Projekt dokumentációja

**07.01**

Végighallgattuk a bevezetést, felmértük a kezdeti érdeklődési köröket. Ismerkedtünk az SAP-HANA Cloud Foundry-val.

**07.02**

Sikerült regisztrálni az SAP-HANA Cloud Foundry-ba, de egy egyszerű üres adatbázist nem sikerült buildelnem. Próbálkoztam egy node.js modul létrehozásával, de az se sikerült. Végül egy egyszerű „Hello World” HTML5 modult sikerült csinálni.

Ennek a tutorial-sorozatnak a végigolvasása, általános fogalmak megismerése (MTA, microservices, oszlop tömörítés, in memory).

<https://blogs.sap.com/2017/09/04/xs-advanced-for-not-so-dummies/>

<https://blogs.sap.com/2017/09/05/xs-advanced-for-not-so-dummies-pt-2-multi-target-applications/>

<https://blogs.sap.com/2017/09/22/xs-advanced-for-not-so-dummies-pt-3-microservices/>

<https://blogs.sap.com/2018/02/16/xs-advanced-for-not-so-dummies-routing/>

**07.03**

Kapott e-mail linkjeinek végigolvasása.

XSA dokumentáció

<https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/d8226e641a124b629b0e8f7c111cd1ae.html>

CDS dokumentáció

<https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/d8226e641a124b629b0e8f7c111cd1ae.html>

PAL dokumentáció

<https://help.sap.com/viewer/2cfbc5cf2bc14f028cfbe2a2bba60a50/2.0.03/en-US/c9eeed704f3f4ec39441434db8a874ad.html>

Mivel a szerver hibás működése miatt a kapott tutorialok kipróbálása sikertelen volt, más ismeretek megszerzése felé indultam el. Láttam, hogy a javascript egy nagyon fontos eleme ennek a projectnek, így az alábbi tutorialok segítségével megpróbáltam tapasztalatot szerezni js kód írásában.

<https://www.youtube.com/watch?v=PkZNo7MFNFg>

Adatvizualizáció d3.js könyvtár használatával. Ezután js-be kézzel bevitt kódokat már tudtam vizualizálni.

<https://www.youtube.com/watch?v=nzshmMlOuwI>

**07.04**

Kaptunk egy csv fájlt, végighallgattuk a teendőket vele. Sajnos szerver oldalú hiba miatt a több óra munkával semmit sem értünk el. Miután ezt a tevékenységet feladtam, elkezdtem Thomas Jung SAP-HANA developer youtube tutorial videóit nézni. Ezekben levezeti, hogy miért van szükség erre a technológiára, milyen hardver fejlődések tették elérhetővé, és a fontosabb fogalmakat ábrákkal is vizualizálta. Sajnos, amikor gyakorlati részre ért, a nem működő szerverek miatt nem lehetett kipróbálni.

**07.05**

Megkaptuk az ELTE szervereket, és létrehoztuk az első adatbázist. Elkezdtük az adatok hibájával az ismerkedést.

Adatbázis létrehozása

1. Az egyetemi lokális szerverre bejelentkezés: <https://oktnb132.inf.elte.hu:53075/watt/index.html>; első bejelentkezés után módosítjuk a jelszavunkat.
2. Új Multi Target Applicationt hozunk létre.
3. Jobb klikk a munka mappánkra majd „New Project From Template”.
4. Ezen belül jobb klikkel „New SAP HANA database module”.
5. Így keletkezik egy src mappa, amin jobb klikkel: „New HDB CDS Artifact”.
6. Ezt megnyitva egy kontextust látunk, és azon belül hozzunk létre egy entity-t, mindkettőnek tetszőleges nevet adhatunk.
7. Entity-t megnyitva létrehozzuk az összes fejlécet, és mindegyiknek beállítjuk a típusát. A kapott csv fájl fejléceit hoztuk létre.
8. Majd buildeljük a projektet.
9. Database explorerben létrehozunk egy új adatbázist „Add a database to the Database Explorer” (plusz ikonnal jelölt) lehetőségre, és a felugró listából kiválasztjuk a MTA projectünk nevét.
10. Ezután a table fülön belül megtaláljuk a létrehozott adatbázis modellünket, amire jobb gombbal kattintva felugrik az „Import Data” lehetőség.
11. A felugró menüben kiválasztjuk a csv fájlunkat.
12. Ellenőrizzük, hogy az attribútumokat helyes felismerte-e, ha nem vagy hiányosak, akkor kézzel beállítjuk.
13. Végül a majdnem sikeres import után a két teljesen hibás sort az SQL console-ba írt kóddal töröltem.

