Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TOPIAS KUOSMANEN

Relaatiotietokannan suunnittelu ja toteutus jalkapallotilastoille

TIETOJENKÄSITTELYN KOULUTUSOHJELMA
2021

Tekijä(t)
Kuosmanen, Topias

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
11/2021

Sivumäärä 45

Julkaisun kieli
suomi

Julkaisun nimi
Relaatiotietokannan suunnittelu ja toteutus jalkapallotilastoille

Tutkinto-ohjelma
Tietojenkäsittely

Tiivistelmä
Työssä suunniteltiin ja toteutettiin relaatiotietokanta jalkapallo-otteluista ja pelaajista kerätylle datalle. Ensin kerrottiin teoriaa tietokannoista, niiden suunnittelusta liittyen ER-kaavioon ja normalisointiin sekä työssä käytetyistä tekniikoista, kuten SQL-kielestä, SQL Serveristä ja Python-kielestä. Itse toteutuneesta tietokannasta on kirjoitettu ensin ER-kaavio ja tietokantakaava, jonka jälkeen esitellään taulujen luominen, datan käsittely ja sen vieminen tietokantatauluihin.

Tulevaisuudessa tietokantaa olisi tarkoitus käyttää web-sivulla, jossa olisi tietoja otteluista ja pelaajista koottuna esimerkiksi kausittain ja sarjatasoittain. Web-sivua varten ajatuksena olisi rakentaa käyttöliittymä, jolla tietoja saa lisättyä myös tulevaisuuden otteluista ja pelaajista. Lisäksi suunnitelmissa on kerätä dataa sellaista asioista, joita nykyisessä tietokannassa ei ole, kuten esimerkiksi peliminuuteista, korteista ja maalisyötöistä.

Asiasanat
tietokanta, relaatiotietokanta, käsitemallinnus, SQL

Author(s)
Kuosmanen, Topias

Type of Publication
Bachelor's thesis

Date
11/2021

Number of pages
45

Language of publication:
Finnish

Title of publication
Designing and creating a relational database for football statistics

Degree programme
Business information

Abstract
This work was about designing and creating a relational database for data which was collected about football players and games. The objective of the beginning of this text was to acquaint the reader with databases and their designing and tools. Designing part includes theory about entity modeling and normalization. Theory of tools includes information about SQL language, SQL Server and Python language. After the beginning there is presented databases entity diagram and database formula. The end of the thesis presents the creation of the database tables, data manipulation and exporting data to final database tables.

In the future, the database should be used on a web site, which would present the data about games and players seasonally and by league levels for example. There is also a plan to build a user interface for updating and inserting data about future players and games. In addition to data inserting and updating, data also should be collected about players played minutes, cards, and goal assists.

Key words
database, relational database, entity modeling, SQL

SISÄLLYS

1 JOHDANTO

Yleisesti ottaen Suomessa jalkapallotilastointi on tuntunut olevan huonolla tasolla, etenkin vanhempien tietojen osalta. Viime vuosina se on toki kehittynyt ja esimerkiksi Palloliitto julkaisi maaliskuussa 2021 uuden tulospalvelun, mutta vanhempien tietoja ja alasarjojen tilastoja on haastavaa löytää. Tässä työssä keskityn kuitenkin nimenomaan HIFK:n jalkapallon edustusjoukkueen tilastoihin. Tarkoituksena on saada aikaan kokonaisuus, jonka avulla pystyy helposti löytämään vuosikymmenien ajalta HIFK:n tilastoja niin pelaaja- kuin joukkuetasolta. Mukaan mahtuu viisi eri sarjatasoa, yli 700 pelaajaa, lähes 4000 maalia ja yli 90 kautta, joten data on siltä osin hyvin kattavaa, mutta etenkään vanhemmilta vuosilta se ei ole niin laadukasta kuin nykyaikana. Tämän lisäksi eri sarjatasot vaikuttavat datan laatuun – alemmilta sarjatasoilta mm. malintekijöiden tietoja ei löydy niin helposti kuin pääsarjatasolta. Veikkausliigasta saa hyvinkin tarkkaa dataa, mutta neljänneksi korkeimmalla sarjatasolla se ei ole kovin kattavaa.

Tässä opinnäytetyössä suunnittelen ja toteutan relaatiotietokannan HIFK:n jalkapallotilastoille, joka tulee käyttöön myöhemmin julkaistavalle web-sivulle. Työhön kuuluu myös datan käsittely ja vieminen tietokantaan lopullisiin tauluihin, joista dataa on helppo hakea ulkoiseen käyttöön web-sivulle. Työssä ei tulla käsittelemään niinkään itse web-sivun kehittämistä, suunnittelua tai toteuttamista, vaan itse tietokantaa ja siihen liittyvää työtä. Alkuun kerron yleisesti tietokannoista, niiden suunnittelusta ja SQL:stä. Tämän jälkeen käyn läpi työssä toteutettavan tietokannan suunnittelun, kehittämisen, SQL-koodit ja lopputuloksen.

2 TIETOKANNAT

Tietokanta on kokoelma mitä tahansa tietokoneelle tallennettuja tietoja eli dataa, jota voidaan käyttää eri tavoilla. Tietokannat voidaan jakaa kahteen ryhmään, relaatio- ja muihin tietokantoihin. Muita, kuin relaatiotietokantoja kutsutaan NoSQL (not only SQL) -tietokannoiksi, joita ovat esimerkiksi avain-arvo-tietokannat (key-value database), graafitietokannat (graph database) ja dokumenttiorientoituneet tietokannat (document database). (MongoDB 2021.)

2.1 Data

Data on kokoelma tietoja, kuten numeroita, kirjaimia, sanoja, mittayksiköitä, havaintoja, lämpötiloja, ääniä, videoita tai jopa kuvauksia asioista tai esineistä, jotka antavat meille tietoa yksilöistä, esineistä tai havainnoista (Sedkaoui 2018, 34). Data on käytännössä mitä tahansa tietoa, joka on muotoiltu millä tahansa tavalla. Erilaisissa laitteissa olevat ohjelmistot ovat käytännössä kokoelma ohjeita tai käskyjä, joilla tietoa eli dataa käytetään. (Simplilearn 2020.)

Käytännössä kaikki data voidaan kategorisoida kahteen ryhmään: koneen luettavissa oleva (machine-readable) ja ihmisen luettavissa oleva (human-readable) data. Ihminen voi hyvin lukea dataa, jos se on esimerkiksi luonnollista tekstiä. Toisaalta data voi olla myös hyvin kryptistä, strukturoitua tietokonekieltä, jota ihminen ei ymmärrä. Luonnollisesti data voi olla sekä ihmiselle että tietokoneelle ymmärrettävää, kuten esimerkiksi CSV-, HTML- tai JSON-muotoista. Tosin esimerkiksi CSV-muodossa oleva data voi sisältää miljoonia rivejä, jolloin ihmisen on mahdotonta ymmärtää tai lukea sitä ilman tietokonetta. (Norton 2020.)

2.2 Relaatiotietokanta

Relaatiotietokannat (relational database), jota tässäkin työssä käsitellään, ovat yleisesti käytettyjä tietokantoja. Relaatiotietokannat perustuvat relaatiomalliin (relational model). Relaatiotietokanta sisältää tietokantatauluja, joiden tietoja voi käsitellä SQL-

kielellä. Taulujen tiedot liittyvät usein toisiinsa, mikä aiheuttaa niiden välille suhteita (relationship), joista myös relaatiotietokanta on saanut nimensä. (MongoDB 2021.)

3 TIETOKANNAN SUUNNITTELU

Tietokannan suunnittelu on tärkeä osa tietokannan rakentamista. Hyvällä suunnittelulla varmistetaan, ettei tietokannan rakenteita tarvitse jälkeenpäin korjata tai muuttaa, koska jälkeenpäin korjaaminen voi olla haastavaa ja aikaa vievää.

Tietokannan suunnittelu sisältää vaiheita, jotka auttavat luomaan, toteuttamaan ja ylläpitämään lopullista tietokantaa ja tiedonhallintajärjestelmää. Suunnittelun ensisijainen tarkoitus on luoda fyysisiä ja loogisia malleja lopulliselle tietojärjestelmälle. (Naeem 2019.)

Kaksi hyvin yleistä suunnittelussa käytettyä vaihetta ovat käsitemallinnus ja normalisointi, joihin myös tämän työn suunnittelu perustuu.

3.1 Käsitemallinnus

Käsitemallinnus (entity modeling, entity relationship, ER) on menetelmä, jonka tarkoituksena on saada aikaiseksi käsitteitä ja niiden keskinäisistä riippuvuuksista kertova graafinen käsitemalli, joka voidaan kuvata ER-kaaviona. Mallin tarkoitus olisi toimia helposti ymmärrettävänä ja havainnollistavana tietona, jolloin sitä voi käyttää helposti kommunikoidessa esimerkiksi IT:n ja liiketoiminnan välillä. (Hovi 2018.)

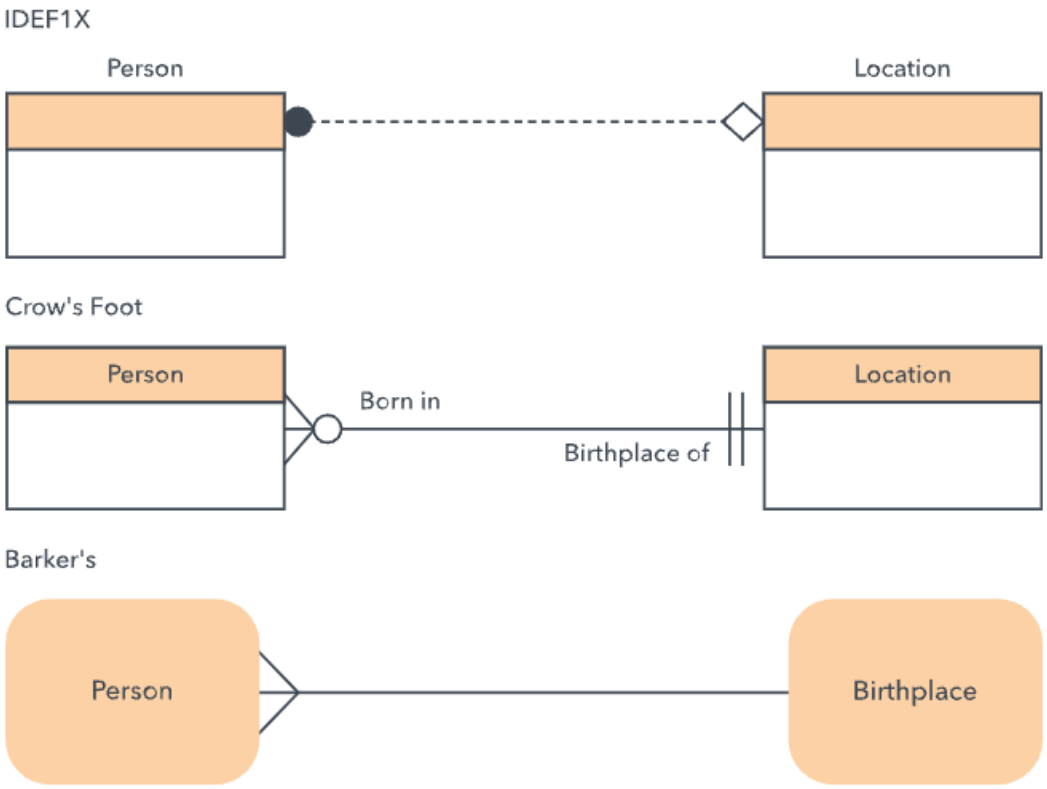
ER-kaavio muodostuu yksilötyypeistä (entity type), yhteyksistä (relationship) ja ominaisuuksista (attribute). Yksilötyyppi kuvaa määriteltävissä olevaa asiaa, kuten henkilöä, esinettä, käsitettä tai tapahtumaa. Esimerkiksi asiakas, opiskelija, auto tai tuote voi olla yksilötyyppi. Yksilötyypit kuvataan yleensä suorakulmioina. Yhteydet kuvaavat sitä, miten yksilötyypit liittyvät toisiinsa. Esimerkiksi kokonaisuudessa, jossa opiskelija voi liittyä kurssille, opiskelija ja kurssi ovat yksilötyyppejä ja niiden välinen yhteys on ilmoittautuminen. Yhteydet kuvataan yleensä salmiakkikuviona tai pelkällä yhdis-

tävällä viivalla. Ominaisuus on kokonaisuuden eli yksilötyypin jokin ominaisuus. Esimerkiksi yksilötyypillä auto voi olla ominaisuuksina mm. väri, merkki ja vuosimalli. Ominaisuudet kuvataan yleensä soikeina tai ympyrän muotoisina kuvioina. (Lucidchart 2021.)

Kuvio 1. Esimerkki yksinkertaisesta ER-kaaviosta. (Lucidchart 2021.)

Kuviossa 1 esitetään yksilötyyppien Person ja Location välillä ns. yhden-suhde-moneen -yhteys (one-to-many relationship) Birthplace. Kaavion merkitys on seuraava: yhdellä henkilöllä (Person) on tasan yksi syntymäpaikka (Location) ja yhdessä syntymäpaikassa on voinut syntyä useita henkilöitä.

