fájl olvasása PL/SQL, UTL\_FILE csomag segítségével

##### A csomag fő függvényei

* UTL\_FILE.FILE\_TYPE;
  + Egy fájlkezelő változótípus
* UTL\_FILE.FOPEN('LJAVA', my\_filename, 'r',32767);
  + Megnyit olvasásra egy fájlt, és visszatér egy UTL\_FILE.FILE\_TYPE típusú kezelővel
  + Paraméterei:
    1. Egy CREATE DIRECTORY paranccsal elkészített könyvtár objektum neve
    2. Egy konkrét fájl szöveges neve
    3. A megnyitás módjának jele (r - olvasás; w - írás)
    4. Egyszerre mekkora méretet enged kiolvasni. (a 32767 a maximális kiolvasási buffer méret, egyben a VARCHAR2 típus maximális hossza is)
* UTL\_FILE.GET\_LINE(data\_file, data\_line);
  + Az első paraméterként megkapott, megnyitott fájlból, a második paraméterként átadott fájlba egy maximum ment buffer méretnyi hosszú sort (ha a sor túl hosszú kivételt vált ki)
* WHEN no\_data\_found THEN
  + A fájl végéhez érve kiváltott kivétel.
* UTL\_FILE.FCLOSE(data\_file);
  + A fájlt bezáró függvény, használat után a fájlt be kell zárni, hogy felszabadúljon a lefoglalt erőforrás.

##### Végleges PL/SQL kód, ami az elkészült fájlt olvassa.

DECLARE

data\_line VARCHAR2(32766); -- Data line read from input file

data\_file UTL\_FILE.FILE\_TYPE; -- Data file handle

my\_filename VARCHAR2(50) := 'OCLInfo.txt';

veg\_str VARCHAR2(32766) := '';

BEGIN

execute immediate 'CREATE OR replace directory LJAVA as ''c:\LJava''';

data\_file := UTL\_FILE.FOPEN('LJAVA', my\_filename, 'r',32767);

LOOP

UTL\_FILE.GET\_LINE(data\_file, data\_line);

veg\_str := veg\_str||' '||data\_line;

END LOOP;

EXCEPTION

WHEN no\_data\_found THEN

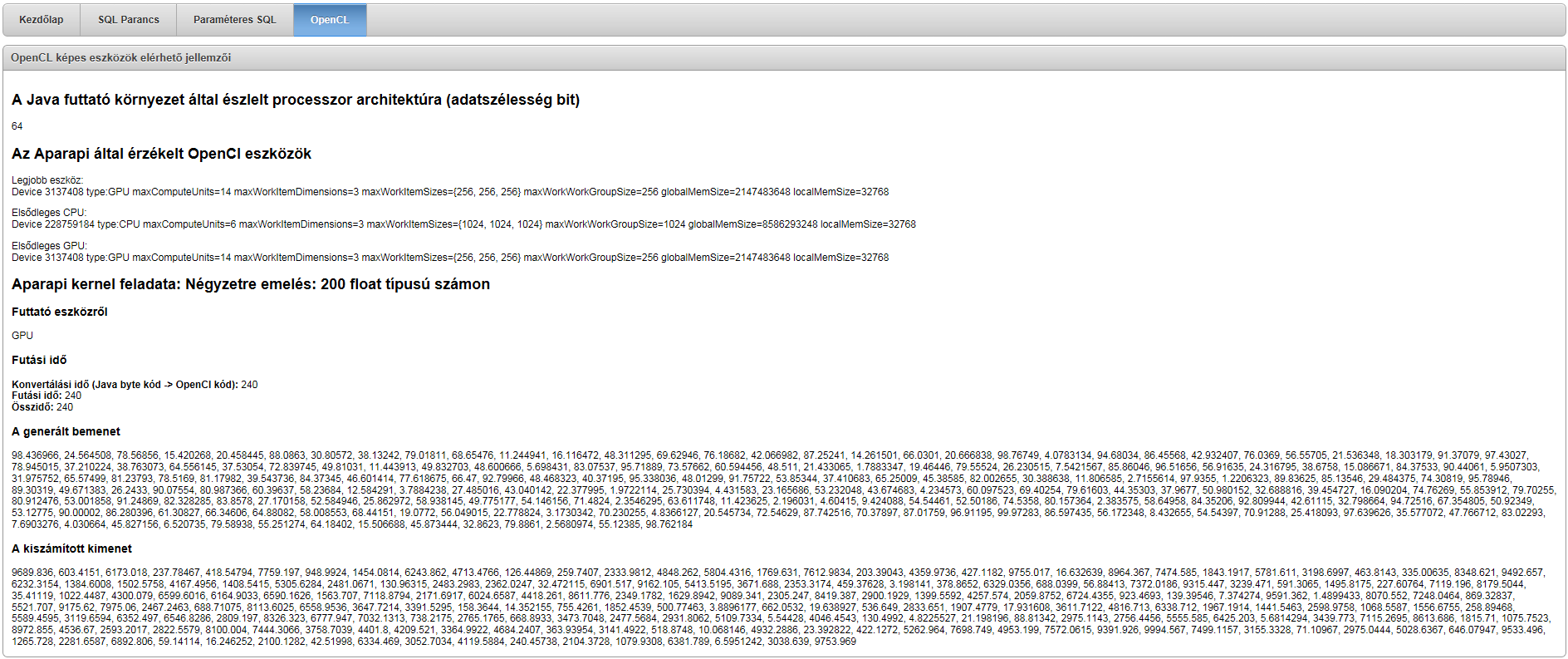
UTL\_FILE.FCLOSE(data\_file);

htp.p(veg\_str);

END;

A fájl végére érve, az eddig összeállított szöveget, a htp.p() segítségével írja ki, egy régióba.

### Futási eredménye



# Irodalomjegyzék

A feladat elkészítése során használt források:

* Különböző internetes keresések kimeneteleként megtalált internetes segédanyagokat használtam.
* Az internetes erőforrások közül leggyakrabban az Oracle hivatalos dokumentációit tároló <http://docs.oracle.com> oldalainak vettem hasznát.
* Az Adatbázis Használat 2 órai segédleteket, mint az APEX könyv, illetve a példaprogramok is hasznomra váltak.
* Rátaláltam egy 248 oldalas Oracle leírásra, mely végig java eljárások meghívásával foglalkozik, ezt, és egy másik hasonló témában íródott könyvrészletet pdf formában elérhetővé tettem a CD mellékleten. (java/ irodalom könyvtár)

# CD Melléklet