**Nagybeszámoló**

1. Minta applikáció készítése

A munkafolyamat során a következő linket használtam:

<https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/4f0a1567616541cbb7bd338fb1b64c35.html>

Ez egyszerű TinyWorld alkalmazás, ami ebben a formában kicsi és használhatatlan, de betanulásnak szerintem megfelelő. Ehhez hozzáadtam egy adatbázis, egy NodeJS és egy HTML5 modult a leírásnak megfelelően, ezek fordultak, és sikeresen futtattam is őket. A következő linkek ennek kibővítésére vonatkoznak, a kalkulációs nézet elkészítésekor akadtam el, egyrészt azért, mert a leírásnak megfelelő elkészítés egy „no measures defined” típusú hibát dobott, másrészt pedig azért, mert megkaptuk a következő feladatunkat.

1. A társaságok alkalmazásban való tevékenykedés

Az alkalmazás röviden arról szól, hogy a csv formátumban kapott társaságadatokhoz készítsünk egy adatbázist a HANA rendszerben, készítsünk hozzá kalkulációs nézetet, majd egy külső alkalmazás segítségével OData-ról elérjük a társaságadatokat, azok címadataiból geokódot állítsunk elő, végül ezt térképen jelenítsük meg.

A fejlesztésben az én általam elvégzett feladatok az adatbázis sémájának kialakítása, az adatok importálása és tisztítása volt, ugyanis az előzetes megbeszélésen ezt kaptam feladatként, ezen felül számos alkalommal nyújtottam kisebb segítséget a geokódot előállító részében.

Az adattáblát először importáltam a HANA rendszerbe, ahol az adatbázis számos hibájára bukkantam rá. Először felfedeztem két olyan sort, amely nem felelt meg a kialakított sémának és formátumnak, ezeket azonnal hibalistára tettem. Később kiderült, hogy a kapott fájlban lévő adatok meglehetősen piszkosak voltak, a két leginkább szembetűnő probléma itt a társaságok neveinek különböző módon történő gépelése (elgépelések, kis- és nagybetűvel írt cégnevek, a nevekben lévő gazdasági formák és cégállapotok különböző formájú megjelenése) és a címek nagyfokú inkonzisztenciája. Az adattábla további részeiben az egyértelmű kód-érték mezőpárok nagyjából megfelelőek voltak, itt csak egy-egy módosítást kellett végrehajtanom az adattáblában. Például az azonos településkódhoz tartozó Dunavecse és DunavecsDunavecse településnevek egységesítve lettek Dunavecse néven.

A társaságok rövid és hosszú neveinek elméletben bijektívnek kellene lennie, a táblában ez a tulajdonság nagyon sok adatnál nem állt fenn, ezért a táblát az újraimportálás után több ízben is módosítottam. A rövid neveknél egységesen eltávolítottam a kezdő újsor karaktert, a hosszan kiírt gazdasági formára utaló részt pedig egységesen felváltottam annak rövidítésével, eltávolítottam a dupla szóközöket és pontokat, a cég állapotára vonatkozó részt eltávolítottam, a név végén lévő felesleges szóközöket kitöröltem, továbbá bizonyos adatokon kézi beavatkozást is elvégeztem. (A kezdő újsor karakter az adatok importálás előtti szemléjekor fel sem tűnt, akkor vettem észre, hogy valami nem stimmel a társaságok rövid neveinél, amikor az egyiket konkrétan lekérdeztem egy SQL select-tel, és egyetlen sort sem adott eredményül. Ezután kimásoltam az SQL felületbe az egyiket két idézőjel közé – hogy biztosan lehessen tudni, hogy milyen karakterek vannak a szöveg elején és végén – és ekkor vettem észre, hogy a beillesztéskor emelt egy sort. Ekkor kitöröltem mindegyik rövid név elejéről az újsor karaktert, mert kiderült, hogy minden rövid név elején van egy, és ezután lefuttatva a fentebb említett SQL select-et megkaptam a megfelelő sorokat.) Hasonló módosításokat végeztem a hosszú neveknél is, de ott a rövidítések helyett a teljes gazdasági formát írtam ki, ahol kellett. Ehhez nagy segítségemre volt a HANA rendszerbe épített SQL workbench, amelynek a szövegkezelő függvényei kiszolgálták a fenti módosításokat, továbbá támogatja az egyes adatok kézi módosítását is. Kísérleteztem azzal is, hogy megnézem, hogy mely rövid, illetve hosszú nevek prefixei egymásnak, ez további segítséget adott az egyes kézi módosításokhoz, ugyanis volt olyanra is példa, hogy a cégnév gépelése a felénél abbamaradt. A nevek alakítgatásának a befejezésével szembesültem azzal, hogy biztosan nem érhető el teljes mértékben az az állapot, hogy a rövid és a hosszú nevek bijektívek legyenek.

