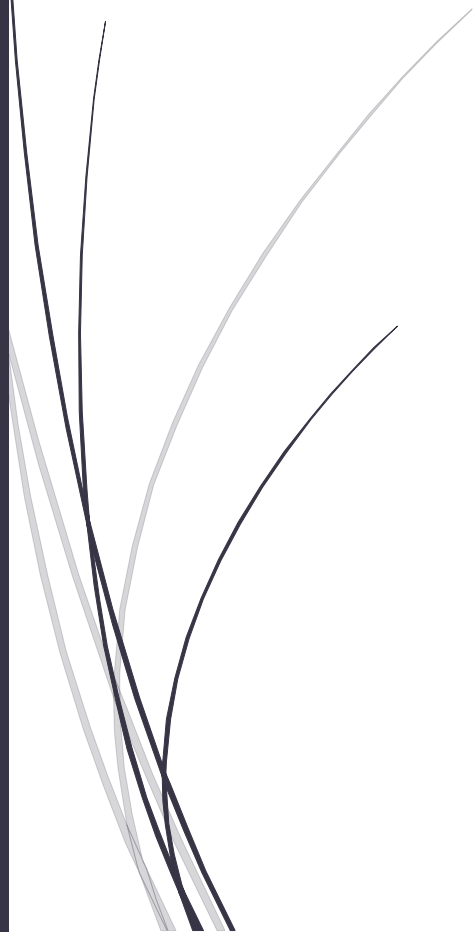
2020. 05. 16.

Projektmunka I.

Féléves feladat dokumentáció

Bózsó Bence



TARTALOMJEGYZÉK

BEVEZETÉS

A PROJEKT TÉMÁJA

A projekthez a *2019/2020/1* félévben az *Adatbázisok* tárgyhoz készült beadandó munkámat[1] fogom felhasználni, kibővíteni olyan módon, hogy az megfeleljen a projektmunka követelményeinek.

A PROJEKT ALAPJÁT ADÓ SZOLGÁLTATÁS MŰKÖDÉSE

Egy mobiltelefon segítségével használható közösségi autóbérlő alkalmazás adatbázisát tervezem meg. A szolgáltatásra való regisztrációkor az alkalmazás rögzíti a felhasználó személyes adatait, valamint a jogosítványa adatait, mivel enélkül az autóbérlés nem lehetséges. Ezután a felhasználó kiválaszthatja a számára legoptimálisabb előfizetési konstrukciót (havidíj + alacsonyabb percdíj, vagy havidíj nélkül magasabb percdíj). Ezt követően a felhasználó számára megjelenik a szolgáltatási övezet térképe, amin szerepelnek az elérhető autók, valamint ezek adatai (típus, pozíció, töltöttség). Többféle típusú, méretű autó is rendelkezésre állhat, bizonyos típusoknál feláras is lehet annak a választása. A jármű kiválasztása után a felhasználó lefoglalhatja azt, a mobilalkalmazás segítségével nyithatja és használhatja. A bérlés végeztével az autót le kell parkolni a szolgáltatási övezeten belül, és az alkalmazásban leállítani a bérlést. Ha bármilyen problémát észlel az autóval (tisztasági probléma, törés stb.) ezt ekkor jelentheti, képekkel dokumentálhatja, ez a rendszerben rögzítésre kerül. A bérlés lezárását követően a rendszer elkészíti a számlát az előfizetése, bérlés időtartama, illetve esetlegesen az autó felára alapján, amit a felhasználónak ki kell egyenlítenie. A bérlés végétől az autó ismét elérhetővé válik a többi felhasználó számára is. A szolgáltatás több városban is működik, minden városban a megfelelő működtetésért felelős személyzettel. Az általam tervezett adatbázis a rendszer működéséhez szükséges legfontosabb adatokat tárolja.

[1] A hivatkozott beadandó feladat elérhető itt.

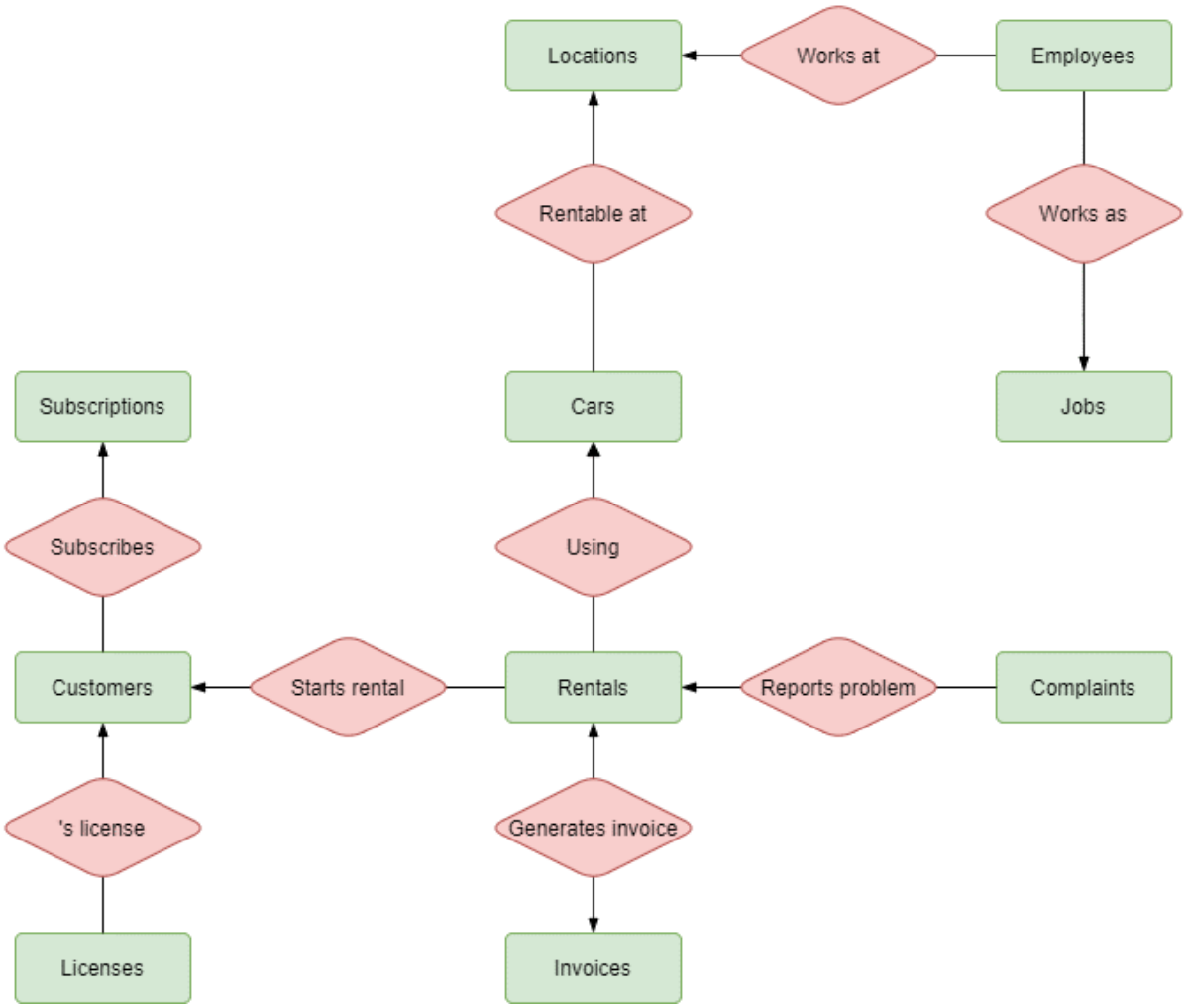
AZ ADATOK FORRÁSA

Mivel egy korábbi projektemet folytatom, ezért nem találtam hozzá megfelelő adathalmazt az interneten, így azokat magamnak generáltam le. Ehhez részben külső alkalmazást használtam, viszont az egyes táblák közti kapcsolat, vagy a szoftver mennyiségi korlátjai miatt ez a legtöbb esetben nem működött, így az adatok jelentős részét saját alkalmazással generáltam le. A továbbiakban minden táblához fel lesz tüntetve, hogy a benne található adatok milyen módon kerültek létrehozásra, szükség esetén kódrészlettel együtt.

TERVEZÉS

EK DIAGRAM

Az egyed-kapcsolat diagram megtervezéséhez a draw.io asztali alkalmazását használtam, mivel ez egy egyszerűen kezelhető, ingyenes, és minden szükséges funkcionalitással rendelkező eszköz.



Az eredeti diagramhoz képest több új táblát is hozzáadtam, valamint egységesen angol nyelvre lett átírva minden tábla, mező, és -néhány kivétellel- a bennük található rekordok is.

AZ ADATBÁZIS TÁBLÁI ÉS TÁROLT ADATAI

CUSTOMERS

Ebben a táblában tároljuk el az ügyfeleket, akik már regisztráltak a szolgáltatás használatára.
A következő adatok kerülnek eltárolásra:

- CUSTOMER_ID: az ügyfél azonosítója, ez lesz az elsődleges kulcs,
- SUBSCRIPTION_ID: a kiválasztott előfizetés azonosítója, idegen kulcs,
- FIRST_NAME: az ügyfél keresztneve,
- LAST_NAME: az ügyfél vezetékneve,
- EMAIL: az ügyfél e-mail címe,
- PHONE: az ügyfél telefonszáma,
- BIRTH_DATE: az ügyfél születési dátuma,
- CITY: az ügyfél lakhelye település szinten.