ER-mallinnuksessa on olemassa useita merkintäjärjestelmiä eli notaatioita, jotka ovat joiltakin osin samankaltaisia, mutta eroavat joidenkin erityispiirteiden osalta. Yksi yleinen notaatio on kuviossa 1 näkyvä Chenin notaatio, jota tässäkin työssä käytetään luvussa 5 kuvatussa ER-kaaviossa. Muita notaatioita ovat esim. Crown harakanvarvas -notaatio, Barkerin notaatio ja IDEF1X, joista on esitetty esimerkit kuviossa 2.



Kuvio 2. Erilaisia notaatioita (Lucidchart 2021.)

3.2 Normalisointi

Tietokannan normalisointi on loogisen suunnittelun vaihe, jossa käytännössä varmistetaan, että rakenne on tiettyjen normaalimuotosääntöjen mukainen. Tällöin tietokanta ei sisällä tarpeetonta tai toistuvaa dataa, jolloin tiedon lisäämisessä, päivittämisessä tai poistamisessa voisi tulla tietokannan rakenteesta johtuvia ongelmia. (Peterson 2021.)

Normalisointi perustuu normalisointimuotoihin (normal forms), joita ovat 1NF (first normal form), 2NF, 3NF, BCNF (Boyce-Codd normal form), 4NF, 5NF ja 6NF. Yleensä normalisoinnissa käytetään kolmea ensimmäistä normaalimuotoa. (Peterson 2021.)

3.2.1 Ensimmäinen normaalimuoto eli 1NF

Ensimmäisessä normaalimuodossa tarkistetaan yksinkertaisuudessaan, ettei yksittäinen attribuutti sisällä useaa arvoa. Ongelman voi korjata jakamalla tietoja useampaan relaatiomuuttujaan tai useampaan attribuuttiin. (Prakteek 2020.)

3.2.2 Toinen normaalimuoto eli 2NF

Toisessa normaalimuodossa tietokantataulun tulee olla ensimmäisessä normaalimuodossa. Lisäksi taulu ei saa sisältää osittaisia riippuvaisuuksia sellaisissa tauluissa, joissa perusavain muodostuu useammasta kuin yhdestä sarakkeesta. Tämä tarkoittaa sitä, että perusavaimeen kuulumaton sarake ei saa olla vain osittain riippuvainen perusavaimesta. Kuvassa 1 on esimerkki tällaisesta tilanteesta. (Prakteek 2020.)

Employee Id	Department Id	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

Kuva 1. Taulu, jossa toinen normaalimuoto ei toteudu. (Prakteek 2020.)

Kuvan 1 taulun perusavain muodostuu sarakkeista Employee Id ja Department Id. Toimiston sijainti riippuu vain Department Id:stä, joten Office Location on vain osittain riippuvainen perusavaimesta. Kuvassa 2 samat tiedot on jaettu kahteen tauluun, jolloin ne ovat toisessa normaalimuodossa.

Employee Id	Department Id
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

Department Id	Office Location
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

Kuva 2. Kuvan 1 tiedot toisessa normaalimuodossa. (Prakteek 2020.)

3.2.3 Kolmas normaalimuoto eli 3NF

Myös kolmannessa normaalimuodossa edelliset, eli ensimmäinen ja toinen normaalimuoto täytyy toteutua. Lisäksi taulun ei-avainattribuutit eivät saa olla riippuvaisia muista perusavaimeen kuulumattomista attribuuteista. Sarakkeet saavat siis olla riippuvaisia vain ja ainoastaan perusavaimesta. (Prakteek 2020.)

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

Kuva 3. Esimerkkitaulu, jossa kolmas normaalimuoto ei toteudu. (Prakteek 2020.)

Kuvan 3 esittämän taulun perusavain on Student Id. Subject on riippuvainen Subject Id:stä. Jotta tiedot olisivat kolmannessa normaalimuodossa, Subject tulisi viedä omaan tauluunsa ID:n kanssa, mutta Subject Id jäisi kuvan tauluun.

4 KÄYTETYT TEKNIIKAT

Tässä luvussa esitellään työssä käyttämiäni työkaluja. Näitä ovat SQL-kieli, SQL Server ja Python-kieli. Lisäksi käytin apuna mm. Draw.io -ohjelmaa ER-kaavion tekemiseen sekä Notepad++ -ohjelmaa tekstinkäsittelyssä.

4.1 SQL

SQL (structured query language) on kieli, jolla voi tallentaa, käsitellä sekä hakea tietoa relaatiotietokannoista (Tutorialspoint 2021).

Tietokannat ovat luonnollisesti hyvin tärkeä osa datan säilyttämistä, tutkimista, hakemista, muuttamista ja analysointia ajatellen, joten SQL-kieli on hyvin yleinen työkalu data-analytiikan ja datan käsittelyn parissa.

SQL:n yleisimmät ja yksinkertaisimmat lauseet ovat datan hakemista tietokantataulusta SELECT-lauseella, datan lisäämistä INSERT-lauseella, datan muuttamista UPDATE-lauseella sekä datan poistamista DELETE-lauseella. Kuvassa 4 on esitetty nämä SQL-lauseet käytännössä.

```
SELECT ID, FIRSTNAME, LASTNAME FROM PLAYER;

INSERT INTO PLAYER (ID, FIRSTNAME, LASTNAME, NUMBER, BIRTHDATE)
VALUES (5, 'Harry', 'Kane', 9, '28.7.1993');

UPDATE PLAYER SET NUMBER = 10 WHERE ID = 5;

DELETE FROM PLAYER WHERE ID = 5;
```

Kuva 4. Esimerkki yksinkertaisista SQL-lauseista.

LeagueId	PlayerId	GameId	Goals	Assists	YellowCards	RedCards	PlayedMinutes
1	1009	6035	0	0	0	0	90
1	1008	6035	0	0	1	0	90
1	15025	6035	0	0	0	0	90
1	1020	6035	1	0	0	0	56
1	1012	6035	0	0	0	0	65
1	15028	6035	0	0	0	1	72

Kuva 5. Esimerkki tietokantataulun sisältämästä datasta.

Kuten kuvasta 5 voi nähdä, tietokantataulu muodostuu sarakkeista ja riveistä eli tietueista. Jokaisessa tietokantataulussa tulee olla perusavain (primary key), joka muodostuu yhdestä tai useammasta sarakkeesta. Perusavain yksilöi jokaisen rivin, eli useammalla rivillä ei voi olla samaa perusavainta. Esimerkiksi kuvan 5 esimerkin taulussa perusavain voisi muodostua sarakkeista LeagueId, PlayerId ja GameId. Perusavain on pakollinen tieto, eli se täytyy määritellä pakolliseksi (NOT NULL), jolloin se ei voi sisältää arvoja, joita ei tiedetä. Arvo, jota ei tiedetä, merkitään ”null”. Muut kuin perusavaimeen kuuluvat sarakkeet voidaan määrittää myös niin, että arvot eivät ole pakollisia (NULL). (Tutorialspoint 2021.)

Taulun jokaiselle sarakkeelle tulee määritellä tietotyyppi (data type). Tietotyyppi määrittää, millaista tietoa sarake voi sisältää. Tyyppejä voi luoda itse, mutta tietokannanhallintajärjestelmät tarjoavat myös valmiita tyyppejä. Eri tietokannanhallintajärjestelmien tyypit eroavat hieman toisistaan. SQL Server tarjoaa valmiina tyyppeinä esim.

numeraaliset tyypit int, smallint, bigint ja decimal, merkkijonotyypit char(N), nvarchar(N) ja päivämäärätyypit date tai datetime. (Petrovic 2019.)

4.2 SQL Server

Työssä käytettiin tietokannan hallintajärjestelmänä (database management system) SQL Serveriä. Se on Microsoftin kehittämä tietokannan hallintajärjestelmä, jossa on käytössä oma standardoidun SQL-kielen toteutus T-SQL (Transact-SQL). SQL Serverin pääkäyttöliittymätyökaluna toimii SQL Server Management Studio. Ensimmäinen versio SQL Serveristä julkaistiin Microsoftin ja Sybasen toimesta vuonna 1989. (Peterson 2021.)

SQL Server on alun perin suunniteltu tietojen hallintaan ja tallentamiseen. Tätä varten SQL Serverin sisällä on erilaisia työkaluja, kuten SQL Server Data Tools tietokantojen kehittämiseen sekä SQL Server Management Studio tietokantojen käyttöönottoon ja hallintaan. Näiden lisäksi SQL Server tukee esimerkiksi erilaisia business intelligence- ja analytiikkatoimintoja. Tällaisia työkaluja ovat esimerkiksi datan käsittelyä, visualisointia ja BI-raportointia varten olevat SQL Server Analysis Services ja SQL Server Reporting Services. (Hughes 2019.)

4.3 Python

Python on monikäyttöinen ja korkeatasoinen ohjelmointikieli, jonka suunnitteli alun perin hollantilainen, myöhemmin mm. Googlella työskennellyt Guido van Rossum vuonna 1991. (Pramanick 2019.)

Eastwoodin (2020) listaus kymmenestä suosituimmasta ohjelmointikielestä listasi Pythonin vuoden 2020 suosituimmaksi. Tämän mukaan syitä Pythonin suosioon ovat mm. sen helppo syntaksi, jonka vuoksi sitä on helppo oppia, laajat kirjastovalikoimat ja työkalut, sekä yhteensopivuus muiden ohjelmointikielien kanssa.

Työssä käytetty Python-kirjasto Pandas on vuonna 2008 kehitetty ja vuodesta 2009 lähtien käytössä ollut avoimen lähdekoodin kirjasto, jolla voi käsitellä taulukkomuotoista dataa. Se perustuu kahteen päätietorakenteeseen, sarjoihin (series), jotka ovat yksiulotteisia listoja ja datakehikkoihin data frames), jotka ovat kaksiulotteisia, matriisimaisia taulukoita. (Custer 2020; Pandas 2020.)

Datakehikon (data frame) tekeminen onnistuu manuaalisesti, mutta sen voi lukea myös esimerkiksi suoraan csv-tiedostosta kuvan 6 mukaisesti. Kuvassa 7 samaa datakehikkoa on käsitelty erilaisilla funktioilla.

```
import numpy as np
import pandas as pd

Data = pd.read_csv('VeikkausliigaGoals2020.csv', encoding='latin-1',
                   usecols = ['Id', 'Winner', 'Penalty', 'FirstName',
                              'LastName', 'Minute', 'AssistFirstName', 'AssistLastName'])
```

Kuva 6. Datakehikon, eli data framen tekeminen read_csv -funktiolla. Funktio lukee tiedot csv-tiedostosta. Usecols-parametrilla valitaan luettavat sarakkeet.

```
Data1 = Data.where(Data['Minute'] < 16)
first = Data1.dropna()
Data2 = Data.where((Data['Minute'] < 31) & (Data['Minute'] > 15))
second = Data2.dropna()
```

Kuva 7. Datakehikon datan käsittelyä where-funktiolla. Dropna-funktio poistaa puuttuvat arvot.

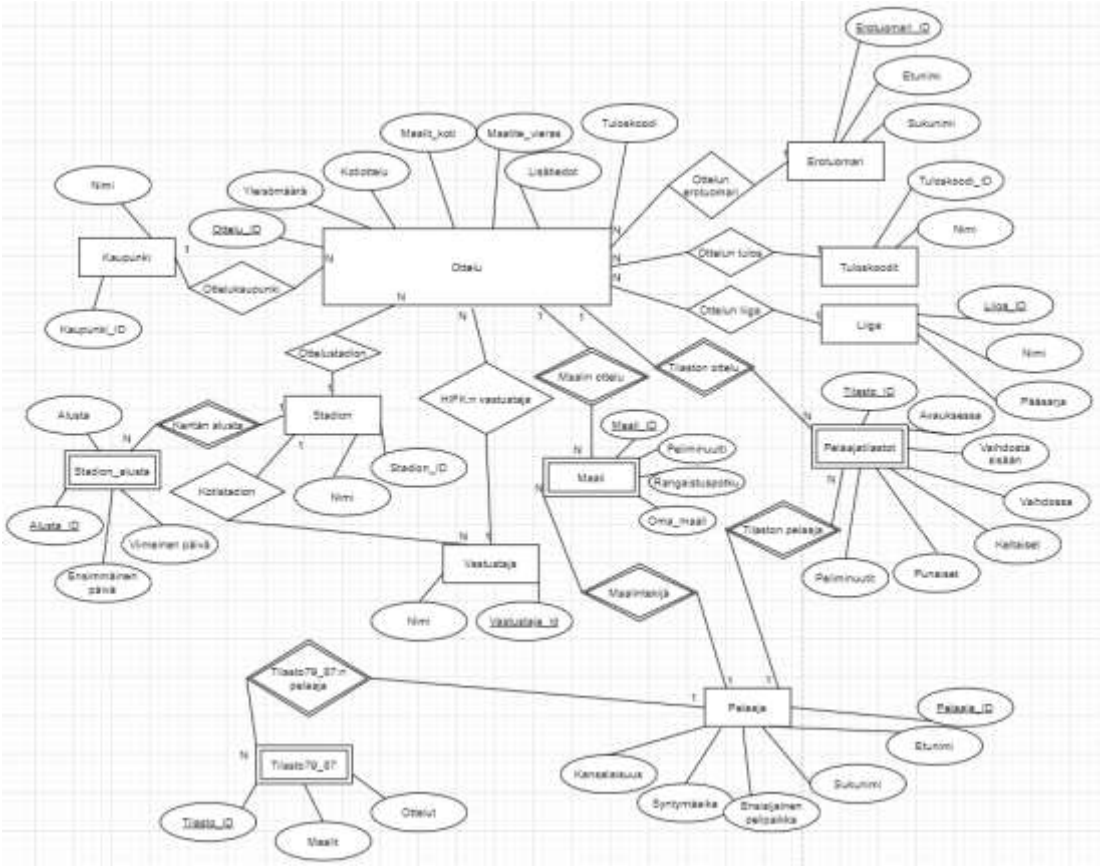
Datakehikkojen sisältöä voi tarkastella mm. head- ja tail-funktioilla, joilla saa näkyviin kehikon ensimmäiset ja viimeiset rivit. Oletuksena on viiden ensimmäisen tai viimeisen rivin näyttäminen, mutta funktiolle voi antaa rivimäärän myös argumenttina. Kuvassa 8 näkyy ensin esimerkit tietojen tulostamisesta. Ominaisuudella dtypes saa näkyviin datakehikon sarakkeiden tyypit.

```
print(Data.dtypes)
print(Data.head())
print(Data.tail(3))
```

Kuva 8. Datakehikon sarakkeiden tyyppien ja ensimmäisten viiden sekä viimeisten kolmen rivin arvojen tulostaminen.