A címadatokban a korábban említett inkonzisztencia jelentett problémát. Az közterület-jelzések megjelentek rövidítésben és teljes kiírásban is, a házszám után vagy megjelent a pont, vagy nem, volt, ahol kettő is, az épületjelzés sem volt mindig jól kiírva. Ezeket SQL utasítások segítségével próbáltam orvosolni, a közterületjelzéseket mindenhol lecseréltem a teljes kiírásra (például u. helyett utca), a pontok után szóközt tettem (mert volt olyan cím, ahol a cím végén lévő komponensek csak ponttal voltak elválasztva), az épületjelzéseket egységesítettem nagybetű épület formátumra, a felesleges pontokat és szóközöket eltávolítottam (már amennyit kézi módosítás nélkül el lehet távolítani), és a településnevek utáni vesszőt eltávolítottam, így egységes lett az is. Továbbá a budapesti címekből kivettem a kerületjelzéseket, mivel az úgyis egyértelműen kiderül az irányítószámból, másrészt ez is van, mikor megjelenik a címben, van, mikor nem. Kísérleteztem olyan program írásával is, amely a címeket szétbontja komponensekre, de annyi volt az olyan formátumú, amivel nem tudtunk mit kezdeni (például a helyrajzi számmal ellátott címek), így inkább elvetettem a továbbfejlesztést. Ez azért lett volna hasznos, mert komponensekben tárolva a címet sokkal átláthatóbb lett volna az adathalmaz, ráadásul a geokódot előállító alkalmazás is profitált volna a komponensenkénti inputból.

A címadatoknál gondolkodtam azon is, hogy esetleg azokat is ki kellene szedni egy külön cím-kód táblába, ahol egyszerűbben meg lehetett volna még azt csinálni, hogy a címek ténylegesen egységesítve legyenek, ennek a segítségével azt is meg lehetett volna oldani, hogyha két cím így egyező lett volna, akkor abból csak az egyiket elég lett volna megtartani, a változást rögzíteni egy külön táblában, majd a végén, mikor már valóban minden cím különböző, akkor a változástábla alapján az eredeti táblában is be lehetett volna állítani a címeknél a megfelelő kódot. A kigyűjtés abból a szempontból is hasznos lett volna, hogy kisebb a redundancia az adatokban, elvégre a címadatok a kapcsolótáblában lettek volna, ami nagyjából ötezer sor, míg a különböző címekből becsléseim szerint nagyjából 1300 darab volt (ha nem kevesebb).

Az adatok tisztítása során számos olyan sor került ki az adatbázisból, ami a fenti anomáliáktól eltekintve azonosak voltak.