LOCATIONS

Ebben a táblában a szolgáltatást működtető vállalat telephelyeit tároljuk.
A következő adatokat tároljuk:

- LOCATION_ID: a telephely azonosítója, ez az elsődleges kulcs
- ADDRESS: a telephely címe,
- POSTAL_CODE: a telephely irányítószáma,
- CITY: a telephely városa,
- STATE: az állam, ahol a telephely található.

EMPLOYEES

Ebben a táblában a cég dolgozóit tároljuk, akik felelősek a szolgáltatás működtetéséért.
A következő adatokat tároljuk:

- EMPLOYEE_ID: az alkalmazott azonosítója, ez lesz az elsődleges kulcs,
- FIRST_NAME: az alkalmazott keresztneve,
- LAST_NAME: az alkalmazott vezetékneve,
- EMAIL: az alkalmazott e-mail címe,
- PHONE: az alkalmazott telefonszáma,
- LOCATION_ID: a telephely azonosítója, ahol az alkalmazott dolgozik, idegen kulcs,
- HIRE_DATE: a munkaviszony kezdete,
- JOB_ID: az alkalmazott munkakörének azonosítója,
- SALARY: az alkalmazott fizetése,
- SALGRADE: az alkalmazott fizetési kategóriája.

JOBS

A vállalatnál aktív munkakörök adatait tároljuk ebben a táblában:

- JOB_ID: a munkakör azonosítója, elsődleges kulcs,
- JOB_TITLE: a munkakör megnevezése,
- BASE_SALARY: a munkakör alapbére, ez kerül szorzásra az alkalmazott fizetési kategóriájával a fizetés megállapításához.

SUBSCRIPTIONS

Ebben a táblában tároljuk az előfizetések adatait, amik közül a felhasználók választhatnak.

- SUBSCRIPTION_ID: az előfizetés azonosítója, elsődleges kulcs,
- MONTHLY_PRICE: az előfizetés havidíja,
- MINUTE_PRICE: az előfizetés percdíja.

CARS

Ebben a táblában a szolgáltatás használata során biztosított autók adatait tároljuk.

- CAR_ID: az autó azonosítója -rendszáma-, ez az elsődleges kulcs,
- BRAND: az autó márkája,
- MODEL: az autó modellje,
- FUEL: az üzemanyag / töltöttség aktuális állapota,
- LOCATION_ID: a telephely azonosítója, amihez az autó tartozik.

LICENSES

A szolgáltatás használatához a felhasználónak mindenképp érvényes jogosítvánnyal kell rendelkezni, ezeknek az adatait tároljuk ebben a táblában.

- LICENSE_ID: a jogosítvány száma, elsődleges kulcs,
- CUSTOMER_ID: az ügyfél, akihez a jogosítvány tartozik,
- CATEGORY: a kategória amire az adott jogosítvány érvényes,
- START_DATE: a jogosítvány érvényességének kezdete,
- EXPIRY_DATE: a jogosítvány érvényességének vége.

RENTALS

Ebben a táblában a bérlések adatait tároljuk el.

- RENTAL_ID: a bérlés azonosítója, ez az elsődleges kulcs,
- CAR_ID: az autó azonosítója, amivel a bérlést indították,
- CUSTOMER_ID: az ügyfél azonosítója, aki a bérlést indította,
- DISTANCE: az utazás során megtett távolság,
- START_TIME: a bérlés indításának időpontja,
- END_TIME: a bérlés befejezésének időpontja.

INVOICES

A bérlés lezárásakor egy számla keletkezik, ezeknek az adatait tároljuk ebben a táblában.

- INVOICE_ID: a számla azonosítója, ez az elsődleges kulcs,
- RENTAL_ID: a bérlés azonosítója, amihez a számla tartozik,
- COMPLETION_TIME: a számla kiegyenlítésének időpontja,
- AMOUNT: a bérlés díja, az ügyfél előfizetésének díjából és a bérlés időtartamából kerül kiszámításra.

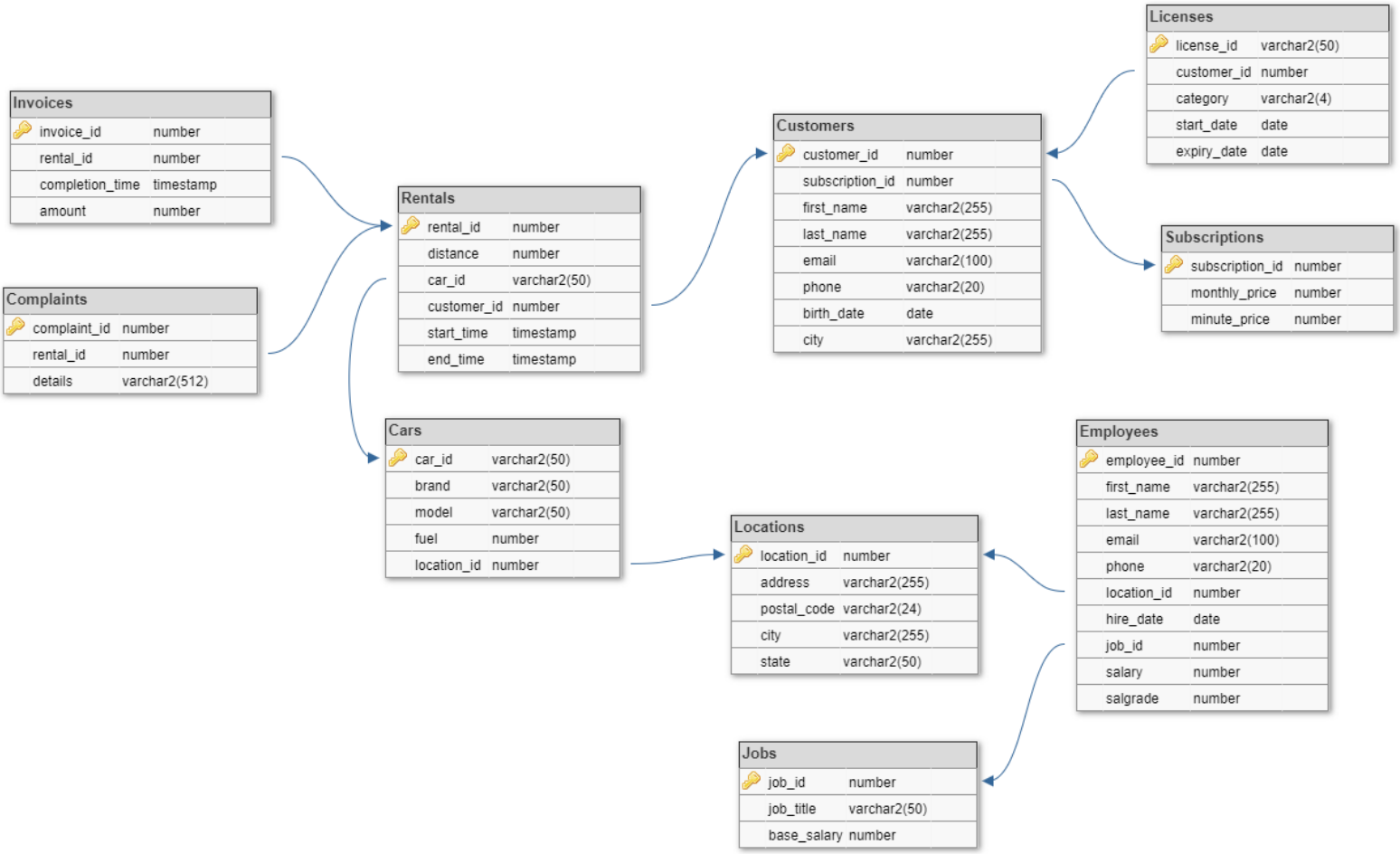
COMPLAINTS

A bérlés lezárásakor a felhasználó jelentheti az esetleges káreseményeket, észrevételeket, ezeknek a bejelentések adatait tároljuk ebben a táblában.

- COMPLAINT_ID: a bejelentés azonosítója, elsődleges kulcs,
- RENTAL_ID: a bérlés azonosítója, amihez a bejelentés tartozik,
- DETAILS: a bejelentés részletei.

A VÉGLEGES TÁBLÁK ÉS KAPCSOLATAIK

A táblák megtervezése[2] után a következő lett az adatbázis végleges modellje:



[2] A táblákat létrehozó script megtalálható a mellékletben, a tartalomjegyzék végén. A modell a ***dbdesigner.net*** használatával készült.

MEGVALÓSÍTÁS

A táblák megtervezése és létrehozása után elkészítettem az ezekhez szükséges adatokat amikkel azokat feltöltöttem, valamint létrehoztam az egyéb szükséges funkciókat. (sequence, trigger)

A TÁBLÁK FELTÖLTÉSE ADATOKKAL

Ebben a fejezetben tábla szintű bontásban ismertetem, hogy milyen módon kerültek az adatok létrehozásra, valamint hány rekordot tartalmaz az adott tábla.

CUSTOMERS

- Az adathalmaz létrehozásában a ***dbForge Studio for Oracle*** nevű eszköz volt segítségemre. Ezzel létre tudtam hozni minden mezőhöz a szükséges adatokat, mivel nem volt olyan idegen kulcs, ami ezt akadályozta volna.
- Ebbe a táblába így **15.000** darab rekordot szúrtam be.

LOCATIONS

- Az adathalmaz létrehozásában itt is a ***dbForge Studio for Oracle*** segített, ugyanis helyszínre vonatkozó adatok is vannak az adatbázisában, egyszerűen lehetett generálni a szükséges adatokat.
- Ez a tábla **9** rekordot tartalmaz, ennél több telephelyet nem láttam indokoltnak.