5 ER-KAAVIO

ER-kaavion tekemiseen käytettiin Draw-io -nimistä ohjelmaa, joka tunnetaan myös nimellä Diagrams.net. Tämän ohjelman valinta oli helppoa sen helppokäyttöisyyden ja yksinkertaisuuden takia. Lisäksi ohjelman työpöytäversio oli ilmainen.



Kuvio 3. Tietokannan ER-kaavio

Kuvio 3 sisältää 12 suorakulmiolla merkittyä yksilötyyppiä, joista neljä on heikkoja (weak entity). Heikot yksilötyypit, jotka on merkitty kahdella reunaviivalla, ovat riippuvaisia vähintään yhdestä vahvasta yksilötyypistä, kuten esimerkiksi pelaajatilastot ovat riippuvaisia pelaajista ja otteluista. Yksilötyyppien väliset yhteydet on merkitty salmiakkikuvioilla, jotka yhdistyvät kahteen eri yksilötyyppiin. Kahdella reunaviivalla merkityissä heikoissa yhteyksissä myös vähintään toinen yksilötyyppi on heikko. Ominaisuudet on merkitty soikeina tai ympyröinä. Alleviivattu ominaisuus on kyseisen yksilötyypin ominaisuus. Näistä tiedoista muodostetaan tietokantakaava luvussa kuusi.

Tietokannan ER-kaaviossa vahvat yksilötyypit ovat Ottelu, Pelaaja, Kaupunki, Stadion, Vastustaja, Liiga, Tuloskoodit ja Erotuomari, sekä heikot Maali, Pelaajatilastot, Tilasto79_87 ja Stadion_alusta. Näistä yksilötyypeistä eniten yhteyksiä muihin yksilötyyppeihin on Ottelu-yksilötyypillä, joka yhdistyy kahdeksaan muuhun yksilötyyppiin. Ottelu yhdistyy myös kaikkiin vahvoihin yksilötyyppeihin, paitsi Pelaajaan. Vahvoja yksilötyyppejä, joihin ottelu yhdistyy, ovat Kaupunki, Stadion, Vastustaja, Liiga, Tuloskoodit ja Erotuomari. Näitä kaikkia ottelulla voi olla vain yksi. Lisäksi ottelu liittyy heikkoihin yksilötyyppeihin Maaliin ja Pelaajatilastoon, joita ottelulla voi olla useita.

Pelaaja-yksilötyyppi ei liity yhteenkään vahvaan yksilötyyppiin, sillä jokainen pelaajaan liittyvä yksilötyyppi vaatii pelaajan olemassaolon, koska pelaajatilastoja ei voi olla ilman pelaajaa. Lisäksi pelaaja liittyy maaliin, joka on riippuvainen sekä pelaajasta, että ottelusta. Pelaajalla voi olla useita maaleja ja tilastomerkintöjä.

Ottelun ja Pelaajan yhteyksien lisäksi kaaviossa on kaksi yhteyttä. Ne koskevat vahvoja yksilötyyppejä Stadion ja Vastustaja, sekä heikkoa yksilötyyppiä Stadion_alusta. Stadionin ja Vastustajan yhteys kuvaa vastustajan kotistadionia ja Stadionin ja Stadion_alustan yhteys kuvaa luonnollisesti stadionin alustaa.

6 TIETOKANTAKAAVA

ER-kaavion tiedosta muodostui tietokantakaava, jossa kuvataan relaatiomuuttujina tietokannat tiedot. Relaatiomuuttujista muodostetaan tietokantataulut. Koko tietokantakaava löytyy liitteestä 1.

```
VAR OTTELU BASE RELATION_TYPE {
 OTTELU_ID INT NOT NULL,
 AJANKOHTA [DATE] NOT NULL,
 YLEISOMAARA [INT] NULL,
 KOTIOTTELU [BIT] NOT NULL,
 MAALIT_KOTI [INT] NOT NULL,
 MAALIT_VIERAS [INT] NOT NULL,
 LISATIEDOT [VARCHAR](150) NULL,
 VASTUSTAJA_ID INT NOT NULL,
 LIIGA_ID INT NOT NULL,
 KAUPUNKI_ID INT NULL,
 STADION_ID INT NULL,
 EROTUOMARI_ID INT NULL,
 TULOSKOODI INT NOT NULL }
PRIMARY KEY {OTTELU_ID}
FOREIGN KEY {VASTUSTAJA_ID} REFERENCES VASTUSTAJA
FOREIGN KEY {LIIGA_ID} REFERENCES LIIGA
FOREIGN KEY {KAUPUNKI_ID} REFERENCES KAUPUNKI
FOREIGN KEY {STADION_ID} REFERENCES STADION
FOREIGN KEY {EROTUOMARI_ID} REFERENCES EROTUOMARI
FOREIGN KEY {TULOSKOODI} REFERENCES TULOSKOODIT;
```

Kuva 9. Relaatiomuuttuja Ottelu.

Jokaisesta ER-kaavion yksilötyypistä on muodostettu relaatiomuuttuja, josta tulee tietokantataulu. Yksilötyyppien ominaisuuksista tulee tietokantataulujen sarakkeita. Alleviivatusta ominaisuudesta on muodostettu perusavain (primary key). Kuvassa 9 on esimerkkinä relaatiomuuttuja Ottelu, jonka perusavain on OTTELU_ID.

Jokaisesta ER-kaavion yhden-suhde-moneen yhteydestä on tehty viiteavain (foreign key) relaatiomuuttujaan, johon voi liittyä yhteydessä vain yksi yksilö. Esimerkiksi Kuvassa 9 viiteavain VASTUSTAJA_ID syntyy yhteydestä, jossa ottelulla voi olla vain yksi vastustaja, mutta vastustajalla voi olla monta ottelua. Toisin sanoen lopullisen tietokantataulun dbo.OTTELU sarakkeen Vastustaja_ID täytyy olla jokin arvo VASTUSTAJA-taulun perusavaimesta.

```
VAR PELAAJATILASTOT BASE RELATION_TYPE (
 OTTELU_ID INT NOT NULL,
 PELAAJA_ID INT NOT NULL,
 TILASTO_ID INT NOT NULL,
 AVAUKSESSA BIT NULL,
 VAIHDOSTA_SISAAN BIT NULL,
 VAIHDOSSA BIT NULL,
 KELTAISET INT NULL,
 PUNAISET INT NULL,
 PELIMINUUTIT INT NULL )
PRIMARY KEY {OTTELU_ID, PELAAJA_ID, TILASTO_ID}
FOREIGN KEY {OTTELU_ID} REFERENCES OTTELU
ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY {PELAAJA_ID} REFERENCES PELAAJA
ON UPDATE CASCADE ON DELETE CASCADE;
```

Kuva 10. Tietokantakaavan relaatiomuuttuja PELAAJATILASTOT.

Kuvan 10 relaatiomuuttuja PELAAJATILASTOT on heikosta yksilötyypistä muodostettu relaatiomuuttuja. Koska kyseessä on heikosta yksilötyypistä muodostettu relaatiomuuttuja, sen perusavain muodostuu omasta avainominaisuudesta TILASTO_ID, sekä sen yhteyksien toisen pään perusavaimista OTTELU_ID ja PELAAJA_ID. Lisäksi relaatiomuuttujan viiteavaimeen on määritelty vyörytyssäännöt (CASCADE), eli kun viitattavan taulun tietoja päivitetään tai poistetaan, ne päivittyvät myös taulussa, josta ko. taulu on riippuvainen. Esimerkiksi ottelutiedon poistaminen dbo.OTTELU-taulusta aiheuttaa myös sen, että ko. ottelun tiedot poistuvat myös dbo.PELAAJATILASTOT-tauluista.

7 DATAN VIEMINEN TIETOKANTAAN

Datan vieminen tietokantaan esimerkiksi CSV-tiedostosta on yleisesti ottaen hyvin yksinkertaista, etenkin kun kyseessä ei ole niin valtavat datamäärät, että koko itsessään aiheuttaisi ongelmia. Jokaisella tietokannanhallintajärjestelmällä on omat valmiit käytännöt tätä varten ja SQL Server Database Management Studion käytäntö on hyvin käyttäjäystävällinen.

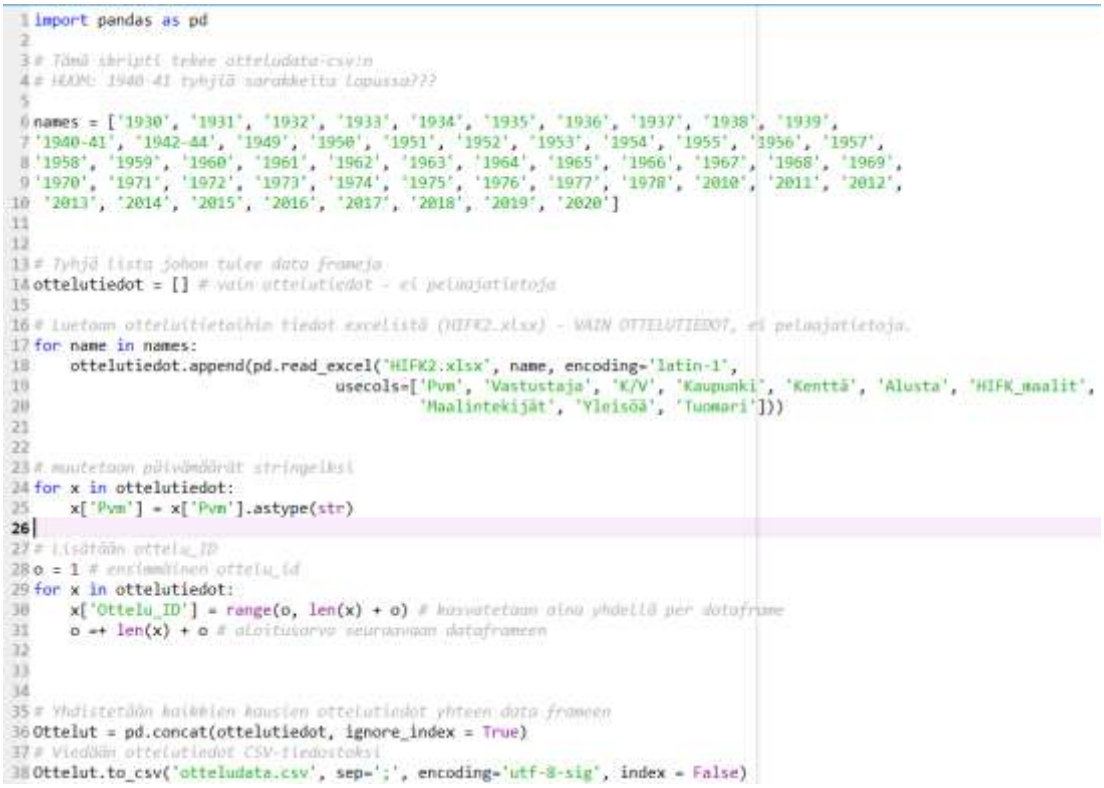
Tässä tapauksessa kuitenkaan datan vieminen tietokantaan ei ole yksinkertaista. Alkuperäinen data ei ole siinä muodossa, että sen voisi viedä sellaisenaan tietokantaan. Alkuperäinen data on yhdessä Excel-tiedostossa, jossa jokaiselle kaudelle on oma taulukko (worksheet tab). Jokainen taulukko sisältää sarakkeet ottelutiedoista, sekä sa-

rakkeen jokaiselle pelaajalle, joka ko. kaudella on pelannut. Tästä johtuen yhtenä ongelmana on esimerkiksi se, että pelaajat, jotka ovat pelanneet useammalla kuin yhdellä kaudella, ovat Excelissä useassa sarakkeessa, mutta kukin sarake sisältää erilaisia tietoja.