Miután az adatok már relatíve letisztultak, megpróbáltam felépíteni azt a sémát, amelyben minél kevesebb kulcs van, és minél kevesebb redundancia legyen az adatokban. Az adatok tisztítása közben számos alkalommal kellett újraterveznem az addig kialakított sémát, mert a közben végrehajtott SQL lekérdezések újra és újra rámutattak egyre bonyolultabb (és érdekesebb) összefüggésekre, amelyekre az adatok első pár sorát megnézve szinte nem is lehet rájönni. A végén azonban sikerült egy olyan, elfogadhatónak nevezhető sémát kialakítani, amelyben a huszonhat oszlophoz mindössze három kulcs tartozik, elvégre a társaságok minden további adatát meghatározza az adószám, az adatszolgáltatás éve és a gazdasági forma. Utóbbira is szükség van, mert volt számos olyan cég, amelyről egyazon évben két bejegyzés is készült eltérő gazdasági formával, amit nem küszöbölhetünk ki anélkül, hogy számos nem megfelelő sor képződne ezzel együtt. Ezeken felül a név-kód típusú párokat két helyen is hierarchiába lehetett szervezni, amivel a kapcsolótáblából kikerültek további oszlopok is. a hierarchiába szervezés a következő két esetben valósulhatott meg: a nemzetgazdasági szakágazat egyértelműen meghatározza a nemzetgazdasági ágazatot, és a nemzetgazdasági ágazat egyértelműen meghatározza a nemzetgazdasági ágat, illetve a település egyértelműen meghatározza a megyét, a megye egyértelműen meghatározza a régiót és az országot. A régiókból ebben az adatbázisban azért nem következik egyértelműen az ország, mert a határon túli cégeknél a régió mező értéke egyéb, az ország viszont egyértelműen meg van adva, továbbá a fordított következés sem áll fenn, mert összesen hét magyarországi régió szerepel az adatbázisban.

Ezen meggondolások alapján a HANA rendszerben is el lehetett készíteni a végső sémát, és ki lehetett szervezni a név-kód párokat, illetve a kapcsolótáblát új táblákba.

Az adatbázis kialakításán túl egyéb feladatokat is végeztem. A geokódokat előállító alkalmazás ugyebár JSON formátumban kéri be az adatokat, az online konverterek viszont csak az első ezer sort alakítják át CSV formátumból. Arra tettem kísérletet, hogy írjak egy olyan programot, amely ezernél több sort is át tud alakítani. Output mintának az egyik olyan kisebb JSON fájlt használtam mintának, amelyet még OData-n keresztül nyertünk ki a HANA rendszerből. Hosszú órák munkájával végül sikerült megírni hozzá a C++ kódot. A kapott JSON fájl mérete nagyon nagy lett, az általunk használt online JSON validátor (<https://jsonformatter.org/>) is elfogadta a JSON fájlunkat, viszont a geokódot előállító JAVA alkalmazás minden adatot nullértékként olvasott be, amit a mi tudásunkkal már nem tudtunk kezelni, így ez a kísérlet is kudarcba fulladt.

További kísérletezéseket folytattam a kalkulációs nézet létrehozására és megjelenítésére is, amely alapvetően az általunk elvárt működést produkálta, azonban az egyik kísérletnél az előnézetben rábukkantam egy olyan hiányosságra, miszerint aggregálásnál nem lehet beállítani azt, hogy csak a különböző nevek legyenek számolva.

1. Szenzor alkalmazásban való ténykedés

Az alkalmazás röviden arról szól, hogy az adatfolyamon érkező szenzoradatokat előfeldolgozzuk, ezután betesszük őket egy adatbázisba, majd azt meg is jelenítjük. Idő hiányában ebből csak a feladat egy részét tudtuk megoldani, az éles alkalmazás elkészítése a szakmai gyakorlat keretein kívül kerül majd megvalósításra.

Első részfeladatként az adatok streamen való beérkezésén és előfeldolgozásán gondolkodtam. Ugye eredendően az lett volna itt a lényeg, hogy a szenzorból az adatok először egy „ködbe” érkeznek, és majd ott történik az előfeldolgozás, illetve az adatbázisban való véglegesítés. Az én ötletem az volt, hogy ez a köd is az SAP HANA rendszer legyen, ahol a beérkező adatokat egy ideiglenes táblában helyezzük el, és minden óra végén ezekből az adatokból aggregáljuk az adatbázisba bekerülő adatot, majd az ideiglenes táblát kiürítjük. Az előzetes elmélkedésben felmerültek bennem olyan kérdések is, hogy az adatok megjelenítésére milyen idősávokat kínálunk a felhasználónak, illetve hogy az adatbázis túlságosan nagyra növekedését elkerülendő a régebbi adatok milyen időintervallumokra legyenek tárolva. Végül ezek a kérdések nem lettek tovább pedzegetve, ugyanis idáig már nem jutottunk el.