**07.08**

Modulok létrehozását tanultam.

<https://developers.sap.com/tutorials/xsa-connecting-webide.html>

<https://developers.sap.com/tutorials/xsa-html5-module.html>

<https://developers.sap.com/tutorials/xsa-hdi-module.html>

<https://developers.sap.com/tutorials/xsa-xsjs-xsodata.html>

HDI container létrehozása, adatokkal feltöltése. Végül a front-end fejlesztését Angularban csináltam, így később nem kellett használni html5 modult. Megpróbáltam odata-t létrehozni, de jogosultságok miatt az nem sikerült. Továbbiakban az adathalamazt tisztítottam SQL kód segítségével, és létrehoztam további táblákat, hogy a felesleges ismétlődéseket elkerüljem.

Adat tisztítása és normalizálása

1. Az összes segéd-táblánknak először létrehozunk egy entity-t. Buildelés után ez az entity megjelenik a Database Explorer Tables menüpontja alatt.
2. SQL kód segítségével feltöltjük adatokkal ezeket a táblákat.
3. A táblázatok létrehozásában a felesleges információk ismétlődésének minimalizálására törekedtem.
4. Következő segédtáblákat hoztam létre:
   1. Eredeti CEGEKTABLA tábla minden attribútummal
   2. TARSTIPUS(TARS\_TIPUS\_MEGNEV, TARS\_TIPUS\_KOD)
   3. GAZD\_FORM(GAZD\_FORM, GAZD\_FORM\_KOD)
   4. CEGALL(CEGALL, CEGALL\_KOD)
   5. NEMGAZD\_AG(NEMGAZD\_AG\_MEGNEV, NEMGAZD\_AG\_KOD)
   6. NEMGAZD\_AGAZATNEMGAZD\_AGAZAT\_MEGNEV, NEMGAZD\_AGAZAT\_KOD)
   7. NEMGAZD\_SZAKAGAZAT(NEMGAZD\_SZAKAGAZAT\_MEGNEV, NEMGAZD\_SZAKAGAZAT\_KOD)
   8. ORSZAG(ORSZAG, ORSZAG\_KOD)
   9. REGIO(REGIO, REGIO\_KOD)
   10. MEGYE(MEGYE, MEGYE\_KOD)
   11. TELEPULES(TELEPULES, TELEPULES\_KOD)
   12. NCEGEKTABLA(TARS\_ROV\_NEV, TARS\_HOSZ\_NEV, CIM\_EGYBEN, TARS\_TIPUS\_KOD, ADOSZAM, GAZD\_FORM\_KOD, CEGALL\_KOD, NEMGAZD\_AG\_KOD, NEMGAZD\_AGAZAT\_KOD, NEMGAZD\_SZAKAGAZAT\_KOD, JEGYZ\_TOKE\_ERT\_HUF, ORSZAG\_KOD, REGIO\_KOD, MEGYE\_KOD, TELEPULES\_KOD, ASZ\_EVE)

07.09

Sikerült Odatát létrehozni. Utána az Odatának a lekérdezésével js kódban próbálkoztam, de ez nem sikerült.

Odata létrehozása

1. Az eddig elkészített MTA-ban létrehozunk egy „New Node.js Module”-t.
2. „Enable XSJS support” lehetőséget ki kell pipálni.
3. Ebben az állományban lib mappájában készítünk egy új fájlt, aminek a kiterjesztése xsodata.
4. xsodata állományba a következőt írtam:

service {

"CEGEKTABLA" as "tablazatodata" keys generate local "ID";

"kalkulacios" as "kalkulaciosodata" keys generate local "ID";

"CMEGYE" as "cmegyeodata";

}

1. Ezzel a beállítással elérhető url-ben az eredeti tábla, a kalkulációs nézet, és egy egyedi lekérdezés táblám, „CMEGYE”, amiben megyénkként a cégek száma van.
2. Server.js fájl-nak a redirectUrl utáni részt át kell írni „/<név>.xsodata”-ra.
3. A buildelés után kapunk egy URL linket, amin keresztül elérhetjük a service-ben szereplő táblákat.