EMPLOYEES

- Az eddigiekhez hasonlóan itt is tudtam használni a már bemutatott ***dbForge Studio for Oracle*** eszközt, mivel egyik mező sem olyan, ami komolyabb egyeztetést igényelne, így a program gond nélkül létre tudta hozni a szükséges mennyiségű adatot.
- A tábla **5000** darab rekordot tartalmaz.

JOBS

- Ez a tábla csak az egyes munkakörök adatait tartalmazza, amiből jellemzően nincs kiemelkedően sok, ezért ehhez kézzel készítettem el az adatokat, nem használtam hozzá egyéb eszközt.
- A tábla **8** darab rekordot tartalmaz, ennél több létrehozását szükségtelennek láttam.

SUBSCRIPTIONS

- A szolgáltatás működésének leírásánál is csak három féle előfizetési konstrukció volt meghatározva, ezért ezeket szintén kézzel készítettem el. Ugyanakkor fel van készítve a bővítésre, új előfizetés hozzáadásakor is megfelelően működik a rendszer.
- A tábla **3** darab rekordot tartalmaz.

CARS

- Az autók létrehozását megpróbáltam a fentebb is említett ***dbForge Studio for Oracle*** eszköz használatával, azonban nem sikerült megfelelően összeegyeztetni a modellt a gyártóval, sem a rendszámot megfelelően, egyediként kialakítani, pedig ez az elsődleges kulcs miatt alapvető elvárás. Emiatt saját programot írtam, ami sikeresen előállította a szükséges adatokat figyelve arra is, hogy kétszer ugyanaz a rendszám semmiképp nem fordulhat elő. Alább látható az autókat létrehozó kódrészlet.
- A tábla **9572** rekordot tartalmaz.

```
class CarGeneration
{
    List<BrandModelPair> availableCars;
    List<Car> cars;
    public CarGeneration()
    {
        availableCars = new List<BrandModelPair>();
        availableCars.Add(new BrandModelPair("Volkswagen", "e-Golf"));
        availableCars.Add(new BrandModelPair("Volkswagen", "e-Up"));
        availableCars.Add(new BrandModelPair("Audi", "e-Tron"));
        availableCars.Add(new BrandModelPair("Nissan", "Leaf"));
        availableCars.Add(new BrandModelPair("BMW", "i3"));
        cars = new List<Car>();
        rnd = new Random();
    }
    static Random rnd;
    private void GenerateCars(int count)
    {

        for (int i = 0; i < count; i++)
        {
            BrandModelPair bmp = availableCars[rnd.Next(availableCars.Count)];
            Car c = new Car(GenerateRegistration(), bmp.Brand, bmp.Model, rnd.Next(15, 101), rnd.Next(1, 10));
            while (cars.Contains(c))
            {
                c = new Car(GenerateRegistration(), bmp.Brand, bmp.Model, rnd.Next(15, 101), rnd.Next(1, 10));
            }
            cars.Add(c);
        }
        ExportToJson(cars);
    }

    private string GenerateRegistration()
    {
        string abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        string carId = rnd.Next(10).ToString();
        for (int j = 0; j < 3; j++)
        {
            carId += abc[rnd.Next(abc.Length)].ToString();
        }
        carId += rnd.Next(100, 1000).ToString();
        return carId;
    }

    public void ExportSQL(List<Car> input)
    {
        StreamWriter sw = new StreamWriter("cars.sql");
        foreach (Car item in input)
        {
            sw.WriteLine(item);
        }
        sw.Close();
    }
```

LICENSES

- A jogosítványok létrehozásában a ***dbForge Studio for Oracle*** segítségét vettem igénybe, mivel sikeresen tudta kezelni azt az egy idegen kulcsot, amit ez a tábla tartalmaz.
- A táblában **13.275** darab rekord található.

RENTALS

- Ennek a táblának a feltöltése volt a legnehezebb az összes közül. Egy-egy rekord generálása során figyelni kell arra, hogy létező ügyfelet, autót rendeljünk hozzá, a kezdő és befejező időpontot úgy kell kisorsolni, hogy egy autót ne használjon több ügyfél egy időben, valamint egy ügyfélnek szintén ne legyen több bérlése ugyanabban az időintervallumban. Ilyen szintű adategyeztetéseket az általam ismert tesztadat-generátorok nem voltak képesek elvégezni, így ezek használata megoldhatatlan volt, a problémát az sem könnyítette, hogy ebben a táblában található a legtöbb rekord. Az adatok generálását ezért itt is saját programmal oldottam meg, ami képes volt kezelni ezeket a problémákat is. Alább látható az ezt megvalósító kódrészlet.
- A táblába végül **1.435.800** darab rekord lett beszúrva.

```
class RentalGeneration
{
    public List<Car> Cars { get; set; }
    public List<Rental> Rentals { get; set; }
    private int index;
    public RentalGeneration(List<Car> cars)
    {
        Cars = cars;
        Rentals = new List<Rental>();
        rnd = new Random();
        index = 0;
    }
    public void ExportDSV(List<Rental> rentals)
    {
        StreamWriter sw = new StreamWriter("rentals2.dsv");
        string starterLine = $"\"RENTAL_ID\";\"DISTANCE\";\"CAR_ID\";\"CUSTOMER_ID\";\"START_TIME\";\"END_TIME\"";
        sw.WriteLine(starterLine);

        foreach (var item in rentals)
        {
            sw.WriteLine(item);
        }

        sw.Close();
    }
    private void GenerateRent()
    {
        int c = 0;
        for (int k = 0; k < Cars.Count; k++)
        {
            Car item = Cars[k];
            for (int i = 0; i < 150; i++)
            {
                int rentalId = ++index;
                int customerId = rnd.Next(1, 15000);
                string carId = item.CarId;
                DateTime[] dates = GenerateDate();
                double distance = Math.Round((dates[1].Subtract(dates[0]).TotalMinutes * 0.7), 0);
                Rental r = new Rental(rentalId, (int)distance, carId, customerId, dates[0], dates[1]);
                while (Rentals.Contains(r))
                {
                    dates = GenerateDate();
                    distance = Math.Round((dates[1].Subtract(dates[0]).TotalMinutes * 0.7), 0);
                    r = new Rental(rentalId, (int)distance, carId, customerId, dates[0], dates[1]);
                }
                Rentals.Add(r);
                c++;
                double percent = ((double)c / (150 * Cars.Count))*100;
                Console.WriteLine(i +"\t" + item.CarId + "\t" + c + "\t" + percent + "%");
            }
        }
        ExportJson(Rentals);


    }
    private DateTime[] GenerateDate()
    {
        DateTime first = new DateTime(2017, 1, 2, 0, 0, 0);
        DateTime start = first.AddDays(rnd.Next(DateTime.Now.Subtract(first).Days));
        DateTime end = start.AddMinutes(rnd.Next(10, 180));
```

INVOICES

- A tábla tartalma szorosan kötődik a bérlésekhez, ezért itt is a saját programot kellett bővítenem olyan módon, hogy az minden bérléshez hozzárendeljen egy számlát is. Ennek az összege egyelőre üresen maradt, azt egy későbbi fejezetben lekérdezésekkel fogom kiszámolni és frissíteni az adatok alapján. Az ezt megvalósító kódrészlet alább látható.
- A tábla **1.435.800** rekordot tartalmaz, mivel minden bérléshez pontosan egy számlát rendelünk hozzá.

```
static class InvoiceGenerator
{
    static Random rnd = new Random();
    public static List<Invoice> GenerateInvoices(List<Rental> rentals)
    {
        int idx = 100;
        List<Invoice> invoice = new List<Invoice>();
        foreach (var rental in rentals)
        {
            Invoice i = new Invoice()
            {
                InvoiceId = idx++,
                RentalId = rental.RentalId,
                CompletionTime = rental.EndTime.AddMinutes(rnd.Next(1, 15)),
                Amount = 0,
            };
            invoice.Add(i);

        }
        return invoice;
    }
    public static void ExportDSV(List<Invoice> invoices)
    {
        StreamWriter sw = new StreamWriter("invoices.dsv");
        string starterLine = $"\"INVOICE_ID\";\"RENTAL_ID\";\"COMPLETION_TIME\";\"AMOUNT\"";
        sw.WriteLine(starterLine);

        foreach (var item in invoices)
        {
            sw.WriteLine(item);
        }

        sw.Close();
    }
}
```

COMPLAINTS

- A panaszok létrehozásához is a saját programom bővítettem ki, ami így már a bérlésekhez megadott darabszámú panaszt generál.
- A tábla összesen **112.351** rekordot tartalmaz.

```
static class ComplaintGenerator
{
    static string[] detailTypes =
    {
        "Nem megfelelo tisztasagu az auto.",
        "Baleset tortent, szemelyi serules nelkul.",
        "Meghuztak az autot amig parkolt.",
        "Baleset tortent szemelyi serulessel.",
    };
    static Random rnd = new Random();
    public static List<Complaint> GenerateComplaints(List<Rental> rentals, int count)
    {
        int idx = 1;
        int pcs = 0;
        List<Complaint> complaints = new List<Complaint>();
        while (pcs < count)
        {
            Complaint c = new Complaint();
            while (complaints.Contains(c))
            {
                c = new Complaint()
                {
                    RentalId = rentals[rnd.Next(rentals.Count)].RentalId,
                    Details = detailTypes[rnd.Next(detailTypes.Length)],
                };
            }
            if (c.Details != string.Empty)
            {
                c.ComplaintId = idx++;
                complaints.Add(c);
                pcs++;
                Console.WriteLine(pcs);
            }

        }
        return complaints;
    }
```

EGYÉB FUNKCIÓK LÉTREHOZÁSA

Az adatbázishoz létrehoztam szekvenciákat, triggereket és tárolt eljárásokat, amik hasznosak lehetnek, ebben a fejezetben ezekről lesz szó.