A feladat kiadásakor kaptunk korábbi adatokat az adatbázis kialakításához mintaként, először ezeket kellett importálnunk. Először viszont a kapott tizenkét CSV fájlt (2017 minden hónapjára egy) egy átalakításnak kellett alávetnünk, ugyanis egy adatsor összesen 336 adatot tartalmazott az időjelzésen túl, az ötvenhat szenzorhoz tartozó 6-6 tulajdonságot. Erre írtam egy C++ programot, amely azt csinálja, hogy ennek a tizenkét CSV fájlnak a felhasználásával egy nagy CSV fájlt hoz létre, amelyben minden adatsorban az egy időjelzéshez, egy szenzorhoz és egy tulajdonsághoz tartozó adatot tartalmazza. Ez összesen kicsit több, mint négymillió adatsort jelent (az átalakítás futásideje nagyjából 18,5 másodperc volt, az eredmény egy nagyjából száz megabájtos fájl lett), ugyanakkor ezt ebben a formában sokkal könnyebb volt már importálni a HANA rendszerbe. Nem is beszélve arról, hogy ebből a táblából el tudtuk készíteni a kívánt sémájú adattáblánkat egyetlen viszonylag egyszerű SQL utasítással. Ez a kívánt séma nyolc mezőből áll, az időjelzésből, a szenzor sorszámából és a hat tulajdonsághoz tartozó értékből. Ezen felül a szenzoradatok is importálva lettek, amelyek a szenzor sorszámát és geokoordinátáit tartalmazta.

Számos SQL lekérdezést hajtottam végre az adatok megismeréséhez és értelmezéséhez. A majdani megjelenítéshez elkészítettem a hőmérsékleti adatok napi, heti és havi kivonatát, ugyanis gyanítottam, hogyha majd kívülről szeretnénk az adatokat ilyen idősávú megjelenítésre elérni, akkor már legyen egy minta ahhoz, hogy ezt miként is kellene megvalósítani. A tapasztalataim ebből a szempontból viszont meglehetősen negatívak voltak, ugyanis számos SQL kulcsszó nem elérhető a HANA rendszerben (például a natural join, de ez már korábban is szembeötlött). Végül nagy nehezen sikerült megtalálni az az oldalat, ahol elérhetőek voltak a HANA rendszerben használható SQL lekérdezések listája: <https://help.sap.com/viewer/4fe29514fd584807ac9f2a04f6754767/2.0.03/en-US/20a61f29751910149f99f0300dd95cd9.html>. Ennek, illetve a <https://www.w3schools.com/> oldal SQL alfejezetének a segítségével a HANA rendszerből hiányzó dátumidő-kezelő függvényeket is sikerült helyettesíteni más függvények együttes használatával.

A napi lekérdezés a HANA rendszerben:

select substring("date\_time", 1, 10), max("temperature"), min("temperature"), avg("temperature")

from "SENSORS\_HDI\_SENSORS\_DB"."sensors.sensor\_data\_2"

where "sensor\_id" = 171 and "temperature" is not null

group by (substring("date\_time", 1, 10))

order by 1;

A heti lekérdezés a HANA rendszerben:

select

add\_days(cast(substring("date\_time", 1, 10) as date), -(mod(days\_between( '1900-01-01', cast(substring("date\_time", 1, 10) as date)), 7))) as first\_day\_of\_week,

add\_days(cast(substring("date\_time", 1, 10) as date), 6-(mod(days\_between( '1900-01-01', cast(substring("date\_time", 1, 10) as date)), 7))) as last\_day\_of\_week,

max("temperature"), min("temperature"), avg("temperature")

from "SENSORS\_HDI\_SENSORS\_DB"."sensors.sensor\_data\_2"

where "sensor\_id" = 171 and "temperature" is not null

group by

add\_days(cast(substring("date\_time", 1, 10) as date), -(mod(days\_between( '1900-01-01', cast(substring("date\_time", 1, 10) as date)), 7))),

add\_days(cast(substring("date\_time", 1, 10) as date), 6-(mod(days\_between( '1900-01-01', cast(substring("date\_time", 1, 10) as date)), 7)))

order by 1;