07.10

Calculation View létrehozása

1. Az első létrehozott mappának a .hdiconfig kiterjesztésű fájlában a plugin\_version-t átírjuk 2.0.30.0-ra.
2. A src-mappára jobb klikkel kattintva „New Calculation View”.
3. Create Jointtal létrehozunk egy joint, amit át is nevezhetünk.
4. Táblázatok megadása forrásként „Add Data Source” funkcióval.
5. Kétszer kattintunk a joinra, Join Definition fülben meghatározzuk az inner joinokat az adattagok összekötésével.
6. Mapping fülben mindegyik adattagok behúzzuk, de csak egyszer.
7. Bezárjuk a szerkesztőt, dupla kattintás az Aggregationra.
8. Mapping fülben is minden adatot behúzunk.
9. Build után már működni fog a Data Preview.

07.11-12

Végül az Angular sikeres telepítése után és egy hallgató kollégától kapott kezdetleges kód segítségével elkezdtem a front-end fejlesztését és a dokumentáció megírását.

Front-End, Angular

1. Angular telepítése console-ból npm install -g @angular/cli paranccsal
2. Ha ez a parancs nem működik, telepíteni kell a node.js-t
3. Az adatok vizualizácójához töltsük le a canvasjs könyvtárat.
4. app.component.ts állományba importáljuk a canvasjs állományt
5. services mappán belül, odata.service.ts-t hozunk létre, és ezzel a kóddal beállítjuk az odata url-t.

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Injectable()

export class OdataService {

constructor(private http: HttpClient) { }

configUrl = '/hana/myodata.xsodata/';

getData(params)

{

    return this.http.get(this.configUrl + params);

}

}

1. Odatáról az adatok megszerzése:

export class AppComponent implements OnInit {

ezen belül:

this.odata.getData('tablazatodata?$top=1000&$orderby=JEGYZ\_TOKE\_ERT\_HUF+desc&distinct=true').subscribe((res :any) => {

this.data = res.d.results;

show\_graph(this.data,this.data2);

});

majd ezután:

function show\_graph(db,db2) {

var asd1 = [];

var asd2 = [];

for (var i = 0; i < db.length; ++i) {

asd1.push({ label: db[i].TARS\_ROV\_NEV, y: db[i].JEGYZ\_TOKE\_ERT\_HUF / 1000000 })

}

show\_graph függvény végén megírjuk a kirajzolást a canvasjs állományban található példák alapján.

07.15 – 07.16

A projektünk dokumentációját véglegesítettük. Regisztráltunk GitHubra. Előkészítettük az állományt, hogy a feltöltésnek megfelelő formátuma legyen. Végül feltöltöttük GitHubra. Megbeszélésen véglegesítettük, hogy a project milyen részén fogunk dolgozni.

07.17-18

Az általam választott feladatkör az adatbányászat volt az SAP HANA környezetben. Ezért elkezdtem ennek a dokumentációját olvasni. Milyen kategóriái vannak az adatbányászatnak. Mire jó az adatbányászat, az egész folyamatot hogyan kell elképzelni.

Nagyon rövid összefoglaló az olvasottakról:

* Cluster Analysis: Hasonló adatok csoportosítása.
* Classification: Adatok közti kapcsolat vizsgálata.
* Regression analysis: Függöség, függetlenség vizsgálata.
* Association: Nagyobb adathalmazok közti kapcsolat vizsgálata.
* Time Series: Adatok vizsgálata időrendben.
* Data preprocessing: Adatgyűjtés után, adatok tisztítása, normalizálása.
* Social Network: Szociális háló vizsgálata.
* Recommender system: Reklámok elhelyezése, megjóslása annak, hogy adott terméket kik vennének.
* Miscellaneous: Egy képből másik kép készítése, képek vizsgálata.

07.19

SAP Hana dokumentációt olvastam, és megpróbáltam meghívni az első PAL függvényemet. Egész napos próbálkozás és állítgatások után, végül nem vezetett eredményre ez a folyamat.