SZEKVENCIÁK

Az elsődleges kulcsok későbbi egyszerűbb generálásához szekvenciákat hoztam létre néhány táblához. A szekvenciával ellátott táblák tehát a következők:

- Customers,
- Complaints,
- Employees,
- Jobs,
- Locations,
- Subscriptions.

A szekvenciákat létrehozó script megtalálható a mellékletben.

TÁROLT ELJÁRÁSOK

A bérlések költségeit annak időtartama és az ügyfél előfizetésének díjai alapján számítjuk ki, ezt alapból nem tartalmazza az importált adathalmaz. Ennek kiszámításához létrehoztam egy tárolt eljárást. Az ezt megvalósító script a mellékletben is megtalálható.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE CalculatePrice IS
invoice invoices%ROWTYPE;
BEGIN
FOR invoice IN (SELECT * FROM invoices)
LOOP
UPDATE invoices SET amount =
(SELECT (CAST(end_time AS DATE) - CAST(start_time AS DATE)) * 1440 * minute_price
FROM invoices
INNER JOIN rentals ON (invoices.rental_id = rentals.rental_id)
INNER JOIN customers ON (rentals.customer_id = customers.customer_id)
INNER JOIN subscriptions ON (customers.subscription_id = subscriptions.subscription_id)
WHERE invoices.invoice_id = invoice.invoice_id)
WHERE invoices.invoice_id = invoice.invoice_id;
END LOOP;
dbms_output.put_line('Számlák összege kiszámolva.');
END;

execute CalculatePrice();
```

```
Script Output x
Task completed in 60,473 seconds
Procedure CALCULATEPRICE compiled

PL/SQL procedure successfully completed.

Számlák összege kiszámolva.
```

TRIGGEREK

Az előbbi tárolt eljárás csak a már korábban létrehozott bérlések költségeinek kiszámítására alkalmas, ennek minden beszúráskor történő lefuttatása rendkívül időigényes lenne, mivel önmagában az eljárás percekig futott. Ezért létrehoztam egy triggert, ami ha új bérlés kerül az adatbázisba, elkészíti hozzá a számlát, és egyúttal kiszámolja a bérlés költségét is. A triggert megvalósító script megtalálható a mellékletben is.

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER CreateInvoice
AFTER INSERT ON rentals
FOR EACH ROW
BEGIN
  INSERT INTO INVOICES VALUES(invoices_seq.nextval, :NEW.rental_id, :NEW.end_time,
  (SELECT (CAST(:NEW.end_time AS DATE) - CAST(:NEW.start_time AS DATE)) * 1440 * minute_price
    FROM customers
    INNER JOIN subscriptions ON (customers.subscription_id = subscriptions.subscription_id)
    WHERE customers.customer_id = :NEW.customer_id)
    );
    dbms_output.put_line('A bérléshez tartozó számla létrehozva!');
END;
```

Script Output x | Query Result x

Task completed in 0,014 seconds

```
Trigger CREATEINVOICE compiled
```

LEKÉRDEZÉSEK ELEMZÉSE, OPTIMALIZÁCIÓ

$200 FELETTI BÉRLÉSEK, AHOL PARKOLÁS KÖZBEN SÉRÜLT AZ AUTÓ

Az első lekérdezésben azt szeretnénk megtudni, hogy melyek azok a $200 feletti értékű bérlések és autók, amelyeknél jelentették, hogy az autó megsérült parkolás közben. Ehhez négy táblát kellett összekapcsolni, ezekből kettő is közel másfél millió rekordot tartalmaz, egy pedig több mint százezret. A lekérdezés a következőképp néz ki:

```
SELECT RENTALS.RENTAL_ID, CARS.BRAND, CARS.MODEL, INVOICES.AMOUNT, COMPLAINTS.DETAILS FROM RENTALS
INNER JOIN CARS ON(CARS.CAR_ID = RENTALS.CAR_ID)
INNER JOIN INVOICES ON(RENTALS.RENTAL_ID = INVOICES.RENTAL_ID)
INNER JOIN COMPLAINTS ON(RENTALS.RENTAL_ID = COMPLAINTS.RENTAL_ID)
WHERE INVOICES.AMOUNT > 200
AND COMPLAINTS.DETAILS LIKE '%parkol%';
```

ELEMZÉS

Az adatbázis kezelő a lekérdezést a következő módon hajtja végre:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		4471	9264
HASH JOIN		4471	9264
Access Predicates			
CARS.CAR_ID=RENTALS.CAR_ID			
TABLE ACCESS (FULL)	CARS	9572	13
HASH JOIN		4471	9251
Access Predicates			
RENTALS.RENTAL_ID=INVOICES.RENTAL_ID			
TABLE ACCESS (FULL)	INVOICES	223306	1751
Filter Predicates			
INVOICES.AMOUNT>200			
HASH JOIN		28939	6655
Access Predicates			
RENTALS.RENTAL_ID=COMPLAINTS.RENTAL_ID			
NESTED LOOPS		28939	6655
NESTED LOOPS			
STATISTICS COLLECTOR			
TABLE ACCESS (FULL)	COMPLAINTS	28939	239
Filter Predicates			
AND			
COMPLAINTS.DETAILS IS NOT NULL			
COMPLAINTS.DETAILS LIKE '%parkol%'			
INDEX (UNIQUE SCAN)	PK_RENTALS_RENTAL_ID		
Access Predicates			
RENTALS.RENTAL_ID=COMPLAINTS.RENTAL_ID			
TABLE ACCESS (BY INDEX ROWID)	RENTALS	1	2469
TABLE ACCESS (FULL)	RENTALS	1445404	2469

- Elsőként egy hash joint hajt végre a bérlések és a panaszok táblára. Azért ezt a műveletet választotta az adatbázis kezelő, mert a panaszok tábla logikusan jóval kisebb, mint a bérlések, így ez az optimális művelet. Két nested loop is felhasználásra kerül, a belsőben full table scan segítségével beolvassuk a panaszok tábla adatait, a feltételnek (a panasz szövege tartalmazza a parkol szót) megfelelő rekordokat filter predicates választja ki. A külső nested loop ezután full table scan használatával beolvassa a bérlések tábla adatait, és végül a hash join összekapcsolja a két tábla feltételnek megfelelő rekordjait, ehhez access predicates kerül felhasználásra, ami a két tábla azonosítóit köti össze. Látható, hogy a szűrésnek köszönhetően az adatbázis kezelő jelentősen alacsonyabb kardinalitást becsül, mint a panaszok tényleges száma.

- Ezt követően végrehajtunk egy másik hash joint a számlák és a korábban kiválasztott bérlések között. Ez is full table scan segítségével történik, a feltételnek (a számla összege nagyobb mint 200) megfelelő rekordok szűrését filter predicates használatával végezzük el. Ezután végrehajtjuk a hash joint a bérlések és a számlák között, ami access predicates-t használ a két tábla összekapcsolására.
- Végül egy újabb hash join segítségével összekapcsoljuk az autókat is az eddigiekkel, amihez full table scant használ az adatbázis kezelő.
- Az adatbázis kezelő a lekérdezés kardinalitását 4471-re becsüli, a tényleges pedig 4294. A lekérdezés költsége 9264.

OPTIMALIZÁCIÓ

A lekérdezés optimalizálásához létrehoztam három indexet:

- az ***Invoices*** tábla *rental_id* és *amount* mezőjére,
- a ***Complaints*** tábla *rental_id* és *details* mezőjére,
- valamint a ***Rentals*** tábla *rental_id* és *car_id* mezőjére.

Ezt követően a végrehajtási terv a következőképp néz ki:

- Az előzőhez hasonlóan itt is hash joint hajtunk végre először a bérlések és panaszok táblák között, azonban a full table scan helyett kihasználjuk a létrehozott indexet, ami itt kis mértékben csökkentette a költséget.
- Ezt követően egy újabb hash join következik, a bérlések és a számlák között. Ez a létrehozott indexnek köszönhetően nem full table scant használ, így csökkenteni tudtuk a költséget ezen a részen is.
- Végül az autókat összekapcsoljuk a bérlésekkel, itt nem tudtam gyorsítani, ugyanúgy full table scant használ.
- Összesítve, a költséget sikerült **9264**-ről **7778**-ra csökkenteni az optimalizáció után.