Dataa täytyy siis käsitellä ennen sen viemistä tietokantaan. Tässä se tapahtuu Pythonilla kahdella erillisellä skriptillä. Kuvassa 11 oleva skripti tekee pelaajat_kaikki -nimisen CSV-tiedoston, joka sisältää sarakkeen Ottelu_ID, sekä jokaiselle pelaajalle oman sarakkeensa, johon tulee tietoja otteluittain. Kuvassa 12 oleva skripti tekee otteludata-nimisen CSV-tiedoston. Tähän tulee sarakkeet Ottelu_ID, Pvm, Vastustaja, K/V, Kaupunki, Kenttä, Alusta, HIFK_maalit, Vastustaja_maalit, Maalintekijät, Yleisöä ja Tuomari. Ottelu_ID:llä voi siis yhdistellä tietoja näiden kahden CSV-tiedoston välillä, jotka myöhemmässä vaiheessa viedään tietokantaan.

```python
import pandas as pd

# Tämä skripti tekee pelaajat-csv:n

# Excel-tiedoston sheetien nimet.
names = ['1930', '1931', '1932', '1933', '1934', '1935', '1936', '1937', '1938', '1939',
'1940-41', '1942-44', '1949', '1950', '1951', '1952', '1953', '1954', '1955', '1956', '1957',
'1958', '1959', '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968', '1969',
'1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977', '1978', '2010', '2011', '2012',
 '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020']

# Tyhjä lista johon tulee data frameja
pelaajatiedot = [] # kaikki tiedot Excelistä

# Luetaan pelaajatietoihin tiedot excelistä (HIFK2.xlsx)
for name in names:
    pelaajatiedot.append(pd.read_excel('HIFK2.xlsx', name, encoding='latin-1'))

# muutetaan päivämäärät stringeiksi
for x in pelaajatiedot:
    x['Pvm'] = x['Pvm'].astype(str)

# lisätään otteluihin ottelu_id:t
o = 1 # ensimmäinen ottelu_id
for x in pelaajatiedot:
    x['Ottelu_ID'] = range(o, len(x) + o) # kasvatetaan aina yhdellä per dataframe
    o =+ len(x) + o # aloitusarvo seuraavaan dataframeen



# Yhdistetään kaikkien kausien ottelutiedot yhteen data frameen concatilla
Pelaajat = pd.concat(pelaajatiedot, ignore_index = True)

# Poistetaan ottelutiedot (vain Ottelu_ID jää)
Pelaajat.drop(['Pvm', 'Vastustaja', 'K/V', 'Kaupunki', 'Kenttä', 'Alusta', 'HIFK_maalit',
               'Vastustaja_maalit', 'Maalintekijät', 'Yleisöä', 'Tuomari'], axis='columns', inplace=True)



# Viedään pelaajatiedot CSV-tiedostoksi
Pelaajat.to_csv('pelaajat_kaikki.csv', sep=';', encoding='utf-8-sig', index=False)
```

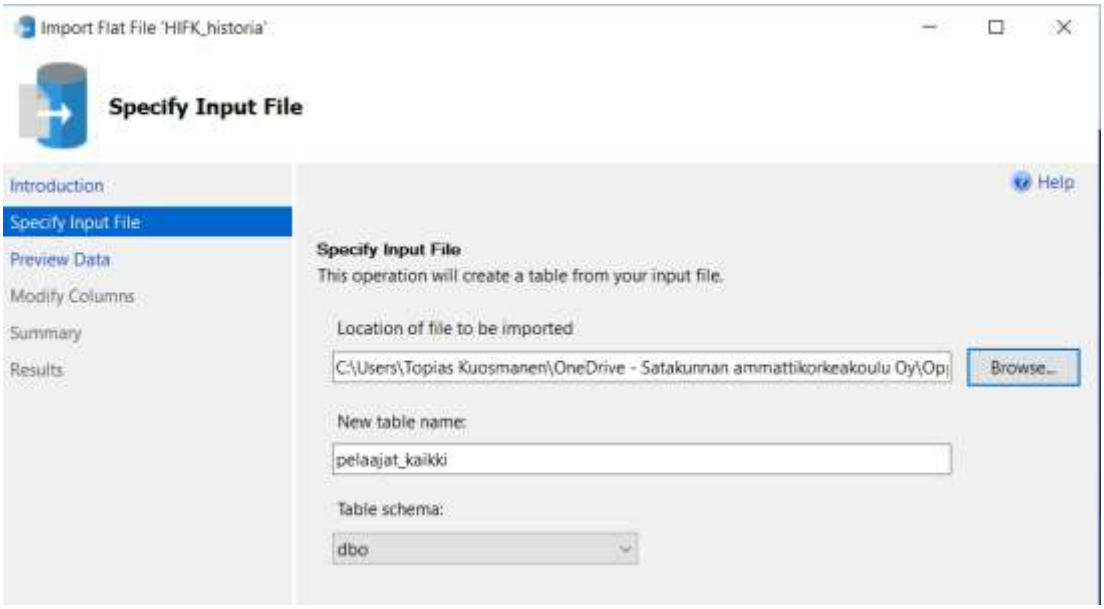
Kuva 11. Python-skripti, jolla luodaan pelaajat_kaikki -niminen CSV-tiedosto.



```
import pandas as pd

# Tämä skripti tekee otteludata-csv:n
# HUOM: 1940-41 tyhjiä sarakkeita lopussa???

names = ['1930', '1931', '1932', '1933', '1934', '1935', '1936', '1937', '1938', '1939',
'1940-41', '1942-44', '1949', '1950', '1951', '1952', '1953', '1954', '1955', '1956', '1957',
'1958', '1959', '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968', '1969',
'1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977', '1978', '2010', '2011', '2012',
'2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020']


# Tyhjä lista johon tulee data frameja
ottelutiedot = [] # vain ottelutiedot - ei pelaajatietoja

# luetaan otteluittietoihin tiedot excelistä (HIFK2.xlsx) - VAIN OTTELUTIEDOT, ei pelaajatietoja.
for name in names:
    ottelutiedot.append(pd.read_excel('HIFK2.xlsx', name, encoding='latin-1',
                                usecols=['Pvm', 'Vastustaja', 'K/V', 'Kaupunki', 'Kenttä', 'Alusta', 'HIFK_maalit',
                                         'Maalintekijät', 'Yleisöä', 'Tuomari']))


# muutetaan päivämäärät stringeiksi
for x in ottelutiedot:
    x['Pvm'] = x['Pvm'].astype(str)

# Lisätään ottelu_ID
o = 1 # ensimmäinen ottelu_id
for x in ottelutiedot:
    x['Ottelu_ID'] = range(o, len(x) + o) # kasvatetaan aina yhdellä per dataframe
    o =+ len(x) + o # aloitusarvo seuraavaan dataframeen


# Yhdistetään kaikkien kausien ottelutiedot yhteen data frameen
Ottelut = pd.concat(ottelutiedot, ignore_index = True)
# Viedään ottelutiedot CSV-tiedostoksi
Ottelut.to_csv('otteludata.csv', sep=';', encoding='utf-8-sig', index = False)
```

Kuva 12. Python-skripti, jolla luodaan otteludata-niminen CSV-tiedosto.

CSV-tiedostojen tekemisen jälkeen data on siinä muodossa, että sen voi viedä sellaisenaan tietokantaan. Microsoftin SQL Server Management Studio, kuten muutkin tietokannanhallintajärjestelmän käyttöliittymät tarjoavat omat tavat, joilla datan tuominen tietokantaan onnistuu. Kyseessä on hyvin yksinkertainen ja käyttäjäystävällinen toimenpide. Microsoftin SQL Server Management Studiossa dataa pääsee tuomaan toiminnolla, jonka nimi on Import flat file. Kuvassa 13 on näkymä, jonka kautta dataa voi tuoda.



Kuva 13. SQL Server Management Studion käyttöliittymä datan tuomista varten.

8 SQL-LAUSEET

Dataa tuotiin tietokantaan kahteen eri tauluun: dbo.otteludata ja dbo.pelaajat_kaikki. Näiden taulujen tietoja yhdistää sarake Ottelu_ID. Muutoin dbo.otteludata sisältää kaiken oleellisen tiedon otteluista ja dbo.pelaajat_kaikki sisältää pelaajien nimet sekä tiedon, onko ko. pelaaja ollut tietyssä ottelussa avauskokoonpanossa, tullut vaihdosta kentälle vai ollut koko pelin vaihtopenkillä. Tauluja ei ole tarkoitus käyttää valmiissa versiossa, vaan nämä ovat tauluja, joissa data on sellaisessa muodossa, missä se on tietokantaan raa'assa muodossa tuotu. Tästä eteenpäin dataa kuitenkin käsitellään SQL-lauseilla ja näistä kahdesta taulusta on tarkoitus kerätä dataa varsinaisiin tietokantatauluihin, jotka luodaan seuraavassa vaiheessa.

8.1 Tietokantataulujen luominen

Tietokantaan luotiin tietokantakaavan mukaisesti 12 taulua. Vain kolmeen tauluun, eli dbo.LIIGA- ja dbo.TULOSKOODIT- ja dbo.TILASTO79_87-tauluihin tietoja ei haettu dbo.pelaajat_kaikki- tai dbo.otteludata-tauluista, eli alkuperäisestä datasta. Muut taulut muodostuivat kyseisistä tiedoista. Kaikki taulujen luonti -lauseet löytyvät liitteestä 2.

```
CREATE TABLE [dbo].[VASTUSTAJA](
    [VASTUSTAJA_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [NIMI] [VARCHAR](50) NOT NULL,
    [KOTISTADION_ID] [INT] NULL,
    CONSTRAINT FK_VASTUSTAJA_KOTISTADION_ID
  FOREIGN KEY (KOTISTADION_ID) REFERENCES STADION(STADION_ID)
    )

CREATE TABLE [dbo].[OTTELU](
    [OTTELU_ID] [INT] PRIMARY KEY NOT NULL, -- Tähän ei lisätä automaattisesti ottelu_ID:tä.
    [AJANKOHTA] [DATE] NOT NULL,
    [VASTUSTAJA_ID] [INT] NOT NULL,
    [LIIGA_ID] [INT] NOT NULL,
    [YLEISOMAARA] [INT] NULL,
    [EROTUOMARI_ID] [INT] NULL,
    [STADION_ID] [INT] NULL,
    [KAUPUNKI_ID] [INT] NULL,
    [KOTIOTTELU] [BIT] NOT NULL,
    [MAALIT_KOTI] [INT] NOT NULL,
    [MAALIT_VIERAS] [INT] NOT NULL,
    [LISATIEDOT] [VARCHAR](150) NULL,
    [TULOSKOODI] [INT] NOT NULL,
    CONSTRAINT FK_OTTELU_VASTUSTAJA_ID
  FOREIGN KEY (VASTUSTAJA_ID) REFERENCES VASTUSTAJA(VASTUSTAJA_ID),
  CONSTRAINT FK_OTTELU_EROTUOMARI_ID
  FOREIGN KEY (EROTUOMARI_ID) REFERENCES EROTUOMARI(EROTUOMARI_ID),
  CONSTRAINT FK_OTTELU_STADION_ID
  FOREIGN KEY (STADION_ID) REFERENCES STADION(STADION_ID),
  CONSTRAINT FK_OTTELU_KAUPUNKI_ID
  FOREIGN KEY (KAUPUNKI_ID) REFERENCES KAUPUNKI(KAUPUNKI_ID),
  CONSTRAINT FK_OTTELU_LIIGA_ID
  FOREIGN KEY (LIIGA_ID) REFERENCES LIIGA(LIIGA_ID),
  CONSTRAINT FK_OTTELU_TULOSKOODI_ID
  FOREIGN KEY (TULOSKOODI) REFERENCES TULOSKOODIT(TULOSKOODI_ID)
    )
```

Kuva 14. Taulujen dbo.VASTUSTAJA ja dbo.OTTELU luominen.

Lähes jokaisen taulun perusavain (primary key) on määritelty automaattisesti yhdellä kasvavaksi numeraaliseksi sarakkeeksi. Tietokanta siis luo tällaiselle taululle itsestään perusavaimen, eikä sitä tarvitse lisätä. Parametreissä määritellään, mistä numerosta perusavain alkaa ja kuinka monella se kasvaa. Poikkeuksena kuvassa 14 näkyvä dbo.OTTELU, jossa ottelun ID on muodostettu jo aikaisemmassa vaiheessa, joten se tuodaan tauluun sellaisenaan.

8.2 Tietojen tarkistaminen ja korjaaminen

Alkuperäinen data sisälsi luonnollisesti virheitä, joista ensimmäiset räikeimmät virheet korjataan SQL-lauseilla. Näitä tietoja oli mm. virheellisesti kirjoitetut joukkueen nimet, erotuomarin nimet, stadionit tai kaupungit. Tämä tarkistustyö piti tehdä hyvin pitkälti manuaalisena, mutta yksittäisiä tietoja ei ollut kovin paljoa, joten ne oli suhteellisen nopeasti tarkistettu. Tiedot, joista ei ollut varmuutta, tarkistettiin datan toimittajalta. Ideana oli siis korjata tiedot dbo.otteludata- ja dbo.pelaajat_kaikki -tauluihin, jotta niistä voi viedä valmista ja laadukasta tietoa suoraan lopullisiin tauluihin.

Alkuperäisen datan haastavin ja eniten käsittelyä vaativa sarake on Maalintekijät, joka sijaitsee taulussa otteludata. Sarake sisältää tiedot maalintekijöistä, maalintekoajoista sekä siitä, onko kyseessä rangaistuspotku tai oma maali. Osassa otteluita näistä tiedoista on vain osa tai ei tietoja ollenkaan, kuten kuvassa 15 näkyy. Lisäksi sarakkeessa on erikseen maininta, jos sama pelaaja on tehnyt useamman maalin ottelussa, vaikka sen voisi merkata pelkillä peliminuuteilla. Näin ollen yksi sarake sisältää useita erilaisia tietoja eri tavalla merkittyinä, jotka pitää saada omiin sarakkeisiinsa. Kuvassa 15 näkyy myös dokumentoituna korjausta vaativia asioita, joita sarakkeeseen liittyy, kuten merkintä puuttuvista peliminuuteista ”[??]”.