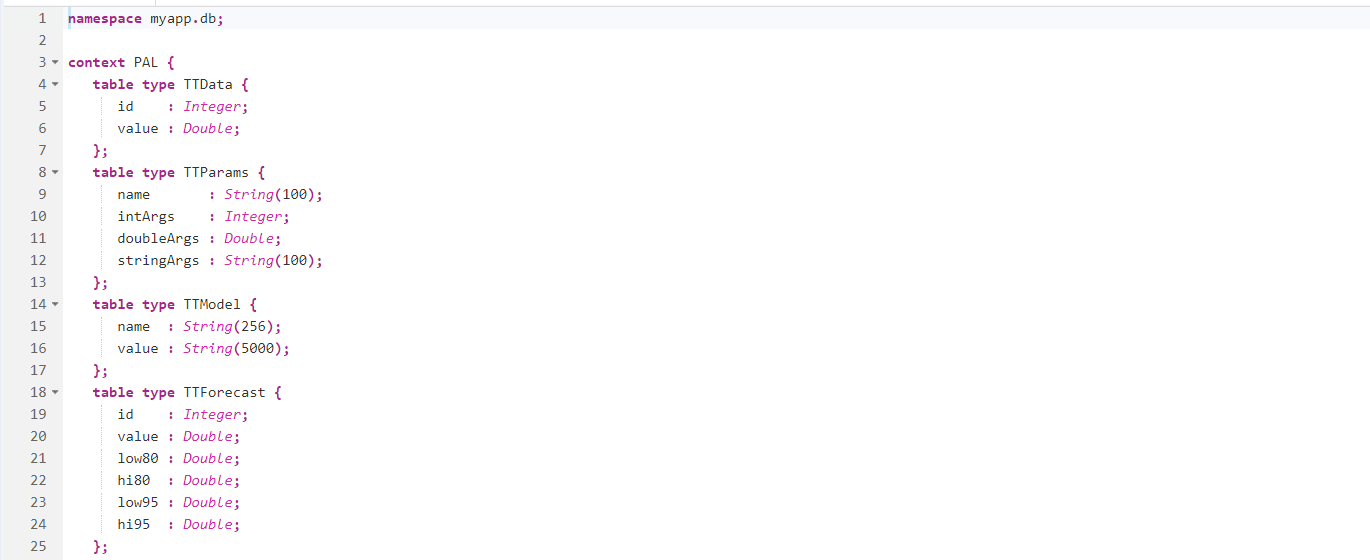
07.22-23-24

Függvények meghívását feladtam. PAL-ban található függvényekkel ismerkedtem. Elolvastam, hogy milyen táblák kellenek egy ilyen függvény meghívásához (INPUT, OUTPUT, PARAMETER). Kaptunk egy tutorialt arról, hogy hogyan kellene meghívni ezeket az eljárásokat. A tutorialt végigcsináltam, létrehoztam az összes állományt, definíciókat, deklarációkat, de script szerver hiánya miatt ez a munka nem vezetett eredményre. Nem tudtuk meghívni a létrehozott eljárásokat, csak hibaüzenetet kaptam.

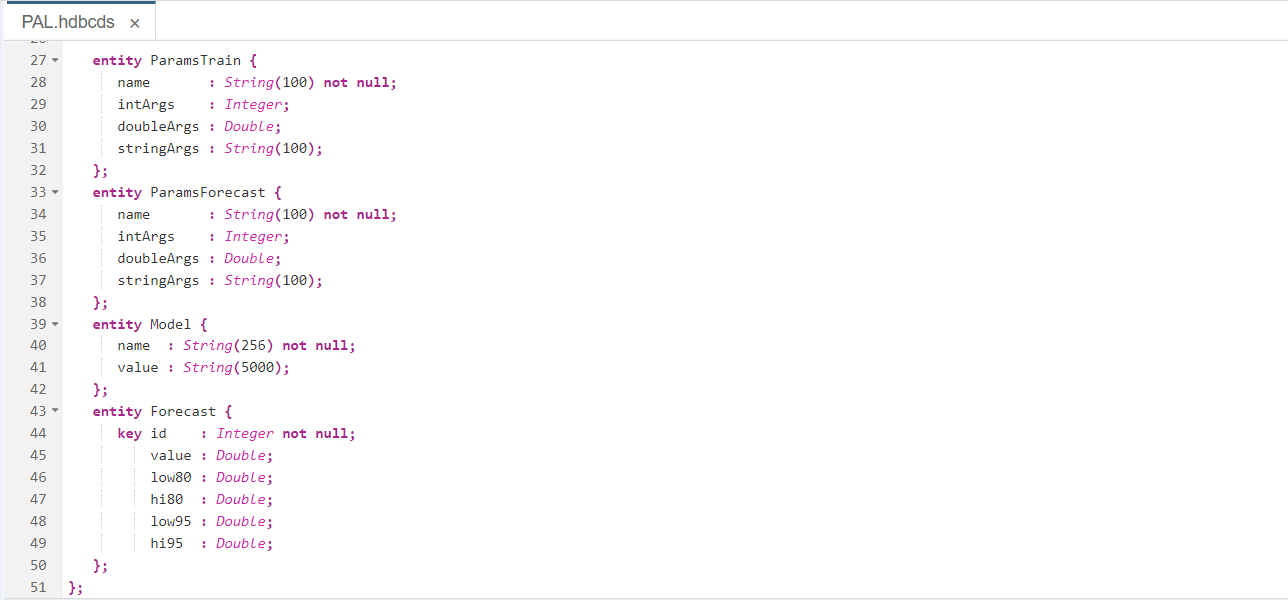
<https://www.youtube.com/watch?v=xzeG3kByKEs&list=PLkzo92owKnVwL3AWaWVbFVrfErKkMY02a&index=68>