BEVÉTELEK ÖSSZEGE AUTÓNKÉNT CSOPORTOSÍTVA

A második lekérdezésben azt szeretnénk megtudni, hogy melyik városból összesen mennyi bevétel érkezett, csak azokat a bérléseket vizsgáljuk, amikkel kapcsolatban nem érkezett bejelentés, és az eredményt a bérlések száma szerint csökkenően rendezzük. A lekérdezést a következő módon írtam meg:

```
SELECT CITY, SUM(AMOUNT) FROM RENTALS
INNER JOIN INVOICES ON (RENTALS.RENTAL_ID = INVOICES.RENTAL_ID)
INNER JOIN CUSTOMERS ON(CUSTOMERS.CUSTOMER_ID = RENTALS.CUSTOMER_ID)
LEFT JOIN COMPLAINTS ON (COMPLAINTS.RENTAL_ID = RENTALS.RENTAL_ID)
WHERE COMPLAINTS.RENTAL_ID IS NULL
GROUP BY CITY ORDER BY SUM(AMOUNT) DESC
```

ELEMZÉS

Az adatbázis kezelő a következő tervet javasolta a lekérdezés végrehajtására:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1335602	20582
SORT (ORDER BY)		1335602	20582
HASH (GROUP BY)		1335602	20582
HASH JOIN		1335602	20518
Access Predicates			
CUSTOMERS.CUSTOMER_ID=RENTALS.CUSTOMER_ID			
TABLE ACCESS (FULL)	CUSTOMERS	14001	68
HASH JOIN (RIGHT ANTI)		1445405	15079
Access Predicates			
COMPLAINTS.RENTAL_ID=RENTALS.RENTAL_ID			
TABLE ACCESS (FULL)	COMPLAINTS	116478	239
HASH JOIN		1445405	10322
Access Predicates			
RENTALS.RENTAL_ID=INVOICES.RENTAL_ID			
TABLE ACCESS (FULL)	RENTALS	1445404	2469
TABLE ACCESS (FULL)	INVOICES	1507721	1749

- Először egy hash join kerül végrehajtásra a bérlések és számlák tábla között. Ehhez full table scant használ az adatbázis kezelő mindkét táblán, és az access predicates a *rental_id* mezők értéke alapján kapcsolja össze a két táblát.
- Ezt követően a külső hash join következik, ami szintén full table scant használ, és az eddigi eredményt összekapcsolja a panaszok táblával. A kardinalitás csökken, mivel azokat a bérléseket, amikhez tartozik panasz, eldobjuk.
- Egy újabb hash join összekapcsolja az eddigi eredményeket a vásárlók táblával, amihez full table scant használ.
- Ezután egy hash group by művelet segítségével csoportosítjuk az értékeket a városok szerint, végül a sort order by használatával csökkenőbe rendezzük az eredményeket.
- A lekérdezés költsége **20.582**.

OPTIMALIZÁCIÓ

A lekérdezés optimalizálásához indexeket hoztam létre:

- a **Complaints** tábla *rental_id* mezőjére,
- valamint az **Invoices** tábla *rental_id* mezőjére.

Az optimalizáció után a következő végrehajtási tervet javasolta az adatbázis kezelő:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1335602	20419
SORT (ORDER BY)		1335602	20419
HASH (GROUP BY)		1335602	20419
HASH JOIN		1335602	20355
Access Predicates			
CUSTOMERS.CUSTOMER_ID=RENTALS.CUSTOMER_ID			
TABLE ACCESS (FULL)	CUSTOMERS	14001	68
HASH JOIN (RIGHT ANTI)		1445405	14916
Access Predicates			
COMPLAINTS.RENTAL_ID=RENTALS.RENTAL_ID			
INDEX (FAST FULL SCAN)	COMPLAINT_RENTAL_IDX	116478	75
HASH JOIN		1445405	10322
Access Predicates			
RENTALS.RENTAL_ID=INVOICES.RENTAL_ID			
NESTED LOOPS		1445405	10322
NESTED LOOPS			
STATISTICS COLLECTOR			
TABLE ACCESS (FULL)	RENTALS	1445404	2469
INDEX (RANGE SCAN)	INVOICE_RENTAL_ID		
Access Predicates			
RENTALS.RENTAL_ID=INVOICES.RENTAL_ID			
TABLE ACCESS (BY INDEX ROWID)	INVOICES	1	1749
TABLE ACCESS (FULL)	INVOICES	1507721	1749

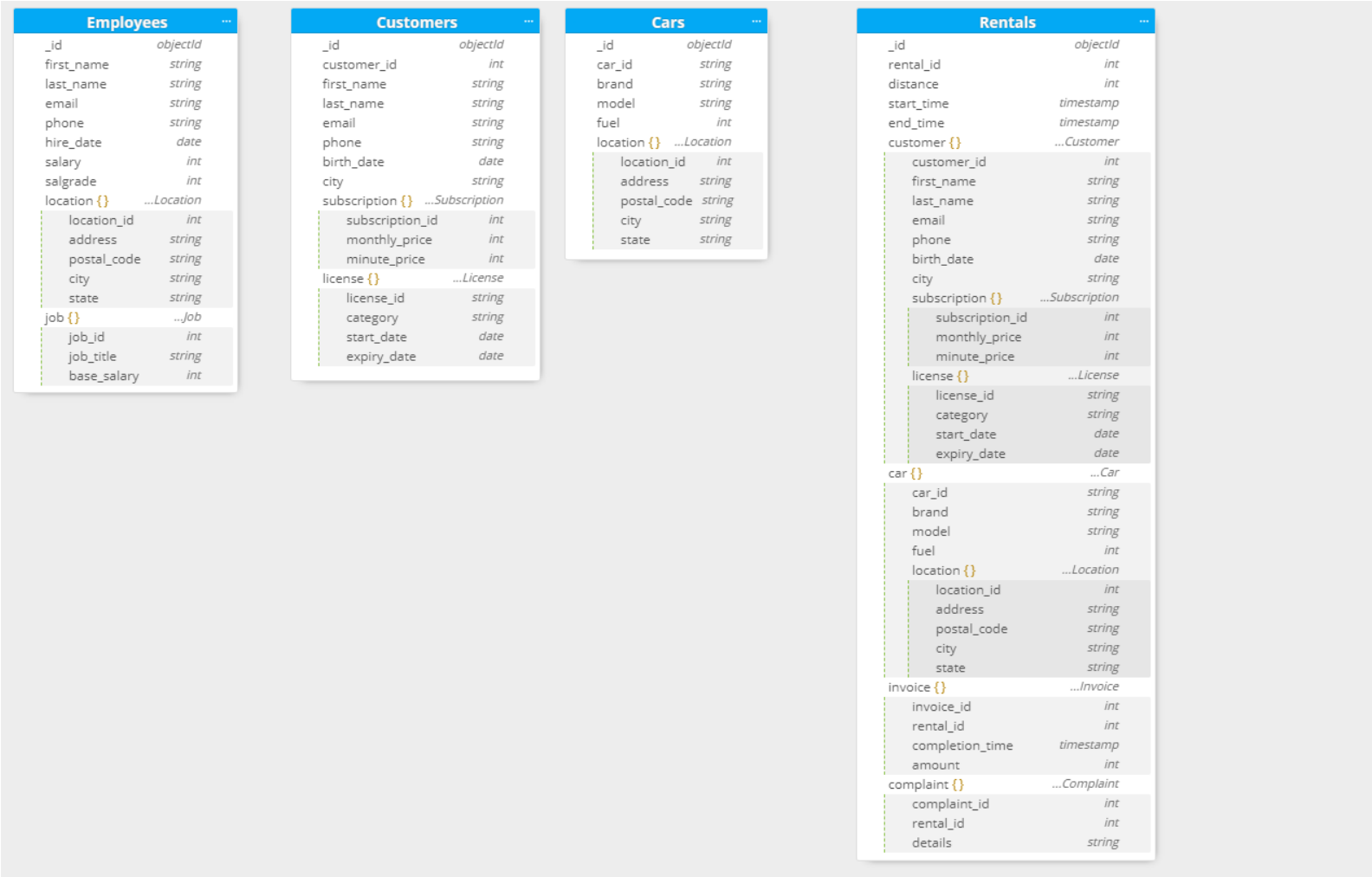
- Elsőként egy hash join műveletet hajtunk végre a bérlések és a számlák tábla között, itt az előző végrehajtási tervhez képest változás, hogy ezúttal két nested loop is létrejött, valamint a számlákon már index range scan használatával megy végig, ami némileg javít a teljesítményen.
- Ezután egy újabb hash join következik, ami azokat a rekordokat válogatja ki a bérlések közül, amelyekhez nem tartozik panasz. Az előzőhöz képest annyi eltérés tapasztalható, hogy itt is használja a létrehozott indexet, és fast full scan használatával végzi el a műveletet.
- Ezt követően a külső hash join összekapcsolja a vásárlók táblával, ehhez full table scant használ, ezen a részen nem történt változás.
- A költséget kis mértékben sikerült csökkenteni, így az **20.419** lett.

NOSQL ADATBÁZIS KEZELÉS

A NoSQL adatbázis kiválasztásakor a ***MongoDB***-re esett a választásom. Ennek oka, hogy az adatbázisban tárolt adatok szerkezete inkább dokumentum jellegű, így ennek használatát tartom célszerűnek.

AZ ADATBÁZIS TÁBLÁINAK ÁTALAKÍTÁSA

Az alábbi képen, valamint az alatta lévő bekezdésben látható magyarázat arról, hogy milyen változtatásokat hajtottam végre az adattáblákon.