Maalintekijät
Sundberg [6], Huttunen [74/rp], Haila [76]
NULL
Heltola 2, Ekman
NULL
Ekman [80]
Ekman
Sundberg
Ilkka Haila [??], Heltola [69]
NULL
NULL
Lehtinen 2 [17, 30]
NULL
Sundberg 2 [18, 31]
NULL
Sundberg 2 [29, 50], Järn [56], Huttunen [80]

Kuva 15. Otteludata-taulun Maalintekijät-sarakkeen ei-rakenteellista dataa, jota on merkitty usealla eri tavalla.

8.3 Tietojen siirtäminen tietokantatauluihin

Kun tietokantataulut on luotu ja alkuperäinen data on tarkistettu ja korjattu siltä osin kuin se on mahdollista, tiedot voidaan siirtää lopullisiin tietokantatauluihin. Alla käydään läpi esimerkkejä erilaisten tietojen syöttämisestä tauluihin, mutta esimerkit kaikkien taulujen tietojen syöttämisestä löytyvät liitteestä 3.

```sql
-- Lisätään vain sarjojen nimet ja tieto pääsarjasta. Tämä on yksinkertainen taulu

    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Veikkausliiga', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Ykkönen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kakkonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kolmonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Nelonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Vitonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kutonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Seiska', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Mestaruussarja', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('SM-kilpailu', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Suomen Cup', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Suomensarja', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('SM-sarja', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('I-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('II-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('III-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('IV-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Palloliiton Mestaruussarja', 1);
```

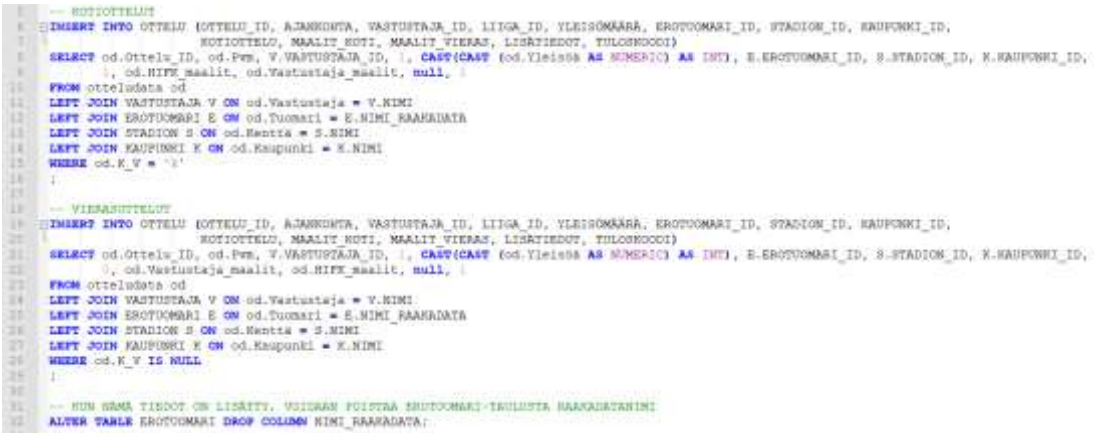
Kuva 16. Tietojen syöttäminen dbo.LIIGA-tauluun.

Suomalaisen jalkapallon pääsarjalla on ollut historiassa useampi nimi, nykyinen Veikkausliiga on ollut käytössä vuodesta 1990 alkaen. Taulussa on sarake ”PÄÄSARJA”, jotta myöhemmin voi tarkastella erikseen pääsarjatietoja. Kuvassa 16 on esimerkki tietojen syöttämisestä tauluun, jonka tiedot eivät tule tauluista dbo.otteludata tai dbo.pelaajat_kaikki.

```sql
-- ETUNIMI
WITH ETUNIMI (ETU_NIMI, COLUMN_NAME)
AS
(
    SELECT RIGHT(COLUMN_NAME,LEN(COLUMN_NAME)-CHARINDEX('_',COLUMN_NAME)) AS ETU_NIMI, COLUMN_NAME
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = 'pelaajat_kaikki'
    AND COLUMN_NAME != 'Ottelu_ID'
    AND COLUMN_NAME NOT LIKE '%(ny)'
    AND CHARINDEX('_',COLUMN_NAME) <> 0
    AND COLUMN_NAME NOT LIKE '%\_%' ESCAPE '\'' -- Kaksi heittomerkkiä, koodia ajaessa vain yksi
)
INSERT INTO PELAAJA_ETUNIMI (ETUNIMI, KOKONIMI)
SELECT LEFT(ETU_NIMI, ISNULL(NULLIF(CHARINDEX('_', ETU_NIMI) - 1, -1), LEN(ETU_NIMI))) AS ETU_NIMI, COLUMN_NAME AS KOKONIMI
FROM ETUNIMI
;

-- Seuraavaksi pelkät sukunimet sisältävät pelaajat (näitä ei ole montaa)

INSERT INTO PELAAJA_SUKUNIMI (KOKONIMI, SUKUNIMI)
SELECT COLUMN_NAME AS KOKONIMI,  LEFT(COLUMN_NAME, ISNULL(NULLIF(CHARINDEX('_', COLUMN_NAME) - 1, -1), LEN(COLUMN_NAME))) AS SUKUNIMI
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'pelaajat_kaikki'
AND COLUMN_NAME != 'Ottelu_ID'
AND COLUMN_NAME NOT LIKE '%(ny)'
AND COLUMN_NAME NOT LIKE '%\_%' ESCAPE '\'' -- 4 -- Kaksi heittomerkkiä, koodia ajaessa vain yksi
;

-- Lopuksi lisätään nämä oikeaan PELAAJA-tauluun ja poistetaan välitaulut.
-- Käytetään LEFT JOIN, koska jokaisella pelaajalla ei ole etunimeä (Esim. "Vitinho").
INSERT INTO PELAAJA (SUKUNIMI, ETUNIMI, KOKONIMI)
SELECT A.SUKUNIMI, B.ETUNIMI, A.KOKONIMI
FROM PELAAJA_SUKUNIMI A LEFT JOIN PELAAJA_ETUNIMI B ON A.KOKONIMI = B.KOKONIMI -- 614
;

-- Poistetaan välitaulut:
DROP TABLE PELAAJA_SUKUNIMI;
DROP TABLE PELAAJA_ETUNIMI;
```

Kuva 17. Tietojen syöttäminen dbo.PELAAJA-tauluun muiden kuin maalivahtien osalta.

Kuvassa 17 on esimerkki tietojen syöttämisestä tauluun, johon tietoja haettiin dbo.pelaajat_kaikki-taulusta. Tiedot on haettu kahdessa osassa, koska kaikkien pelaajien etunimiä ei ole tiedossa.



```
-- KOTIOTTELUT
INSERT INTO OTTELU (OTTELU_ID, AJANKOHTA, VASTUSTAJA_ID, LIIGA_ID, YLEISÖMÄÄRÄ, EROTUOMARI_ID, STADION_ID, KAUPUNKI_ID,
                    KOTIOTTELU, MAALIT_KOTI, MAALIT_VIERAS, LISÄTIEDOT, TULOSKOODI)
SELECT od.Ottelu_ID, od.Pvm, V.VASTUSTAJA_ID, 1, CAST(CAST (od.Yleisö AS NUMERIC) AS INT), E.EROTUOMARI_ID, S.STADION_ID, K.KAUPUNKI_ID,
       1, od.HIFK_maalit, od.Vastustaja_maalit, null, 1
FROM otteludata od
LEFT JOIN VASTUSTAJA V ON od.Vastustaja = V.NIMI
LEFT JOIN EROTUOMARI E ON od.Tuomari = E.NIMI_RAAKADATA
LEFT JOIN STADION S ON od.Kenttä = S.NIMI
LEFT JOIN KAUPUNKI K ON od.Kaupunki = K.NIMI
WHERE od.K_V = '1'
;

-- VIERASOTTELUT
INSERT INTO OTTELU (OTTELU_ID, AJANKOHTA, VASTUSTAJA_ID, LIIGA_ID, YLEISÖMÄÄRÄ, EROTUOMARI_ID, STADION_ID, KAUPUNKI_ID,
                    KOTIOTTELU, MAALIT_KOTI, MAALIT_VIERAS, LISÄTIEDOT, TULOSKOODI)
SELECT od.Ottelu_ID, od.Pvm, V.VASTUSTAJA_ID, 1, CAST(CAST (od.Yleisö AS NUMERIC) AS INT), E.EROTUOMARI_ID, S.STADION_ID, K.KAUPUNKI_ID,
       0, od.Vastustaja_maalit, od.HIFK_maalit, null, 1
FROM otteludata od
LEFT JOIN VASTUSTAJA V ON od.Vastustaja = V.NIMI
LEFT JOIN EROTUOMARI E ON od.Tuomari = E.NIMI_RAAKADATA
LEFT JOIN STADION S ON od.Kenttä = S.NIMI
LEFT JOIN KAUPUNKI K ON od.Kaupunki = K.NIMI
WHERE od.K_V IS NULL
;

-- KUN KAIKKI TIEDOT ON LISÄTTY, VOIDAAN POISTAA EROTUOMARI-TAULUSTA RAAKADATANIMI
ALTER TABLE EROTUOMARI DROP COLUMN NIMI_RAAKADATA;
```

Kuva 18. Tietojen syöttäminen dbo.OTTELU-tauluun.

Kuvassa 18 on esimerkki tietojen syöttämisestä tauluun, johon tiedot haetaan dbo.otteludata-taulusta. Koti- ja vierasottelut täytyi siirtää erikseen, koska alkuperäisessä datassa ne olivat eriteltynä HIFK:n ja vastustajan maaleihin, eikä esimerkiksi koti- ja vierasmaaleihin.

9 LOPUKSI

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa tietokanta ulkopuolisen tahon luomaa web-sivua varten. Alkuperäinen data oli Excel-taulukoissa epärakenteellisesti, mutta nyt data on relaatiotietokannassa rakenteellisessa muodossa. Työn suurimpana haasteena oli nimenomaan alkuperäisen datan ei-rakenteellisten tietojen vieminen tietokantaan.

Työtä tehdessäni ymmärsin yhä paremmin, miten tärkeä osuus suunnitteluvaihe on tietokantaa tehdessä. Datan käsittely oli yllättävän monimutkaista ja ajoittain haastavaa, joten pyrin dokumentoimaan kaiken siihen liittyvän työn ja pitämään dokumentoinnin ajan tasalla, jotta voisin tarkistaa ja korjata asioita helpommin. Mielestäni dokumentointi oli datan käsittelyvaiheessa todella tärkeä ja onnistunut asia. Jatkossa aion perehtyä yhä lisää tietokannan suunnitteluun ja kehittyä siinä.

Jatkossa web-sivua ja tietokantaa voisi kehittää niin, että siihen voisi helposti lisätä dataa myös tulevista peleistä ja pelaajista. Tähän tarvitsisi luoda käyttöliittymä tietojen lisäämiselle ja päivittämiselle. Toinen kehitysmahdollisuus olisi automatisoida niin, että data päivittyisi itsestään aina otteluiden jälkeen, eli tiedot täytyisi hakea suoraan Palloliiton rajapinnasta. Lisäksi tietokantaan on tarkoitus lisätä uusimmilta vuosilta ja jatkossa tietoja, joita tässä vaiheessa ei vielä ole, kuten varoitukset, kentältäpoistot, peliminuutit, spesifimmät pelaajatiedot ja Suomen Cupin ottelut.