A havi lekérdezés a HANA rendszerben:

select substring("date\_time", 1, 7), max("temperature"), min("temperature"), avg("temperature")

from "SENSORS\_HDI\_SENSORS\_DB"."sensors.sensor\_data\_2"

where "sensor\_id" = 171 and "temperature" is not null

group by (substring("date\_time", 1, 7))

order by 1;

Megjegyzés: az adatbázisomban az időjelzések szöveg formátumban lettek tárolva, YYYY-MM-DDTHH:MM::SS formátumban.

A fentieken túl kísérletet tettem egy HTML5 modul elkészítésére is a feladathoz. A modul lényege az lett volna, hogy OData-n keresztül lekéri az adatbázisból az adatokat, és ahhoz valamiféle megjelenítést ad. Nagyon elégedett lettem volna, ha sikerült volna grafikonos megjelenítést adni az adatoknak, de végül be kellett érnem a táblázatos formával is.

A modul úgy működik, hogy megjelenít két választódobozt (ComboBox), ahol ki lehet választani a szenzor sorszámát és a lekért adat típusát. Előbbihez a szenzorok táblájából kértem le az adatokat, utóbbihoz pedig az adatok táblájából kértem le a fejléceket, így ha az adatbázis sémája valamilyen oknál fogva változik, akkor a lekérhető tulajdonságok is megváltoznak. Értelemszerűen ha valaki érvénytelen adatot akar megadni a két választódoboz bármelyikében, akkor az az egész lekérés érvénytelen lesz. A két megadott érvényes adat segítségével egy harmadik kérést is küldünk az OData-nak, amellyel már a kimenő adatokat fogjuk megkapni. Vagyis csak az első 500 sort, mert annyit tudunk lekérni. A kapott táblázat két oszlopból áll, az elsőben az időjelzés, a másodikban pedig az adatok vannak. Az időjelzés itt azért is nagyon fontos, mert direkt csak a nem null értékeket kértük le, hogy amit megjelenítünk, annak értelme is legyen.

A modul fejlesztése meglehetősen nehéz volt. Az SAP-nak az API reference manual-ja nem túlságosan bőbeszédű (a hozzá tartozó link: <https://sapui5.hana.ondemand.com/#/api/sap.m>), ráadásul ami le van írva, az sem magától értetődő. A választódobozok e létrehozásánál többszöri újragondolás és újraolvasás után jöttem rá, hogy az elemek hozzáadásakor nem szöveg típusú adatként kell hozzáadni új elemeket a listához, hanem Item típusú elemeket kell hozzáadni, amelynek ő jeleníti meg a szöveg attribútumát. Nehézségeim voltak továbbá a táblázat létrehozásakor is, ugyanis azt is az SAP erre vonatkozó könyvtárának a segítségével próbáltam megcsinálni, de a kísérletem kudarcba fulladt, mert a táblázathoz se sort, se oszlopot, se fejlécet nem tudtam hozzáadni. Így a többi GUI-s elemmel ellentétben ezt már JavaScriptből hoztam létre az adatok számától függően. A táblázat elkészítéséhez segítségemre volt a következő link: <https://renenyffenegger.ch/notes/development/web/HTML/tags/table/create-with-js>.

A modul JavaScript háttérkódját ennél egy kicsit könnyebben valósítottam meg, hiszen a szakmai gyakorlat keretein belül ismerkedtem meg a nyelvvel, ez külön fejezetben is ki lesz fejtve. Azonban itt is szembesültem egy problémával, mert a JavaScript beépített JSON.parse függvénye a kapott JSON formátumú tömberedményt nem volt hajlandó parszolni, így a tömbelemeket kézzel kellett szétválasztani, az egyes elemekre meg már jól működött a beépített függvény.