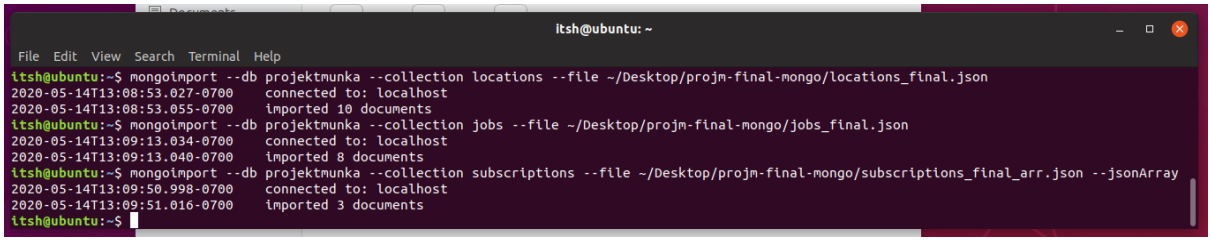


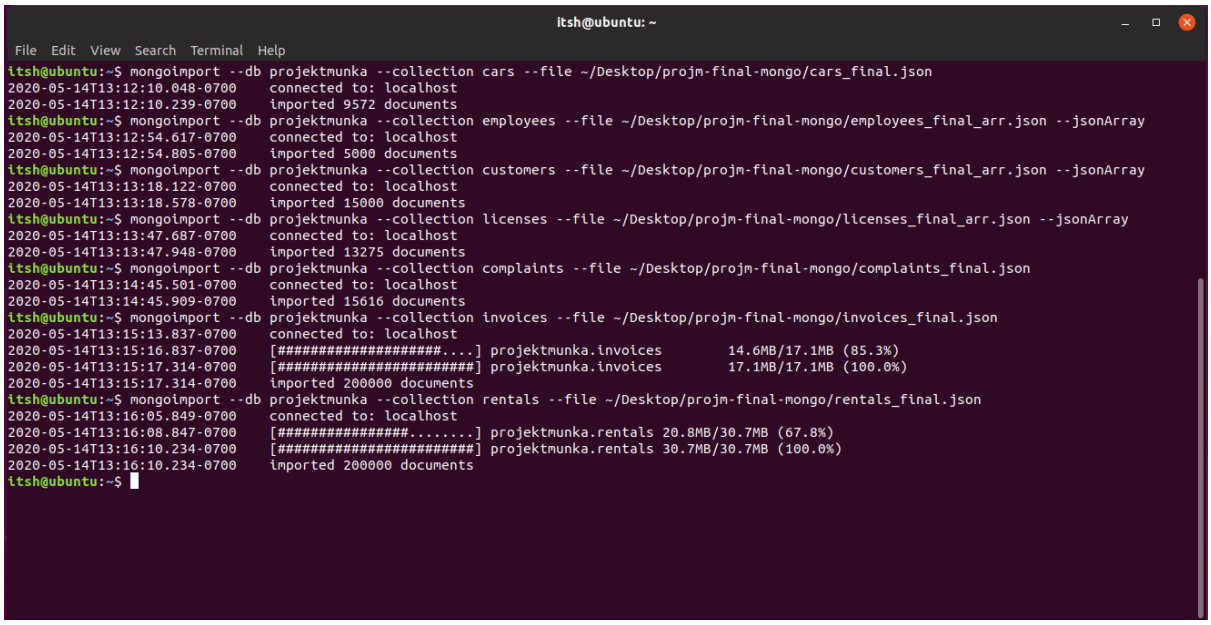
Ahogyan a képen látható, kiválasztottam azokat a táblákat amiknek önmagukban is van jelentőségük, nem csak egy másik táblával együtt értelmezhetők, ezek lesznek a gyűjtemények. Azok a táblák pedig, amik önmagukban nem hasznosak, beágyazott dokumentumként fognak megjelenni az adatbázisban. Erre egy példa, hogy az ügyfél előfizetésének, valamint jogosítványának adatai ezentúl nem alkotnak önálló táblát, hanem

be lesznek ágyazva az ügyfeleket tartalmazó gyűjteménybe. Ugyanígy döntöttem a helyszín és a munkakör esetében is, ugyanis ezek is nagyon ritkán módosulnak, és önmagukban nem használjuk őket. A panaszokat és számlákat tartalmazó táblák szintén beágyazott dokumentumként lesznek megvalósítva, ezeket mindig csak az adott bérléssel kapcsolatosan vizsgáljuk, önmagukban nem, valamint szintén nagyon ritkán módosulnak, így célszerű a bérléssel azonos dokumentumban tárolni. Ezen felül minden bérléshez eltároljuk az azt kezdeményező ügyfelet, valamint az ahhoz használt autót is beágyazott dokumentumként.

AZ ADATOK ÁTVITELE MONGODB-BE

A gyűjtemények szerkezetének megtervezése után importálom az adatokat. Ehhez felhasználtam az ***SQL Developer*** exportálás funkcióját, ami a táblákban található adatokat ***json*** formátumban elmentette, amiket így egyszerűen tudtam importálni ***MongoDB***-be.





Az adatok importálását követően a ***MongoDB*** aggregációs függvényeivel, a *lookup stage* segítségével kialakítottam a fentebb meghatározott szerkezetet minden gyűjtemény esetében. A bérlések és számlák táblából csak az első ***200.000*** rekordot hoztam át, mivel ez még meghaladja a követelmények szerinti adatmennyiséget, viszont a teljes táblára az aggregáció túl sok ideig tartott, illetve lefagyott a gép.

```
db.employees.aggregate([
{
     $lookup:{
          from: "locations",
          localField: "location_id",
          foreignField: "location_id",
          as: "location"
          }
     },
{
     $lookup: {
          from: "jobs",
          localField: "job_id",
          foreignField: "job_id",
          as: "job"
          }
     },
{
     $out: "employees"
     }
]);
```

employees 0.502 sec.

```
Fetched 0 record(s) in 0ms
```

```
db.customers.aggregate([
{
      $lookup:{
           from: "subscriptions",
           localField: "subscription_id",
           foreignField: "subscription_id",
           as: "subscription"
           }
      },
{
      $lookup: {
           from: "licenses",
           localField: "customer_id",
           foreignField: "customer_id",
           as: "license"
           }
      },
      {
           $out: "customers"
           }
]);
```

customers 53.7 sec.

```
Fetched 0 record(s) in 0ms
```

```
db.rentals.aggregate([
{
    $lookup:{
        from: "cars",
        localField: "car_id",
        foreignField: "car_id",
        as: "car"
        }
    },
{
    $lookup: {
        from: "customers",
        localField: "customer_id",
        foreignField: "customer_id",
        as: "customer"
        }
    },
{
    $lookup: {
        from: "invoices",
        localField: "invoice_id",
        foreignField: "invoice_id",
        as: "invoice"
        }
    },
{
    $lookup: {
        from: "complaints",
        localField: "complaint_id",
        foreignField: "complaint_id",
        as: "complaint"
        }
    },
    {
        $out: "rentals"
        }
]);
```

```
db.cars.aggregate([
{
    $lookup:{
        from: "locations",
        localField: "location_id",
        foreignField: "location_id",
        as: "location"
        }
    },
{
    $out: "cars"
    }
]);
```

cars 0.261 sec.

```
Fetched 0 record(s) in 0ms
```

Ezt követően a felesleges táblákat és a beágyazott dokumentumok miatt már nem használt mezőket eltávolítottam, így a dokumentumok a következőképp néznek ki:

```
db.getCollection('cars').find({})
```

cars 0.001 sec.

Key	Value	Type
(1) ObjectId("5ebaa5b6a86f363c5334517b")	{ 6 fields }	Object
_id	ObjectId("5ebaa5b6a86f363c5334517b")	ObjectId
car_id	2SUH945	String
brand	Volkswagen	String
model	e-Up	String
fuel	39	Int32
location	[1 element]	Array

```
db.getCollection('customers').find({})
```

customers 0.001 sec.

Key	Value	Type
(1) ObjectId("5ebab311a86f363c5362b6b8")	{ 10 fields }	Object
_id	ObjectId("5ebab311a86f363c5362b6b8")	ObjectId
customer_id	1	Int32
first_name	Ronald	String
last_name	Fain	String
email	Rosia.Smalley4@example.com	String
phone	(396) 475-6485	String
birth_date	1990-04-14	String
city	Bartonsville	String
subscription	[1 element]	Array
licenses	[1 element]	Array

```
db.getCollection('employees').find({})
```

employees 0.001 sec.

Key	Value	Type
(1) ObjectId("5eba5fdea86f363c533434cf")	{ 11 fields }	Object
_id	ObjectId("5eba5fdea86f363c533434cf")	ObjectId
employee_id	5000	Int32
first_name	Samara	String
last_name	Logue	String
email	Samara.Logue@gmail.com	String
phone	(969) 319-0579	String
hire_date	2003-01-10	String
salary	4862.8786049894	Double
salgrade	4.12108356355034	Double
location	[1 element]	Array
job	[1 element]	Array

```
db.getCollection('rentals').find({})
```

rentals 0.001 sec.

Key	Value	Type
(1) ObjectId("5ebb2aea814105fd141fdf78")	{ 9 fields }	Object
_id	ObjectId("5ebb2aea814105fd141fdf78")	ObjectId
rental_id	199844	Int32
distance	40	Int32
start_time	2017-07-24 03:29:00	String
end_time	2017-07-24 04:26:00	String
complaint	[0 elements]	Array
car	[1 element]	Array
invoice	[1 element]	Array
customer	[1 element]	Array

LEKÉRDEZÉSEK

A lekérdezések szöveges formátumban megtalálhatóak a mellékletben.