LÄHTEET

Custer, C, 2020. 15 Python libraries for data science you should know. Viitattu 11.9.2021 https://www.dataquest.io/blog/15-python-libraries-for-data-science/

Eastwood, B, 2020. The 10 most popular programming languages to learn in 2020. Viitattu 11.9.2021 https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/

Hovi, J. 2018. Data-alan termien selitykset ja kuvaukset. Viitattu 18.10.2021 https://www.arihovi.com/3274-2/

Hughes, A. 2019. Microsoft SQL Server https://searchdatamanagement.techtarget.com/definition/SQL-Server

Lucidchart 2021. Viitattu 20.10.2021 https://www.lucidchart.com/pages/er-diagrams/#section_0

MongoDB 2021. Viitattu 27.10.2021 https://www.mongodb.com/compare/relational-vs-non-relational-databases

Naeem, T. 2019. Viitattu 25.10.2021 https://www.astera.com/type/blog/all-you-need-to-know-about-database-design/

Norton, K. 2020. Data Definition & Meaning. Viitattu 1.9.2021 https://www.webopedia.com/TERM/D/data.html

Pandas 2021. About Pandas. Viitattu 11.9.2021 https://pandas.pydata.org/about/index.html

Peterson, R. 2021. What is Normalization in DBMS (SQL)? Viitattu 26.10.2021 https://www.guru99.com/database-normalization.html

Peterson, R. 2021. What is SQL Server? Introduction, History, Types, Versions. Viitattu 28.10.2021 https://www.guru99.com/sql-server-introduction.html

Petrovic, B. 2019 Implementing SQL data types https://www.sqlshack.com/implementing-sql-data-types/

Prakteek, T. 2020. What is Normalization in SQL and what are its types? Viitattu 16.10.2021 https://www.edureka.co/blog/normalization-in-sql/#1stNF

Pramanick, S. 2019. History of Python. Viitattu 4.9.2021 https://www.geeksforgeeks.org/history-of-python/

Simplilearn 2020. What is data. Viitattu 16.8.2021 https://www.simplilearn.com/what-is-data-article

Sedkakoui, S. 2018. Data Analytics and Big Data. New Jersey: John Wiley & Sons. Viitattu 12.8.2021. https://ebookcentral.proquest.com/lib/samk/reader.action?docID=5401178

Tutorialspoint 2021. What is SQL? Viitattu 2.9.2021 https://www.tutorialspoint.com/sql/sql-overview.htm

Tutorialspoint 2021. SQL – Primary Key. Viitattu 9.11.2021 https://www.tutorialspoint.com/sql/sql-primary-key.htm

LIITE 1

Tietokantakaava

```
VAR OTTELU BASE RELATION_TYPE (
 OTTELU_ID INT NOT NULL,
 AJANKOHTA [DATE] NOT NULL,
 YLEISOMAARA [INT] NULL,
 KOTIOTTELU [BIT] NOT NULL,
 MAALIT_KOTI [INT] NOT NULL,
 MAALIT_VIERAS [INT] NOT NULL,
 LISATIEDOT [VARCHAR](150) NULL,
 VASTUSTAJA_ID INT NOT NULL,
 LIIGA_ID INT NOT NULL,
 KAUPUNKI_ID INT NULL,
 STADION_ID INT NULL,
 EROTUOMARI_ID INT NULL,
 TULOSKOODI INT NOT NULL )
PRIMARY KEY {OTTELU_ID}
FOREIGN KEY {VASTUSTAJA_ID} REFERENCES VASTUSTAJA
FOREIGN KEY {LIIGA_ID} REFERENCES LIIGA
FOREIGN KEY {KAUPUNKI_ID} REFERENCES KAUPUNKI
FOREIGN KEY {STADION_ID} REFERENCES STADION
FOREIGN KEY {EROTUOMARI_ID} REFERENCES EROTUOMARI
FOREIGN KEY {TULOSKOODI} REFERENCES TULOSKOODIT;

VAR KAUPUNKI BASE RELATION_TYPE (
 KAUPUNKI_ID INT NOT NULL,
 NIMI VARCHAR(50) NOT NULL )
PRIMARY KEY {KAUPUNKI_ID};

VAR EROTUOMARI BASE RELATION_TYPE (
 EROTUOMARI_ID INT NOT NULL,
 ETUNIMI VARCHAR(50) NULL,
 SUKUNIMI VARCHAR(50) NOT NULL)
PRIMARY KEY {EROTUOMARI_ID};

VAR TULOSKOODIT BASE RELATION_TYPE (
 TULOSKOODI_ID INT NOT NULL,
 NIMI VARCHAR(50) NOT NULL  )
PRIMARY KEY {TULOSKOODI_ID};

VAR LIIGA BASE RELATION_TYPE (
 LIIGA_ID INT NOT NULL,
 NIMI VARCHAR(50) NOT NULL )
PRIMARY KEY {LIIGA_ID};

VAR PELAAJA BASE RELATION_TYPE (
 PELAAJA_ID INT NOT NULL,
 ETUNIMI VARCHAR(50) NULL,
 SUKUNIMI VARCHAR(50) NOT NULL,
 ENSISIJAINEN_PELIPAIKKA VARCHAR(50) NULL,
 SYNTYMA_AIKA DATE NULL,
 KANSALAISUUS VARCHAR(50) NULL )
PRIMARY KEY {PELAAJA_ID};

VAR VASTUSTAJA BASE RELATION_TYPE (
 VASTUSTAJA_ID INT NOT NULL
 NIMI VARCHAR(50) NOT NULL,
 KOTISTADION_ID INT NULL )
PRIMARY KEY {VASTUSTAJA_ID}
FOREIGN KEY {KOTISTADION_ID} REFERENCES STADION;

VAR STADION BASE RELATION_TYPE (
 STADION_ID INT NOT NULL
 NIMI VARCHAR(50) NOT NULL )
PRIMARY KEY {STADION_ID};
```

```
65  VAR STADION_ALUSTA BASE RELATION_TYPE (
66   STADION_ID INT NOT NULL,
67   ALUSTA_ID INT NOT NULL
68   ALUSTA VARCHAR(15) NOT NULL,
69   ENSIMMAINEN_PAIVA DATE NULL,
70   VIIMEINEN_PAIVA DATE NULL )
71  PRIMARY KEY (STADION_ID, ALUSTA_ID)
72  FOREIGN KEY (STADION_ID) REFERENCES STADION
73  ON UPDATE CASCADE ON DELETE CASCADE;
74
75  VAR MAALI BASE RELATION_TYPE (
76   OTTELU_ID INT NOT NULL,
77   MAALI_ID INT NOT NULL,
78   PELIMINUUTTI INT NULL,
79   RANGAISTUSPOTKU BIT NULL,
80   OMA_MAALI BIT NULL )
81  PRIMARY KEY (OTTELU_ID, MAALI_ID)
82  FOREIGN KEY (OTTELU_ID) REFERENCES OTTELU
83  ON UPDATE CASCADE ON DELETE CASCADE
84  FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA
85  ON UPDATE CASCADE ON DELETE CASCADE;
86
87  VAR PELAAJATILASTOT BASE RELATION_TYPE (
88   OTTELU_ID INT NOT NULL,
89   PELAAJA_ID INT NOT NULL,
90   TILASTO_ID INT NOT NULL,
91   AVAUKSESSA BIT NULL,
92   VAIHDOSTA_SISAAN BIT NULL,
93   VAIHDOSSA BIT NULL,
94   KELTAISET INT NULL,
95   PUNAISET INT NULL,
96   PELIMINUUTIT INT NULL )
97  PRIMARY KEY (OTTELU_ID, PELAAJA_ID, TILASTO_ID)
98  FOREIGN KEY (OTTELU_ID) REFERENCES OTTELU
99  ON UPDATE CASCADE ON DELETE CASCADE,
100 FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA
101 ON UPDATE CASCADE ON DELETE CASCADE;
102
103 VAR TILASTO79_87 BASE RELATION_TYPE (
104  PELAAJA_ID INT NOT NULL,
105  TILASTO_ID INT NOT NULL
106  OTTELUT INT NULL,
107  MAALIT INT NULL )
108 PRIMARY KEY (PELAAJA_ID, TILASTO_ID)
109 FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA
110 ON UPDATE CASCADE ON DELETE CASCADE;
```

LIITE 2

Taulujen luomisen SQL-lauseet

```
CREATE TABLE [dbo].[PELAAJA](
    [PELAAJA_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [ETUNIMI] [VARCHAR](50) NULL,
    [SUKUNIMI] [VARCHAR](50) NOT NULL,
    [ENSISIJAINEN_PELIPAIKKA] [VARCHAR](50) NULL,
    [SYNTYMÄ_AIKA] DATE NULL,
    [KANSALAISUUS] VARCHAR(50) NULL,
    )

CREATE TABLE [dbo].[LIIGA](
    [LIIGA_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [LIIGA] [VARCHAR](50) NOT NULL,
    [PÄÄSARJA] [BIT] NOT NULL
    )

CREATE TABLE [dbo].[TULOSKOODIT](
    [TULOSKOODI_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [NIMI] [VARCHAR](30) NOT NULL
    )

CREATE TABLE [dbo].[EROTUOMARI](
    [EROTUOMARI_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [ETUNIMI] [VARCHAR](50) NULL,
    [SUKUNIMI] [VARCHAR](50) NOT NULL
    )

CREATE TABLE [dbo].[KAUPUNKI](
    [KAUPUNKI_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [NIMI] [VARCHAR](30) NOT NULL
    )

CREATE TABLE [dbo].[STADION](
    [STADION_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [NIMI] [VARCHAR](50) NOT NULL
    )
```

```
CREATE TABLE [dbo].[STADION_ALUSTA](
    [ALUSTA_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [STADION_ID] [INT] PRIMARY KEY NOT NULL,
    [ALUSTA] [VARCHAR](15) NOT NULL,
    [ENSIMMAINEN_PAIVA] [DATE] NULL,
    [VIIMEINEN_PAIVA] [DATE] NULL,
    CONSTRAINT PK_STADION_ALUSTA PRIMARY KEY (ALUSTA_ID, STADION_ID),
    CONSTRAINT FK_STADION_ALUSTA_STADION_ID
  FOREIGN KEY (STADION_ID) REFERENCES STADION(STADION_ID)
  ON UPDATE CASCADE ON DELETE CASCADE
    )


CREATE TABLE [dbo].[VASTUSTAJA](
    [VASTUSTAJA_ID] [INT] PRIMARY KEY IDENTITY(1,1) NOT NULL,
    [NIMI] [VARCHAR](50) NOT NULL,
    [KOTISTADION_ID] [INT] NULL,
    CONSTRAINT FK_VASTUSTAJA_KOTISTADION_ID
  FOREIGN KEY (KOTISTADION_ID) REFERENCES STADION(STADION_ID)
    )

CREATE TABLE [dbo].[OTTELU](
    [OTTELU_ID] [INT] PRIMARY KEY NOT NULL, -- Tähän ei lisätä automaattisesti ottelu_ID:tä.
    [AJANKOHTA] [DATE] NOT NULL,
    [VASTUSTAJA_ID] [INT] NOT NULL,
    [LIIGA_ID] [INT] NOT NULL,
    [YLEISOMAARA] [INT] NULL,
    [EROTUOMARI_ID] [INT] NULL,
    [STADION_ID] [INT] NULL,
    [KAUPUNKI_ID] [INT] NULL,
    [KOTIOTTELU] [BIT] NOT NULL,
    [MAALIT_KOTI] [INT] NOT NULL,
    [MAALIT_VIERAS] [INT] NOT NULL,
    [LISATIEDOT] [VARCHAR](150) NULL,
    [TULOSKOODI] [INT] NOT NULL,
    CONSTRAINT FK_OTTELU_VASTUSTAJA_ID
  FOREIGN KEY (VASTUSTAJA_ID) REFERENCES VASTUSTAJA(VASTUSTAJA_ID),
  CONSTRAINT FK_OTTELU_EROTUOMARI_ID
  FOREIGN KEY (EROTUOMARI_ID) REFERENCES EROTUOMARI(EROTUOMARI_ID),
  CONSTRAINT FK_OTTELU_STADION_ID
  FOREIGN KEY (STADION_ID) REFERENCES STADION(STADION_ID),
  CONSTRAINT FK_OTTELU_KAUPUNKI_ID
  FOREIGN KEY (KAUPUNKI_ID) REFERENCES KAUPUNKI(KAUPUNKI_ID),
  CONSTRAINT FK_OTTELU_LIIGA_ID
  FOREIGN KEY (LIIGA_ID) REFERENCES LIIGA(LIIGA_ID),
  CONSTRAINT FK_OTTELU_TULOSKOODI_ID
  FOREIGN KEY (TULOSKOODI) REFERENCES TULOSKOODIT(TULOSKOODI_ID)
    )
```

```
CREATE TABLE [dbo].[PELAAJATILASTOT](
    [TILASTO_ID] [INT] IDENTITY(1,1) NOT NULL,
    [OTTELU_ID] [INT] NOT NULL,
    [PELAAJA_ID] [INT] NOT NULL,
    [AVAUKSESSA] [BIT] NULL,
    [VAIHDOSTA_SISAAN] [BIT] NULL,
    [VAIHDOSSA] [BIT] NULL,
    [KELTAISET] [INT] NULL,
    [PUNAISET] [INT] NULL,
    [PELIMINUUTIT] [INT] NULL,
  CONSTRAINT PK_PELAAJATILASTOT PRIMARY KEY (TILASTO_ID, OTTELU_ID, PELAAJA_ID)
  CONSTRAINT FK_PELAAJATILASTOT_OTTELU_ID
  FOREIGN KEY (OTTELU_ID) REFERENCES OTTELU(OTTELU_ID)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT FK_PELAAJATILASTOT_PELAAJA_ID
  FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA(PELAAJA_ID)
  ON UPDATE CASCADE ON DELETE CASCADE
    )


CREATE TABLE [dbo].[MAALI](
    [MAALI_ID] [INT] IDENTITY(1,1) NOT NULL,
    [OTTELU_ID] [INT] NOT NULL,
    [PELAAJA_ID] [INT] NULL,
    [PELIMINUUTTI] [INT] NULL,
    [RANGAISTUSPOTKU] [BIT] NULL,
    [OMA_MAALI] [BIT] NULL,
  CONSTRAINT PK_MAALI PRIMARY KEY (MAALI_ID, OTTELU_ID),
  CONSTRAINT FK_MAALI_OTTELU_ID
  FOREIGN KEY (OTTELU_ID) REFERENCES OTTELU(OTTELU_ID)
  ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT FK_MAALI_PELAAJA_ID
  FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA(PELAAJA_ID)
  ON UPDATE CASCADE ON DELETE CASCADE
    )


CREATE TABLE [dbo].[TILASTO79_87](
    [TILASTO_ID] [INT] IDENTITY(1,1) NOT NULL,
    [PELAAJA_ID] [INT] NOT NULL,
    [OTTELUT] [INT] NULL,
    [MAALIT] [INT] NULL,
  CONSTRTAINT PK_TILASTO79_87 PRIMARY KEY(TILASTO_ID, PELAAJA_ID),
  CONSTRAINT FK_TILASTO79_87_PELAAJA_ID
  FOREIGN KEY (PELAAJA_ID) REFERENCES PELAAJA(PELAAJA_ID)
  ON UPDATE CASCADE ON DELETE CASCADE
    )
```