Egy procedúra meghívása:

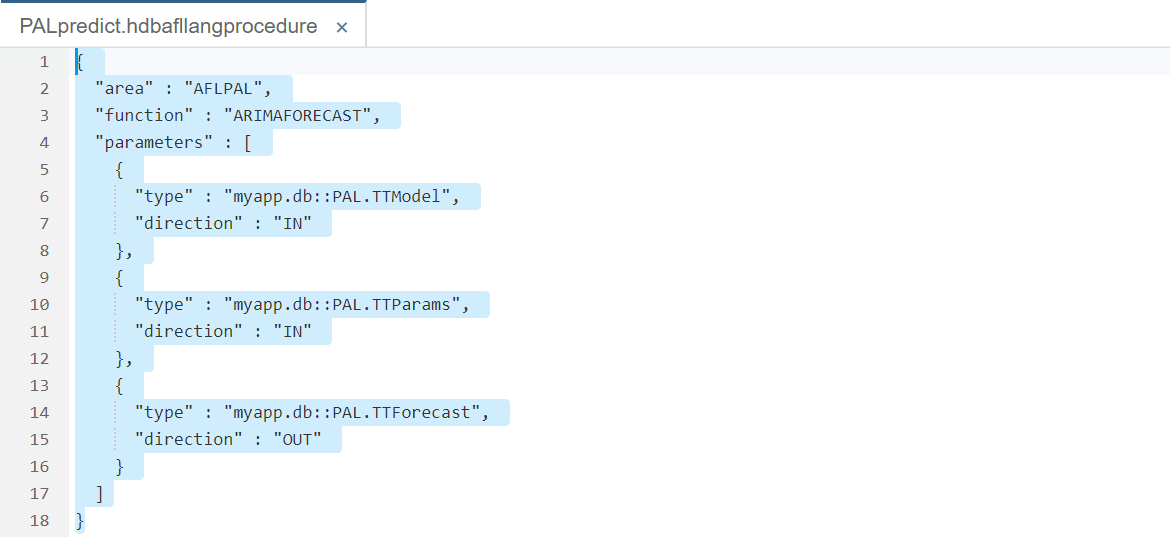
* Létre kell hozni a procedúrához szükséges táblák definícióját és deklarációját hdbcds állományban.
* A definíciója például:



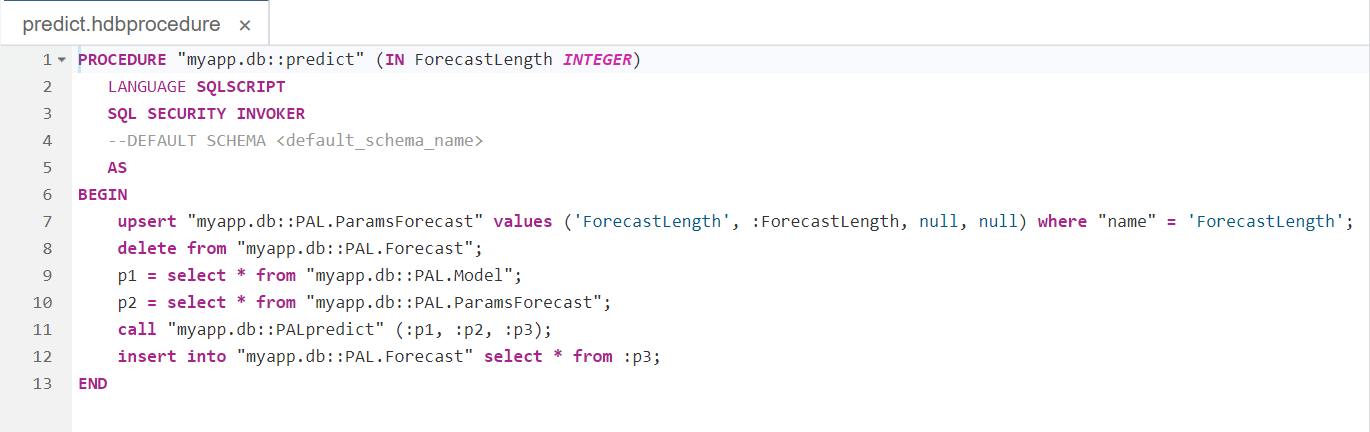
* A deklaráció:



* Meg kell írni a procedúra definícióját egy hdbflangprocedure állományban, amiben a már létrehozott tabletype-ok segítségével megadjuk, hogy a procedúra melyik tábla típusokat használja.



* Ezután sql kódban részletezzük, hogy az eljárás melyik táblákat használja és hova helyezze el az eredményeket.



07.25-26

Megkaptuk a függvények meghívásához szükséges script szervert. A tutorialban szereplő példákat sikeresen meghívtuk. A tutorialban az AUTO ARIMA eljárást mutatták be, és ezt csináltam végig. Létrehoztam a table type-okat és az entity-ket az adatnak, a paraméternek és az eredménynek. Ezután létrehoztam a train és a predict procedúrát. A végeredményt sikerült lefuttatni, csak kis mértékben tért el a videóban láthatótól.

07.29-30

Megpróbáltam további eljárásokat lefuttatni mind a tutorialban szereplő adatbázisra, mind az eredeti országos adattáblára. A lefuttatáshoz meg kellett oldani, hogy legyen ID a táblázatban. Sokat kerestem erre egy gyors megoldást, de végül ezt egy sql procedúrával oldottam meg. A procedúra futtatása előtt beszúrtam a táblába egy üres oszlopot, majd a procedúra ezt az oszlopot soronként feltöltötte a megfelelő számmal.

Nem sikerült lefuttatni a procedúrákat se a tutorialos, sem az eredeti táblázatra. A pythonos csapatnak végül sikerült lefuttatni egy procedúrát az eredeti országos adatokra, de az egészet egy nagy klaszterbe rakta. Ez érthető is, hiszen azok az adatok nevekből és a nevekhez társított rövidítésekből és kódokból állnak.

Az interneten talált dokumentációkat és videókat nem tudtam használni, mivel SAP HANA studiot használtak vagy SQL kódban írták meg az egészet, de erre privilege error-t kaptan minden esetben.

A maradék időben a dokumentáció írásával töltöttem.

07.31

Délelőtt a paraméter táblák megadásának módját próbáltam megérteni, mert ezt szinte sehol se részletezik. Végül rájöttem, hogy azért, mert ennek a táblának típusa mindig ugyanaz, mindegy, milyen eljárást futtatok.

A paraméter tábla típusa:

table type TTParams {

name : String(100);

intArgs : Integer;

doubleArgs : Double;

stringArgs : String(100);

};

Tanári megbeszélés után azt a feladatot kaptuk, hogy keressünk az interneten feldolgozott adatokat és PAL használatával dolgozzuk fel mi is azt, majd vessük össze az eredményt.

08.01-02

Feldolgozott adatokat kerestem, de sajnos sikertelenül. Kétféle probléma volt: az egyik, hogy a feldogozott adat és a feldolgozás módja gyakran nagyon „egyszerű” volt, PAL használata nélkül megkaphatók voltak az eredmények. Például, hogy melyik hónapban milyen színű cipőt vettek a legtöbben. A másik meg hogy annyira bonyolult volt a kiindulás, hogy nem tudtuk megérteni, például DNS és egyéb egészségügyi eredményeket találtam, de ott elindulni nem tudtam a megfelelő szakmai tudás hiánya miatt.