1. HÁNY DOLGOZÓ VAN AKINEK A FIZETÉSE $2000 ALATTI?

```
db.employees.find({salary: {$lte: 2000}}).count();
```

0.003 sec.

```
834
```

2. MELYIK VÁROSOKBAN VANNAK A VÁLLALATNAK AUTÓI?

```
db.cars.distinct("location.city");
```

0.006 sec.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	Emigsville	Waukena	Abbotsford	Zuni	Brockport	Springboro	Abbeville	Brockton

3. MELYIK TELEPHELYEN DOLGOZIK A LEGTÖBB ALKALMAZOTT, ÉS MENNYI?

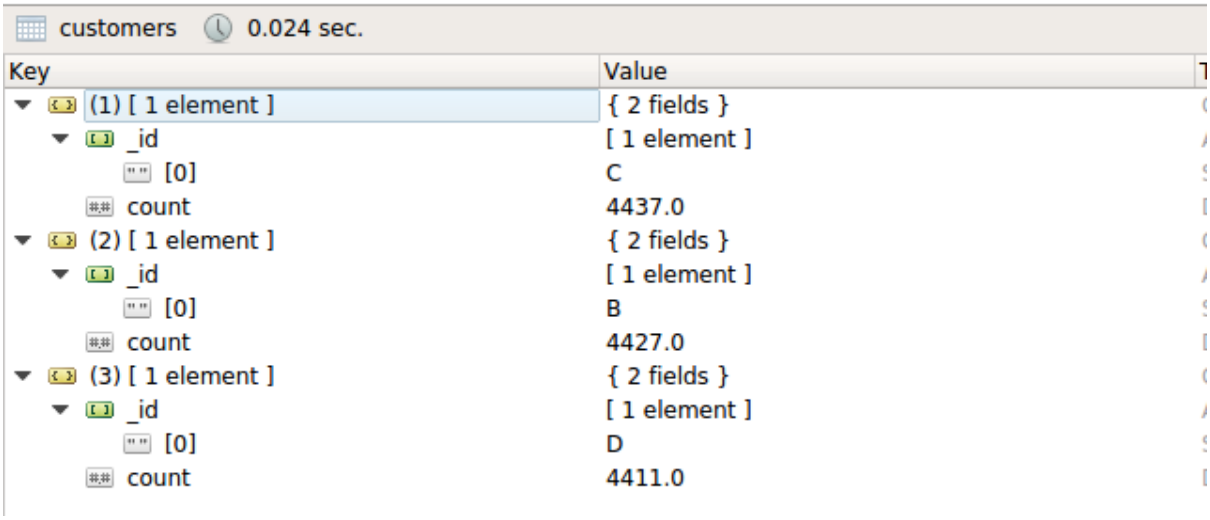
```
db.employees.aggregate(
[
    {$group: {_id:"$location", count: {$sum:1}}},
    {$sort: {count: -1}},
    {$limit:1}
]
);
```

employees 0.011 sec.

Key	Value	Type
(1) [1 element]	{ 2 fields }	Object
_id	[1 element]	Array
[0]	{ 6 fields }	Object
_id	ObjectId("5eb978b804da7b4b969fba49")	ObjectId
location_id	9	Int32
address	1808 Market Hwy	String
postal_code	96299	String
city	Brockton	String
state	Delaware	String
count	616.0	Double

4. HÁNY DARAB JOGOSÍTVÁNY VAN KATEGÓRIÁNKÉNT?

```
db.customers.aggregate([
{$match: {"licenses.license_id": {$exists: true}}},
{$group:{_id:"$licenses.category", count: {$sum:1}}},
{$sort: {count: -1}}
]);
```

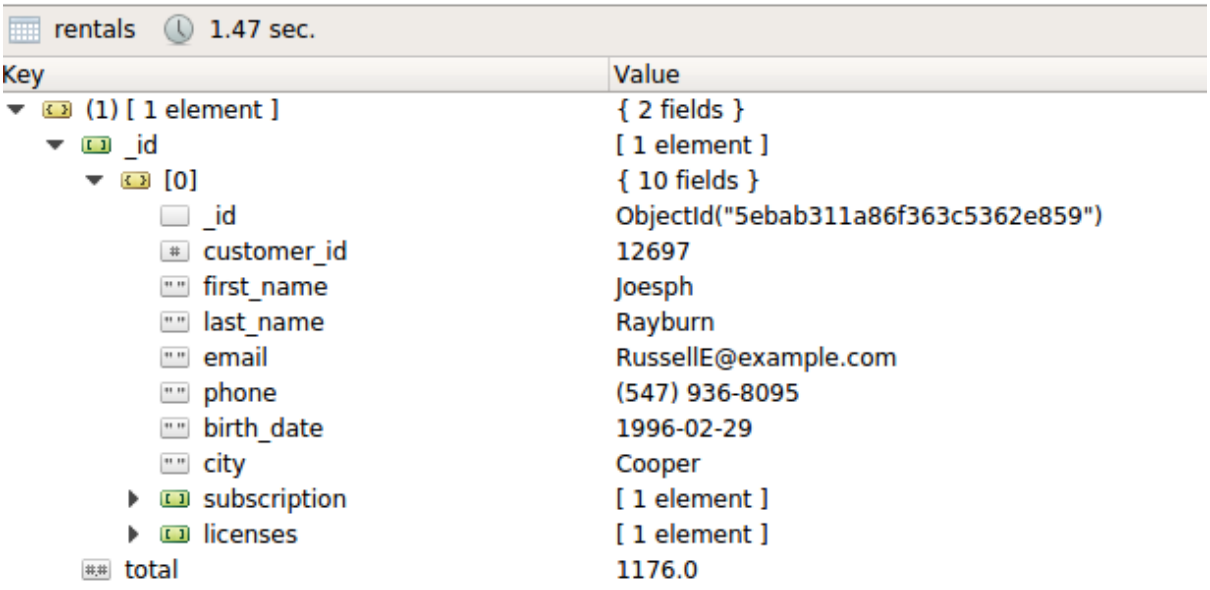


customers 0.024 sec.

Key	Value
(1) [1 element]	{ 2 fields }
_id	[1 element]
[0]	C
count	4437.0
(2) [1 element]	{ 2 fields }
_id	[1 element]
[0]	B
count	4427.0
(3) [1 element]	{ 2 fields }
_id	[1 element]
[0]	D
count	4411.0

5. MELYIK ÜGYFÉL FIZETETT A LEGTÖBBET, ÉS MENNYIT?

```
db.rentals.aggregate([
{$unwind: "$invoice"},
{$group: { _id:"$customer", total: {$sum:"$invoice.amount"}}},
{$sort: {total:-1}},
{$limit: 1}
]);
```



rentals 1.47 sec.

Key	Value
(1) [1 element]	{ 2 fields }
_id	[1 element]
[0]	{ 10 fields }
_id	ObjectId("5ebab311a86f363c5362e859")
customer_id	12697
first_name	Joesph
last_name	Rayburn
email	RussellE@example.com
phone	(547) 936-8095
birth_date	1996-02-29
city	Cooper
subscription	[1 element]
licenses	[1 element]
total	1176.0

6. MELYIK A TOP 3 TELEPHELY ÁTLAGFIZETÉS SZEMPONTJÁBÓL?

```
db.employees.aggregate([
{$unwind: "$location"},
{$group: { _id:"$location", avgSalary: {$avg:"$salary"}}},
{$sort: {avgSalary:-1}},
{$limit: 3}
]);
```

employees 0.014 sec.

Key	Value
(1) { 6 fields }	{ 2 fields }
_id	{ 6 fields }
_id	ObjectId("5eb978b804da7b4b969fba49")
location_id	9
address	1808 Market Hwy
postal_code	96299
city	Brockton
state	Delaware
avgSalary	4671.03311603056
(2) { 6 fields }	{ 2 fields }
_id	{ 6 fields }
_id	ObjectId("5eb978b804da7b4b969fba4d")
location_id	3
address	3017 Pine Tree Parkway
postal_code	35616
city	Zuni
state	South Carolina
avgSalary	4640.88856941663
(3) { 6 fields }	{ 2 fields }
_id	{ 6 fields }
_id	ObjectId("5eb978b804da7b4b969fba46")
location_id	6
address	2381 Front Blvd
postal_code	95270
city	Brockport
state	Michigan
avgSalary	4632.26883876419

7. HÁNYAN VANNAK AKIK A HAVIDÍJ NÉLKÜLI ELŐFIZETÉST VÁLASZTOTTÁK?

```
db.getCollection('customers').find({"subscription.monthly_price":0}).count();
```

0.011 sec.

```
4917
```

8. MELYIK A KÉT LEGNÉPSZERŰBB MUNKAKÖR, ÉS HÁNYAN VÉGZIK AZT?

```
db.employees.aggregate([
{$group:{_id:"$job.job_title", countOfEmployees:{$sum:1}}},
{$sort: {countOfEmployees: -1}},
{$limit: 2}
]);
```

employees 0.009 sec.

Key	Value
(1) [1 element]	{ 2 fields }
_id	[1 element]
[0]	Maintenance
countOfEmployees	1304.0
(2) [1 element]	{ 2 fields }
_id	[1 element]
[0]	Human Resources Representative
countOfEmployees	633.0

9. MELYIK HÁROM AUTÓVAL TETTÉK MEG A LEGTÖBB UTAT, ÉS MENNYIT?

```
db.rentals.aggregate([
{$unwind: "$invoice"},
{$group:{_id:"$car", total:{$sum:"$distance"}}},
{$sort: {total: -1}},
{$limit: 3}
]);
```

rentals 1.07 sec.