LIITE 3

SQL-lauseet tietojen lisäämisestä lopullisiin tietokantatauluihin

Tietojen syöttäminen LIIGA-tauluun. Suomalaisen jalkapallon pääsarjalla on ollut historiassa useampi nimi, nykyinen Veikkausliiga on ollut käytössä vuodesta 1990 alkaen. Taulussa on sarake ”PÄÄSARJA”, jotta myöhemmin voi tarkastella erikseen pääsarjatietoja.

```
-- Lisätään vain sarjojen nimet ja tieto pääsarjasta. Tämä on yksinkertainen taulu

    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Veikkausliiga', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Ykkönen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kakkonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kolmonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Nelonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Vitonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Kutonen', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Seiska', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Mestaruussarja', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('SM-kilpailu', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Suomen Cup', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Suomensarja', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('SM-sarja', 1);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('I-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('II-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('III-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('IV-divisioona', 0);
    INSERT INTO LIIGA (LIIGA, PÄÄSARJA) VALUES ('Palloliiton Mestaruussarja', 1);

```

TULOSKOODIT-taulu sisältää vain seitsemän riviä. Alkuperäisessä datassa oli eritelty vain HIFK:n maalit ja vastustajan maalit, joten näitä tietoja ei ollut saatavilla. OTTELU-tauluun on syötetty sarake, joka viittaa tähän tauluun.

```
-- TULOSKOODIT - Näihin vain lisätään tuloskoodit, tämä on hyvin yksinkertainen taul

insert into TULOSKOODIT (NIMI) values ('Voitto');
insert into TULOSKOODIT (NIMI) values ('Tasapeli');
insert into TULOSKOODIT (NIMI) values ('Tappio');
insert into TULOSKOODIT (NIMI) values ('Voitto jatko-ottelussa');
insert into TULOSKOODIT (NIMI) values ('Voitto pilkuilla');
insert into TULOSKOODIT (NIMI) values ('Tappio jatko-ottelussa');
insert into TULOSKOODIT (NIMI) values ('Tappio pilkuilla');

```

Tietojen syöttäminen KAUPUNKI-tauluun. Taulun tärkeys nousee esille vanhoissa otteluissa, joissa on tiedossa ottelupaikkakunta, mutta ei kenttää/stadionia. Taulussa on vain ID ja kaupungin/kunnan nimi.

```
INSERT INTO KAUPUNKI (NIMI)
SELECT DISTINCT KAUPUNKI FROM OTTELUDATA;
```

Tietojen syöttäminen PELAAJA-tauluun muiden kuin maalivahtien osalta. Tämä tehtiin kahdessa vaiheessa, koska joidenkin pelaajien etunimeä ei ollut tiedossa.

```
-- ETUNIMI
WITH ETUNIMI (ETU_NIMI, COLUMN_NAME)
AS
(
    SELECT RIGHT(COLUMN_NAME,LEN(COLUMN_NAME)-CHARINDEX(' ',COLUMN_NAME)) AS ETU_NIMI, COLUMN_NAME
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = 'pelaajat_kaikki'
    AND COLUMN_NAME != 'Ottelu_ID'
    AND COLUMN_NAME NOT LIKE '%(mv)'
    AND CHARINDEX(' ',COLUMN_NAME) <> 0
    AND COLUMN_NAME NOT LIKE '%\ %\ %' ESCAPE '\'' -- Kaksi heittomerkkiä, koodia ajaessa vain yksi
)
INSERT INTO PELAAJA_ETUNIMI (ETUNIMI, KOKONIMI)
SELECT LEFT(ETU_NIMI, ISNULL(NULLIF(CHARINDEX(' ', ETU_NIMI) - 1, -1), LEN(ETU_NIMI))) AS ETU_NIMI, COLUMN_NAME AS KOKONIMI
FROM ETUNIMI
;

-- Seuraavaksi pelkät sukunimet sisältävät pelaajat (näitä ei ole montaa)

INSERT INTO PELAAJA_SUKUNIMI (KOKONIMI, SUKUNIMI)
SELECT COLUMN_NAME AS KOKONIMI,  LEFT(COLUMN_NAME, ISNULL(NULLIF(CHARINDEX(' ', COLUMN_NAME) - 1, -1), LEN(COLUMN_NAME))) AS SUKUNIMI
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'pelaajat_kaikki'
AND COLUMN_NAME != 'Ottelu_ID'
AND COLUMN_NAME NOT LIKE '%(mv)'
AND COLUMN_NAME NOT LIKE '%\ %' ESCAPE '\'' -- 4 -- Kaksi heittomerkkiä, koodia ajaessa vain yksi
;

-- Lopuksi lisätään nämä oikeaan PELAAJA-tauluun ja poistetaan välitaulut.
-- Käytetään LEFT JOIN, koska jokaisella pelaajalla ei ole etunimeä (Esim. "Vitinho").
INSERT INTO PELAAJA (SUKUNIMI, ETUNIMI, KOKONIMI)
SELECT A.SUKUNIMI, B.ETUNIMI, A.KOKONIMI
FROM PELAAJA_SUKUNIMI A LEFT JOIN PELAAJA_ETUNIMI B ON A.KOKONIMI = B.KOKONIMI -- 616
;

-- Poistetaan välitaulut:
DROP TABLE PELAAJA_SUKUNIMI;
DROP TABLE PELAAJA_ETUNIMI;
```

Tietojen syöttäminen EROTUOMARI-tauluun. Tauluun parsittiin erotuomarin nimestä etu- ja sukunimi. Lisäksi mukaan otettiin alkuperäinen tuomarin nimi otteludata-taulusta, koska sitä tarvitaan myöhemmin OTTELU-taulussa.

```
CREATE TABLE EROTUOMARI_ETUNIMI (KOKONIMI VARCHAR(100), ETUNIMI VARCHAR(100));

WITH ETUNIMI (ETUNIMI_EROTUOMARI, TUOMARI)
AS
(
    SELECT DISTINCT RIGHT(tuomari,LEN(tuomari)-CHARINDEX(' ',tuomari)) AS ETUNIMI_EROTUOMARI, tuomari
    FROM otteludata
    WHERE tuomari IS NOT NULL
)
INSERT INTO EROTUOMARI_ETUNIMI (KOKONIMI, ETUNIMI)
SELECT TUOMARI AS KOKONIMI, LEFT(ETUNIMI_EROTUOMARI, ISNULL(NULLIF(CHARINDEX('_',
        ETUNIMI_EROTUOMARI) - 1, -1), LEN(ETUNIMI_EROTUOMARI))) AS ETUNIMI
FROM ETUNIMI
;

CREATE TABLE EROTUOMARI_SUKUNIMI (KOKONIMI VARCHAR(100), SUKUNIMI VARCHAR(100));

INSERT INTO EROTUOMARI_SUKUNIMI (KOKONIMI, SUKUNIMI)
SELECT tuomari AS KOKONIMI, LEFT(tuomari, ISNULL(NULLIF(CHARINDEX(' ', tuomari) - 1, -1), LEN(tuomari))) AS SUKUNIMI
FROM otteludata
WHERE tuomari is not null
;

-- LISÄTÄÄN VÄLIVAIHEEN HARAKE EROTUOMARITAULUUN, JOSSA ON ALKUPERÄINEN RAAKADATANIMI
ALTER TABLE EROTUOMARI ADD NIMI_RAAKADATA VARCHAR(100);

-- Lisätään tiedot oikeaan tauluun
INSERT INTO EROTUOMARI (SUKUNIMI, ETUNIMI, NIMI_RAAKADATA)
SELECT DISTINCT A.SUKUNIMI, B.ETUNIMI, B.KOKONIMI
FROM EROTUOMARI_SUKUNIMI A JOIN EROTUOMARI_ETUNIMI B ON A.KOKONIMI = B.KOKONIMI
;

DROP TABLE EROTUOMARI_ETUNIMI;
DROP TABLE EROTUOMARI_SUKUNIMI;
```

Tietojen syöttäminen STADION-tauluun. Tässä taulussa on vain Id ja stadionin/kentän nimi. Koska yhdellä kentällä on voinut olla erilaisia alustoja ja ne voivat myös tulevaisuudessa muuttua, tiedot ovat erikseen taulussa STADION_ALUSTA.

```
5   INSERT INTO STADION (NIMI)
6   SELECT DISTINCT Kenttä FROM OTTELUDATA WHERE Kenttä IS NOT NULL;
```

Tietojen syöttäminen STADION_ALUSTA-tauluun. Tiedot haettiin otteludatasta (Kenttä) ja STADION-taulusta (STADION_ID).

```
4   INSERT INTO STADION_ALUSTA (STADION_ID, ALUSTA)
5   SELECT STADION_ID, ALUSTA
6   from otteludata O JOIN STADION S ON O.Kenttä = S.NIMI
7   group by kenttä, STADION_ID, ALUSTA
8   HAVING count(distinct alusta) = 1 AND KENTTÄ IS NOT NULL
```

STADION_ALUSTA-tauluun syötettiin manuaalisesti tietoja alustojen vaihdoista. Ne alustat, joissa on vain ensimmäinen päivä tiedossa, kuvaavat nykyistä alustaa. Alustat ovat aina vaihtuneet kausien välissä, joten päivämääriksi on asetettu vuoden ensimmäiset ja viimeiset päivät.

```
-- lisätään manuaalisesti tiedot päivämääristä niihin, joissa on useampi kuin yksi alusta:

-- Kysely, jolla saa tiedot STADION_ALUSTA -taulun staatioelista, jolla on alustoja enemmän kuin yksi, mutta ei päivämäärä asetettuna:
select a.*, b.nimi from stadion_alusta a join stadion b on a.stadion_id = b.stadion_id
where a.stadion_id in (
    SELECT STADION_ALUSTA.STADION_ID
    FROM STADION_ALUSTA JOIN STADION ON STADION_ALUSTA.STADION_ID = STADION.STADION_ID
    GROUP BY STADION_ALUSTA.STADION_ID HAVING COUNT(*) > 1
)
and (a.ENSIMMÄINEN_PÄIVÄ is null and a.VIIMEINEN_PÄIVÄ is null)


-- Arto Tolsa
UPDATE STADION_ALUSTA SET ENSIMMÄINEN_PÄIVÄ = '01-01-2015' WHERE STADION_ID = 4 AND ALUSTA = 'tekonurmi';
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2014' WHERE STADION_ID = 4 AND ALUSTA = 'nurmi';
--Ikhtaan kenttä:
UPDATE STADION_ALUSTA SET ENSIMMÄINEN_PÄIVÄ = '01-01-2015' WHERE STADION_ID = 104 AND ALUSTA = 'tekonurmi';
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2014' WHERE STADION_ID = 104 AND ALUSTA = 'nurmi';
--Wiklöf Holding Arena:
UPDATE STADION_ALUSTA SET ENSIMMÄINEN_PÄIVÄ = '01-01-2020' WHERE STADION_ID = 122 AND ALUSTA = 'tekonurmi';
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2019' WHERE STADION_ID = 122 AND ALUSTA = 'nurmi';
--Myyrmäen jalkapallostadion:
UPDATE STADION_ALUSTA SET ENSIMMÄINEN_PÄIVÄ = '01-01-2004' WHERE STADION_ID = 135 AND ALUSTA = 'tekonurmi';
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2003' WHERE STADION_ID = 135 AND ALUSTA = 'nurmi';
-- Finlajamaki - nykyään tekonurmi
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2015' WHERE STADION_ID = 85 AND ALUSTA = 'hiekka';
INSERT INTO STADION_ALUSTA (STADION_ID, ALUSTA, ENSIMMÄINEN_PÄIVÄ) VALUES (85, 'tekonurmi', '01-01-2016')
-- Kisapuisto - nykyään tekonurmi
INSERT INTO STADION_ALUSTA (STADION_ID, ALUSTA, ENSIMMÄINEN_PÄIVÄ) VALUES (120, 'tekonurmi', '01-01-2018')
UPDATE STADION_ALUSTA SET VIIMEINEN_PÄIVÄ = '12-31-2017' WHERE STADION_ID = 120 AND ALUSTA = 'nurmi';
```

Tietojen syöttäminen PELAAJA-tauluun ensin maalivahtien osalta, koska alkuperäisessä datassa maalivahdit oli merkitty ”Sukunimi Etunimi (mv)”.

```
-- Lisätään sarake "KOKONIMI" PELAAJA-tauluun. Tätä tarvitaan seuraavissa vaiheissa.
ALTER TABLE PELAAJA ADD KOKONIMI VARCHAR(100);
-- Etunimet
CREATE TABLE MV_PELAAJA_ETUNIMI (KOKONIMI VARCHAR(100), ETUNIMI VARCHAR(100));

WITH ETUNIMI (ETUNIMI_MV, COLUMN_NAME)
AS
(
    SELECT RIGHT(COLUMN_NAME,LEN(COLUMN_NAME)-CHARINDEX(' ',COLUMN_NAME)) AS ETUNIMI_MV, COLUMN_NAME
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = 'pelaajat_taikzi'
    AND COLUMN_NAME LIKE '%(mv)'
)
INSERT INTO MV_PELAAJA_ETUNIMI (KOKONIMI, ETUNIMI)
SELECT COLUMN_NAME AS KOKONIMI, LEFT(ETUNIMI_MV, ISNULL(NULLIF(CHARINDEX(' ', ETUNIMI_MV) - 1, -1), LEN(ETUNIMI_MV))) AS ETUNIMI
FROM ETUNIMI
;

CREATE TABLE MV_PELAAJA_SUKUNIMI (KOKONIMI VARCHAR(100), SUKUNIMI VARCHAR(100));

INSERT INTO MV_PELAAJA_SUKUNIMI (KOKONIMI, SUKUNIMI)
SELECT COLUMN_NAME AS KOKONIMI, LEFT(COLUMN_NAME, ISNULL(NULLIF(CHARINDEX(' ', COLUMN_NAME) - 1, -1), LEN(COLUMN_NAME))) AS SUKUNIMI
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'pelaajat_taikzi'
AND COLUMN_NAME LIKE '%(mv)'
;


-- Lisätään tiedot maalivahdeista
INSERT INTO PELAAJA (SUKUNIMI, ETUNIMI, ENSISIJAINEN_PELIPAIKKA, KOKONIMI)
SELECT A.SUKUNIMI, B.ETUNIMI, 'Maalivahti', A.KOKONIMI
FROM MV_PELAAJA_SUKUNIMI A JOIN MV_PELAAJA_ETUNIMI B ON A.KOKONIMI = B.KOKONIMI
;

-- Poistetaan välitaulut:
DROP TABLE MV_PELAAJA_SUKUNIMI;
DROP TABLE MV_PELAAJA_ETUNIMI;
```