Somlóvári Sára Lucával innentől kezdve közösen dolgoztunk.

A hatodik hét elején a témavezető tanároktól kaptunk egy adatbázist, mely különböző szenzorok által mért időjárás- és légszennyezés-adatokat tartalmazott (hőmérséklet, nyomás stb). Ezen kellett különböző adatbányász eljárásokat végrehajtanunk, a PAL könyvtár használatával.

Az első problémát az jelentette, hogy az oszlopok száma ellehetetlenítette egy olyan tábla manuális létrehozását, amelybe az összes adatot beolvashatnánk. Ezért az adatbázis első sorát, amely egyben tartalmazta az összes oszlopnevet (vesszővel elválasztva), kimásoltuk egy txt fájlba. Ezt követően írtunk egy programot, amely beolvassa ezt a txt fájlt, és egy másik, kimeneti txt fájlba kiírja az adattagokat soronként.

(Tehát ezt a szöveget: „*UTC\_time,3\_temperature,3\_humidity,3\_pressure,3\_pm1,3\_pm25,3\_pm10,140\_temperature,140\_humidity,140\_pressure,140\_pm1,140\_pm25,140\_pm10,142\_temperature,142\_humidity,142\_pressure,142\_pm1,142\_pm25,142\_pm10, (…)*” átalakította egy ilyen formátumúvá:

„*UTC\_time*

*3\_temperature*

*3\_humidity*

*3\_pressure*

*3\_pm1*

*3\_pm25*

*3\_pm10*

*140\_temperature*

*140\_humidity*

*140\_pressure*

*(…)*”.)

Ezután azzal szembesültünk, hogy SAP-ban az oszlopnevek nem kezdődhetnek számmal, ezért írnunk kellett egy új programot, amely megcserélte a karakterek és számok sorrendjét (pl. „*3\_temperature*”-t átírta „*temperature\_3*”-ra). A második programot lefuttattuk a már sorokra tördelt txt fájlon és egy új txt állományba ugyan úgy sorokra bontva, de már megfordított névvel kiírta őket. Végül egy nagyon rövid harmadik programmal minden sor végére odaírtuk hogy „ : Integer;”, ez ahhoz kellett hogy minden oszlop után szerepeljen a típus megnevezése. Ezután egy olyan output fájlt kaptunk, amelynek tartalmát SAP-ba beillesztve létre tudtuk hozni a kívánt táblát.

(*time\_UTC : Integer;*

*temperature\_3 : Integer;*

*humidity\_3 : Integer;*

*pressure\_3 : Integer;*

*pm1\_3 : Integer;*

*pm25\_3 : Integer;*

*pm10\_3 : Integer;*

*(…)*)

*Megjegyzés: a fent leírt kódokat, txt-ket a cpp.zip állomány tartalmazza.*

Miután a januári adatokat sikerült hiba nélkül beolvasni (a *bigDB* táblába), el kellett döntenünk, mely eljárásokat kívánjuk futtatni rajtuk. Választásunk az alábbi háromra esett:

1. Affinity Propagation (AP, klaszterező algoritmus)
2. Anomaly Detection (kiugró adatok kimutatása)
3. ARIMA (az idősor-elemzés egy gyakran használt függvénye).

*Megjegyzés: a továbbiakban említett fájlokat az sap.zip állomány tartalmazza.*

A fent említett három eljárás mindegyike megkövetel egy *Integer* típusú *ID* oszlopot, továbbá nem engedi meg, hogy az adatok között *NULL* is szerepeljen. Hogy ezeket a feltételeket teljesítsük, SQL parancsok segítségével kerestünk 5 szenzort, melyeknek legfeljebb egy sorában szerepelt ismeretlen adat, majd ezeknek azonos idejű, nem NULL adatait kimásoltuk egy új táblába (*fiveID*). Ezt a táblát előre úgy definiáltuk, hogy legyen egy azonosító oszlopa (*time\_stamp*); ezt beolvasáskor az egyik szenzor hőmérséklet-adataival töltöttük fel, majd egy eljárás (*identity.hdbprocedure*) segítségével 1-től kezdődő, egyesével növekvő egész számokkal.