A modul futtatásához egy megjegyzés, hogy a böngészőben telepíteni és engedélyezni kell a CORS policy bővítményt, különben a modul nem éri el OData-n keresztül az adatokat, így azt használni sem lehet.

1. Ismeretanyag-elsajátítás

A szakmai gyakorlat kezdetén a legtöbbünk számára ismeretlen volt az SAP HANA rendszere, így az első két hetünket tanulással töltöttük. Szakmai vezetéssel megismerkedtünk a fejlesztőrendszerrel és elkészítettük az első Hello World szintű alkalmazásunkat (ami nem egyezik az első fejezetben leírt TinyWorld alkalmazással). Ezután önálló tanulással megismerkedtünk a különböző modulok létrehozásával, buildelésével és elindításával, a kalkulációs nézet elkészítésével és megjelenítésével, adatbázis létrehozásával és a beépített SQL felület használatával. Ezt a következő linkek mentén kezdtük el:

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

és

<https://developers.sap.com/mission.xsa-get-started.html>

Sajnálatos módon azonban a második linken lévő anyagok, illetve azok folytatásai elavultak voltak, így az első link kiadásáig kénytelenek voltunk magunkra hagyatkozni az ismerkedést illetően.

Az önfejlesztést nagyban nehezítette az első héten az, hogy nem állt rendelkezésünkre állandó jelleggel működő szerver. A legtöbb anyagot csak olvasással lehetett elsajátítani, a gyakorlatban nem sikerült maradéktalanul kipróbálni az anyagban éppen szereplő funkciókat.

Az első két héten még nem volt körvonalazódva, hogy a modellezés területén fogok leginkább tevékenykedni, így egyéb anyagokat is elkezdtem olvasni, különösen az adatbányászat terén. Tajti Ákos diplomamunkája a multidimenziós adatbázisok adattisztításáról:

<https://dea.lib.unideb.hu/dea/bitstream/handle/2437/101398/tajti_akos_diplomamunka.pdf;jsessionid=BF55D7B0D14F6FE2C9399F842960C145?sequence=1>

A TANE-algoritmusról:

<https://www.lri.fr/~pierres/donn%E9es/save/these/articles/lpr-queue/huhtala99tane.pdf>

Megjegyzés a modellezéshez, mint témakörhöz, hogy annyira sokrétű és önmagában annyira meghatározhatatlan része az adatbázis-kezelésnek, hogy még a Google sem tudott egyértelmű segítséget nyújtani abban, hogy ez alatt az emberek konkrétan mit is értenek, de az előző szemeszterben már tanultunk az adatbázisokról egyet s mást, így volt arról sejtésünk, hogy mit kell csinálni a modellezésben.

A specializációmmal szöges ellentétben elkezdtem a szakmai gyakorlat keretein belül megismerkedni a JavaScript nyelvvel, ehhez a <http://javascript.info/> linket használtam. Azért ennek álltam neki feladat híján, mert egyrészt a JavaScript ismerete nagyon hasznos, másrészt meg elengedhetetlennek bizonyul, ha a továbbiakban Angularral vagy egyéb, JavaScriptet szervesen használó technológiában fogok fejleszteni. Utólag, mint kiderült, tényleg elengedhetetlen volt a HTML modul készítésénél.

Eleinte kicsit nehezítő körülménynek tartottam azt, hogy a debugoláshoz a két leginkább használható módszer a felugró ablak és a konzol, amit a böngészőben csak a megfelelő nézetben lehet látni. Az alábbi dolgokat tanultam meg használni a JavaScript nyelvben:

* alapvető vezérlési szerkezetek
* típusok, illetve az egyes típusokhoz tartozó alapvető kezelőfüggvények
* interakció a felhasználóval
* objektumok használata
* függvények használata
* osztályok, az objektumorientált programozás alapjai és támogatása
* hibakezelés
* aszinkronitás

A tapasztalatom ezzel kapcsolatban annyi, hogy sokkal kellemesebb a JavaScriptet használni, mint az a HTML-t, amelybe elméletileg bele kellene ágyazni.