Key	Value
_id	[1 element]
[0]	{ 6 fields }
_id	ObjectId("5ebaa5b6a86f363c53345328")
car_id	1JYO392
brand	BMW
model	i3
fuel	52
location	[1 element]
total	11601
(2) [1 element]	{ 2 fields }
_id	[1 element]
[0]	{ 6 fields }
_id	ObjectId("5ebaa5b6a86f363c533451bf")
car_id	3YFL961
brand	Nissan
model	Leaf
fuel	21
location	[1 element]
total	11441
(3) [1 element]	{ 2 fields }
_id	[1 element]
[0]	{ 6 fields }
_id	ObjectId("5ebaa5b6a86f363c53345421")
car_id	0CHK299
brand	Volkswagen
model	e-Up
fuel	49
location	[1 element]
total	11346

10. MELYIK HÁROM VÁROSBAN LAKÓK INDÍTOTTÁK A LEGTÖBB BÉRLÉST?

```
db.rentals.aggregate([
{$group:{_id:"$customer.city", countOfRentals:{$sum: 1}}},
{$sort: {countOfRentals:-1}},
{$limit: 3}
]);
```

rentals 0.507 sec.

Key	Value
(1) [1 element]	{ 2 fields }
_id	[1 element]
[0]	Mount Pocono
countOfRentals	47.0
(2) [1 element]	{ 2 fields }
_id	[1 element]
[0]	Holly Springs
countOfRentals	46.0
(3) [1 element]	{ 2 fields }
_id	[1 element]
[0]	Lake Peekskill
countOfRentals	46.0

MELLÉKLETEK

TÁBLÁKAT LÉTREHOZÓ SCRIPT

```
1.  CREATE TABLE Locations (
2.   location_id NUMBER,
3.   address     VARCHAR2(255),
4.   postal_code VARCHAR2(24),
5.   city          VARCHAR2(255),
6.   state       VARCHAR2(50),
7.   CONSTRAINT PK_LOCATIONS_LOCATION_ID PRIMARY KEY (location_id)
8. );
9.
10.  CREATE TABLE Jobs (
11.    job_id      NUMBER,
12.    job_title   VARCHAR2(50),
13.    base_salary NUMBER,
14.    CONSTRAINT PK_JOBS_JOB_ID PRIMARY KEY (job_id)
15.  );
16.
17.  CREATE TABLE Employees (
18.    employee_id NUMBER,
19.    first_name  VARCHAR2(255),
20.    last_name   VARCHAR2(255),
21.    email       VARCHAR2(100),
22.    phone       VARCHAR2(20),
23.    location_id NUMBER,
24.    hire_date   DATE,
25.    job_id      NUMBER,
26.    salary      NUMBER,
27.    salgrade    NUMBER,
28.    CONSTRAINT PK_EMPLOYEES_EMPLOYEE_ID PRIMARY KEY (employee_id),
29.    CONSTRAINT FK_EMPLOYEES_LOCATION_ID FOREIGN KEY (location_id)
30.    REFERENCES Locations (location_id),
31.    CONSTRAINT FK_EMPLOYEES_JOB_ID FOREIGN KEY (job_id)
32.    REFERENCES Jobs (job_id)
33.  );
34.
35.  CREATE TABLE Cars (
36.    car_id      VARCHAR2(50),
37.    brand       VARCHAR2(50),
38.    model       VARCHAR2(50),
39.    fuel        NUMBER,
40.    location_id NUMBER,
41.    CONSTRAINT PK_CARS_CAR_ID PRIMARY KEY (car_id),
42.    CONSTRAINT FK_CARS_LOCATION_ID FOREIGN KEY (location_id)
43.    REFERENCES Locations (location_id)
44.  );
45.
46.  CREATE TABLE Subscriptions (
47.    subscription_id NUMBER,
48.    monthly_price   NUMBER,
49.    minute_price    NUMBER,
50.    CONSTRAINT PK_SUBSCRIPTIONS_SUBSCRIPTION_ PRIMARY KEY
   (subscription_id)
51.  );
52.
```

```
53.  CREATE TABLE Customers (
54.    customer_id     NUMBER,
55.    subscription_id NUMBER,
56.    first_name      VARCHAR2(255),
57.    last_name       VARCHAR2(255),
58.    email           VARCHAR2(100),
59.    phone           VARCHAR2(20),
60.    birth_date      DATE,
61.    city            VARCHAR2(255),
62.    CONSTRAINT PK_CUSTOMERS_CUSTOMER_ID PRIMARY KEY (customer_id),
63.    CONSTRAINT FK_CUSTOMERS_SUBSCRIPTION_ID FOREIGN KEY
   (subscription_id)
64.    REFERENCES Subscriptions (subscription_id)
65.  );
66.
67.  CREATE TABLE Licenses (
68.    license_id  VARCHAR2(50),
69.    customer_id NUMBER,
70.    category    VARCHAR2(4),
71.    start_date  DATE,
72.    expiry_date DATE,
73.    CONSTRAINT PK_LICENSES_LICENSE_ID PRIMARY KEY (license_id),
74.    CONSTRAINT FK_LICENSES_CUSTOMER_ID FOREIGN KEY (customer_id)
75.    REFERENCES Customers (customer_id)
76.  );
77.
78.  CREATE TABLE Rentals (
79.    rental_id   NUMBER,
80.    distance    NUMBER,
81.    car_id      VARCHAR2(50),
82.    customer_id NUMBER,
83.    start_time  TIMESTAMP(0),
84.    end_time    TIMESTAMP(0),
85.    CONSTRAINT PK_RENTALS_RENTAL_ID PRIMARY KEY (rental_id),
86.    CONSTRAINT FK_RENTALS_CUSTOMER_ID FOREIGN KEY (customer_id)
87.    REFERENCES Customers (customer_id),
88.    CONSTRAINT FK_RENTALS_CAR_ID FOREIGN KEY (car_id)
89.    REFERENCES Cars (car_id)
90.  );
91.
92.  CREATE TABLE Invoices (
93.    invoice_id      NUMBER,
94.    rental_id       NUMBER ,
95.    completion_time TIMESTAMP(0),
96.    amount          NUMBER,
97.    CONSTRAINT PK_INVOICES_INVOICE_ID PRIMARY KEY (invoice_id),
98.    CONSTRAINT FK_INVOICES_RENTAL_ID FOREIGN KEY (rental_id)
99.    REFERENCES Rentals (rental_id)
100. );
101.
102. CREATE TABLE Complaints (
103.   complaint_id NUMBER,
104.   rental_id    NUMBER,
105.   details      VARCHAR2(512),
106.   CONSTRAINT PK_COMPLAINTS_COMPLAINT_ID PRIMARY KEY (complaint_id),
107.   CONSTRAINT FK_COMPLAINTS_RENTAL_ID FOREIGN KEY (rental_id)
108.   REFERENCES Rentals (rental_id)
109. );
```

SEQUENCE

```
1. CREATE SEQUENCE complaints_seq
2. START WITH 1
3. INCREMENT BY 1
4. NOCACHE
5. NOCYCLE;
6.
7. CREATE SEQUENCE customers_seq
8. START WITH 1
9. INCREMENT BY 1
10.  NOCACHE
11.  NOCYCLE;
12.
13.  CREATE SEQUENCE employees_seq
14.  START WITH 1
15.  INCREMENT BY 1
16.  NOCACHE
17.  NOCYCLE;
18.
19.  CREATE SEQUENCE invoices_seq
20.  START WITH 1435900
21.  INCREMENT BY 1
22.  NOCACHE
23.  NOCYCLE;
24.
25.  CREATE SEQUENCE jobs_seq
26.  START WITH 1
27.  INCREMENT BY 1
28.  NOCACHE
29.  NOCYCLE;
30.
31.  CREATE SEQUENCE locations_seq
32.  START WITH 1
33.  INCREMENT BY 1
34.  NOCACHE
35.  NOCYCLE;
36.
37.  CREATE SEQUENCE subscriptions_seq
38.  START WITH 1
39.  INCREMENT BY 1
40.  NOCACHE
41.  NOCYCLE;
```

PROCEDURE

```
1. SET SERVEROUTPUT ON;
2. CREATE OR REPLACE PROCEDURE CalculatePrice IS
3. invoice invoices%ROWTYPE;
4. BEGIN
5. FOR invoice IN (SELECT * FROM invoices)
6. LOOP
7. UPDATE invoices SET amount =
8. (SELECT (CAST(end_time AS DATE) - CAST(start_time AS DATE)) * 1440 *
   minute_price
9. FROM invoices
10.  INNER JOIN rentals ON (invoices.rental_id = rentals.rental_id)
11.  INNER JOIN customers ON (rentals.customer_id =
   customers.customer_id)
12.  INNER JOIN subscriptions ON (customers.subscription_id =
   subscriptions.subscription_id)
13.  WHERE invoices.invoice_id = invoice.invoice_id)
14.  WHERE invoices.invoice_id = invoice.invoice_id;
15.  END LOOP;
16.  dbms_output.put_line('Számlák összege kiszámolva.');
17.  END;
18.
19.  execute CalculatePrice();
```

TRIGGER

```
1. SET SERVEROUTPUT ON;
2. CREATE OR REPLACE TRIGGER CreateInvoice
3. AFTER INSERT ON rentals
4. FOR EACH ROW
5. BEGIN
6.    INSERT INTO INVOICES VALUES(invoices_seq.nextval, :NEW.rental_id,
   :NEW.end_time,
7.    (SELECT (CAST(:NEW.end_time AS DATE) - CAST(:NEW.start_time AS
   DATE)) * 1440 * minute_price
8.      FROM customers
9.      INNER JOIN subscriptions ON (customers.subscription_id =
   subscriptions.subscription_id)
10.       WHERE customers.customer_id = :NEW.customer_id)
11.       );
12.       dbms_output.put_line('A bérléshez tartozó számla létrehozva!');
13.  END;
```

ADATOK GENERÁLÁSÁHOZ HASZNÁLT SAJÁT PROGRAM

Az adatok generálásához készített programom forráskódja itt elérhető.