Ezek után elkezdhettük az eljárások megírását. Mivel az AP-hez egy hallgató kollégától is segítséget kaptunk, ezzel kezdtük. Az *apfull* az egész *fiveID*-táblázatot nézi, és az alapján klaszterez, az *aptemp* csak az 5 hőmérséklet-oszlopot veszi figyelembe, míg az *ap211* csak a 211-es szenzor hőmérséklet-adataival számol.

Az *apfullproc*, amikor nem adtuk meg előre a klaszterek számát, 31 klasztert talált (az eredményt a *PAL.REAL\_CLUSTER* táblában találjuk), az *aptempproc* 40-et (*PAL.REAL\_TEMP\_CLUSTER*), az *ap211proc* pedig 24-et (*PAL.Cluster211*). Eredményeink helyadatokkal kiegészülve lennének igazán érdekesek, melyek segítségével megvizsgálhatnánk, az egymáshoz közeli települések tartoznak-e össze, vagy máshol kell keresni az összefüggéseket.

A képen képernyőkép, égbolt látható

Automatikusan generált leírás

**Ábra: részlet a klaszter-elemzés eredményéből**

A továbbiakban azt a döntést hoztuk meg, hogy egy szenzor (a 211-es) hőmérséklet-adataira fogunk koncentrálni, noha később az *ARIMÁ*-t futtattuk az egész *fiveID*-táblára is.

Először azonban az Anomaly Detection eljárást írtuk meg (*ano211.hdbafllangprocedure* és *ano211proc.hdbprocedure*). Ez 74 kiugró adatot talált a 743-ból (*PAL.Ano211*), vagyis nagyjából az adatok tizede mutatott a normálisnál nagyobb ingadozást. Az eljárás működését a futtatása után értettük meg igazán, amikor is világossá vált számunkra, hogy nem egy átlaghoz képest állapítja meg a függvény az outliereket, hanem a szomszédoktól való távolság alapján dönt. A szenzoros adatok között találtunk egy null adatot, amit tetszőlegesen kitöltöttünk a szomszédok alapján. Ha a szomszédokhoz képest 1 fokkal nagyobb vagy kisebb hőmérsékletet adtunk meg, akkor nem került az outlierek közé, ha 2 vagy nagyobb volt a különbség, akkor az eljárás megtalálta és belerakta az eredménytáblába. Megfigyeltük azt is, hogy az eljárás akár teljes csoportokat is outlierek közé rakott, ha a csoport ugyan azokat az értékeket tartalmazta, és a csoport szomszédai 2 értékkel kisebbek vagy nagyobbak voltak.

A képen képernyőkép látható

Automatikusan generált leírás

**Ábra: részlet az Anomaly Detection eredményéből**

Egy hosszabb távú project során érdemes lehet a kiugró adatok okát vizsgálni, illetve az anomáliákat a szomszédokhoz igazítani, hogy megnézhessük, miként változnak az egyéb eljárások eredményei.

Végül az ARIMA következett, melyet a fent említetteknek megfelelően kétféle módon futtattunk. Az *artrainfull*, *artrainfullproc*, *arpredfull* és *arpredfullproc* az egész *fiveID* táblázatot megkapja inputként, míg az *artrain211*, *artrain211proc*, *arpred211*, *arpred211proc* csak a 211-en szenzor hőmérséklet-adatait. Mindkét fajta eljárás eredményét a *PAL.TrainRes* és a *PAL.ForeRes* táblák tartalmazzák.

Az eredményeket látva jöttünk rá első eljárásunk hibájára, ugyanis nem vettük figyelembe, hogy az *ARIMA* egy végeredményt fog adni nekünk az input alapján, s ha ez az input vegyesen tartalmaz hőmérséklet, nyomás stb. adatokat, az előrejelzés értelmetlenné válik.

A második eljárás, amely csak a 211-es szenzor hőmérséklet-adatait vizsgálta, már pontosabb előrejelzést ad, kerekítve -2,91 és 0,45 fok közötti értéket ad eredményül.

A képen képernyőkép látható

Automatikusan generált leírás

**Ábra: a 211-es szenzorra futtatott ARIMA eredménye**

Összefoglalva tehát a PAL könyvtár három különböző területéről sikerült elsajátítanunk egy-egy eljárást. Egy hosszabb távú project célja lehetne az eredmények további elemzése, kiegészülve például helyadatokkal, s az eredmények alapján további, kifinomultabb adatbányászati eljárások írása és futtatása.