Tietojen syöttäminen VASTUSTAJA-tauluun.

```
INSERT INTO VASTUSTAJA (NIMI)
SELECT DISTINCT Vastustaja FROM Otteludata;
```

Tietojen syöttäminen OTTELU-tauluun. Koti- ja vierasottelut täytyi siirtää erikseen, koska alkuperäisessä datassa ne olivat eriteltynä HIFK:n ja vastustajan maaleihin.

```
-- KOTIOTTELUT
INSERT INTO OTTELU (OTTELU_ID, AJANKOHTA, VASTUSTAJA_ID, LIIGA_ID, YLEISÖMÄÄRÄ, EROTUOMARI_ID, STADION_ID, KAUPUNKI_ID,
                    KOTIOTTELU, MAALIT_KOTI, MAALIT_VIERAS, LISÄTIEDOT, TULOSKOODI)
SELECT od.Ottelu_ID, od.Pvm, V.VASTUSTAJA_ID, 1, CAST(CAST (od.Yleisö AS NUMERIC) AS INT), E.EROTUOMARI_ID, S.STADION_ID, K.KAUPUNKI_ID,
       1, od.HIFK_maalit, od.Vastustaja_maalit, null, 1
FROM otteludata od
LEFT JOIN VASTUSTAJA V ON od.Vastustaja = V.NIMI
LEFT JOIN EROTUOMARI E ON od.Tuomari = E.NIMI_RAAKADATA
LEFT JOIN STADION S ON od.Kenttä = S.NIMI
LEFT JOIN KAUPUNKI K ON od.Kaupunki = K.NIMI
WHERE od.K_V = 'K'
;

-- VIERASOTTELUT
INSERT INTO OTTELU (OTTELU_ID, AJANKOHTA, VASTUSTAJA_ID, LIIGA_ID, YLEISÖMÄÄRÄ, EROTUOMARI_ID, STADION_ID, KAUPUNKI_ID,
                    KOTIOTTELU, MAALIT_KOTI, MAALIT_VIERAS, LISÄTIEDOT, TULOSKOODI)
SELECT od.Ottelu_ID, od.Pvm, V.VASTUSTAJA_ID, 1, CAST(CAST (od.Yleisö AS NUMERIC) AS INT), E.EROTUOMARI_ID, S.STADION_ID, K.KAUPUNKI_ID,
       0, od.Vastustaja_maalit, od.HIFK_maalit, null, 1
FROM otteludata od
LEFT JOIN VASTUSTAJA V ON od.Vastustaja = V.NIMI
LEFT JOIN EROTUOMARI E ON od.Tuomari = E.NIMI_RAAKADATA
LEFT JOIN STADION S ON od.Kenttä = S.NIMI
LEFT JOIN KAUPUNKI K ON od.Kaupunki = K.NIMI
WHERE od.K_V IS NULL
;

-- NUN NÄMÄ TIEDOT ON LISÄTTY, VOIDAAN POISTAA EROTUOMARI-TAULUSTA RAAKADATANIMI
ALTER TABLE EROTUOMARI DROP COLUMN NIMI_RAAKADATA;
```

OTTELU-tauluun syötettiin tiedot tuloskoodeista (tasapeli, voitto, tappio), sekä sarjoista (LIIGA_ID), koska näitä tietoja ei ollut alkuperäisessä datassa.

```
UPDATE OTTELU SET TULOSKOODI = 2 WHERE MAALIT_KOTI = MAALIT_VIERAS;

UPDATE OTTELU SET TULOSKOODI = 3 WHERE MAALIT_KOTI < MAALIT_VIERAS AND KOTIOTTELU = 1;
UPDATE OTTELU SET TULOSKOODI = 1 WHERE MAALIT_KOTI > MAALIT_VIERAS AND KOTIOTTELU = 1;
UPDATE OTTELU SET TULOSKOODI = 3 WHERE MAALIT_KOTI < MAALIT_VIERAS AND KOTIOTTELU = 0;
UPDATE OTTELU SET TULOSKOODI = 1 WHERE MAALIT_KOTI > MAALIT_VIERAS AND KOTIOTTELU = 0;
-- TÄHÄN LISÄKSI JATKO-OTTELUT JA PILKUT LISÄTÄÄN MANUAALISESTI, KOSKA NIITÄ EI OLE ERITELTY DATASSA.

--MYÖS LIIGA_ID LISÄTÄÄN MANUAALISESTI, KOSKA SILLE EI OLE HAHAKETTA DATASSA. SEN SIJAAN VUODET KERTOVAT SEN HYVIN:
-- Tähän tyyliin:
UPDATE OTTELU SET LIIGA_ID = 9 WHERE YEAR(AJANKOHTA) BETWEEN 1930 AND 1939
UPDATE OTTELU SET LIIGA_ID = 10 WHERE YEAR(AJANKOHTA) = 1940 AND MONTH(AJANKOHTA) = 9
UPDATE OTTELU SET LIIGA_ID = 9 WHERE (YEAR(AJANKOHTA) = 1940 AND MONTH(AJANKOHTA) BETWEEN 9 AND 12) OR YEAR(AJANKOHTA) = 1941
UPDATE OTTELU SET LIIGA_ID = 10 WHERE YEAR(AJANKOHTA) = 1942
UPDATE OTTELU SET LIIGA_ID = 9 WHERE YEAR(AJANKOHTA) BETWEEN 1943 AND 1944
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1945-05-01' AND '1945-06-10'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1945-06-27' AND '1945-07-10'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1945-08-01' AND '1946-07-01'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1946-07-30' AND '1947-06-20'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA = '1947-10-19'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1947-07-01' AND '1947-07-31'
UPDATE OTTELU SET LIIGA_ID = 10 WHERE AJANKOHTA BETWEEN '1947-08-01' AND '1948-06-20'
UPDATE OTTELU SET LIIGA_ID = 9 WHERE (YEAR(AJANKOHTA) = 1948 AND MONTH(AJANKOHTA) > 6) OR YEAR(AJANKOHTA) = 1949
UPDATE OTTELU SET LIIGA_ID = 10 WHERE YEAR(AJANKOHTA) BETWEEN 1950 AND 1957
UPDATE OTTELU SET LIIGA_ID = 9 WHERE YEAR(AJANKOHTA) BETWEEN 1958 AND 1966
UPDATE OTTELU SET LIIGA_ID = 10 WHERE YEAR(AJANKOHTA) BETWEEN 1967 AND 1969
UPDATE OTTELU SET LIIGA_ID = 11 WHERE YEAR(AJANKOHTA) BETWEEN 1970 AND 1972
UPDATE OTTELU SET LIIGA_ID = 14 WHERE YEAR(AJANKOHTA) BETWEEN 1973 AND 1974
UPDATE OTTELU SET LIIGA_ID = 15 WHERE YEAR(AJANKOHTA) BETWEEN 1975 AND 1978
UPDATE OTTELU SET LIIGA_ID = 16 WHERE YEAR(AJANKOHTA)= 1979
UPDATE OTTELU SET LIIGA_ID = 17 WHERE YEAR(AJANKOHTA) BETWEEN 1980 AND 1983
UPDATE OTTELU SET LIIGA_ID = 16 WHERE YEAR(AJANKOHTA) BETWEEN 1984 AND 1987
UPDATE OTTELU SET LIIGA_ID = 1 WHERE YEAR(AJANKOHTA) BETWEEN 1988 AND 1989
UPDATE OTTELU SET LIIGA_ID = 2 WHERE YEAR(AJANKOHTA) BETWEEN 1990 AND 2002
UPDATE OTTELU SET LIIGA_ID = 5 WHERE YEAR(AJANKOHTA) BETWEEN 2003 AND 2005
UPDATE OTTELU SET LIIGA_ID = 4 WHERE YEAR(AJANKOHTA) BETWEEN 2006 AND 2007
UPDATE OTTELU SET LIIGA_ID = 3 WHERE YEAR(AJANKOHTA) BETWEEN 2008 AND 2010
UPDATE OTTELU SET LIIGA_ID = 2 WHERE YEAR(AJANKOHTA) BETWEEN 2011 AND 2012
UPDATE OTTELU SET LIIGA_ID = 1 WHERE YEAR(AJANKOHTA) = 2013
UPDATE OTTELU SET LIIGA_ID = 2 WHERE YEAR(AJANKOHTA) = 2014
UPDATE OTTELU SET LIIGA_ID = 1 WHERE YEAR(AJANKOHTA) BETWEEN 2015 AND 2017
UPDATE OTTELU SET LIIGA_ID = 2 WHERE YEAR(AJANKOHTA) = 2018
UPDATE OTTELU SET LIIGA_ID = 1 WHERE YEAR(AJANKOHTA) BETWEEN 2019 AND 2020
```

Pelaajatilasto-proseduuri, jonka avulla tietoja pelaajan pelatuista otteluista (avauskokoonpano, vaihdosta sisään ja vaihtopenkillä) voidaan siirtää PELAAJATILASTOT-tauluun. Seuraavassa kuvassa 34 esitetään tapa, jolla proseduuria käytetään.

```
CREATE PROCEDURE [Pelaajatilasto]
    @pelaaja varchar(30)
AS BEGIN
    EXEC('
    WITH Q2 (TILASTO, OTTELU_ID)
    AS (
    SELECT ' + @pelaaja + ' AS TILASTO, [Ottelu_ID] FROM pelaajat_kaikki WHERE [' + @pelaaja + '] IN (''E'', ''O'', ''-'', ''a'', ''s'')
    )
    INSERT INTO pelaajatilastot (OTTELU_ID, PELAAJA_ID, AVAUKSESSA, VAIHDOSTA_SISÄÄN, VAIHDOSSA)
     SELECT OTTELU_ID, PELAAJA_ID, 1 AS AVAUKSESSA, 0 AS VAIHDOSTA_SISÄÄN, 0 AS VAIHDOSSA
     FROM Q2 LEFT JOIN PELAAJA ON PELAAJA.NIMIMERKKI = ''' + @pelaaja + '''
     WHERE TILASTO IN (''E'', ''s'') ;

    WITH Q2 (TILASTO, OTTELU_ID)
    AS (
    SELECT ' + @pelaaja + ' AS TILASTO, [Ottelu_ID] FROM pelaajat_kaikki WHERE [' + @pelaaja + '] IN (''E'', ''O'', ''-'', ''a'', ''s'')
    )
     INSERT INTO pelaajatilastot (OTTELU_ID, PELAAJA_ID, AVAUKSESSA, VAIHDOSTA_SISÄÄN, VAIHDOSSA)
     SELECT OTTELU_ID, PELAAJA_ID, 0 AS AVAUKSESSA, 1 AS VAIHDOSTA_SISÄÄN, 0 AS VAIHDOSSA
     FROM Q2 LEFT JOIN PELAAJA ON PELAAJA.NIMIMERKKI = ''' + @pelaaja + '''
     WHERE TILASTO IN (''O'', ''s'') ;

    WITH Q2 (TILASTO, OTTELU_ID)
    AS (
    SELECT ' + @pelaaja + ' AS TILASTO, [Ottelu_ID] FROM pelaajat_kaikki WHERE [' + @pelaaja + '] IN (''E'', ''O'', ''-'', ''a'', ''s'')
    )
     INSERT INTO pelaajatilastot (OTTELU_ID, PELAAJA_ID, AVAUKSESSA, VAIHDOSTA_SISÄÄN, VAIHDOSSA)
     SELECT OTTELU_ID, PELAAJA_ID, 0 AS AVAUKSESSA, 0 AS VAIHDOSTA_SISÄÄN, 1 AS VAIHDOSSA
     FROM Q2 LEFT JOIN PELAAJA ON PELAAJA.NIMIMERKKI = ''' + @pelaaja + '''
     WHERE TILASTO IN (''-'') ;
    ')
END
```

Tietojen syöttäminen PELAAJATILASTOT-tauluun Pelaajatilasto-proseduurin avulla.

```
DECLARE @cnt INT = 1;

DECLARE @pelaajamaara INT;

-- Asetetaan pelaajamäärä, jotta looppi pyörii oikean määrän.
SELECT @Pelaajamaara = COUNT(COLUMN_NAME)+1
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'pelaajat_kaikki' AND COLUMN_NAME <> 'Ottelu_ID'

print @pelaajamaara;

WHILE @cnt < @pelaajamaara -- @pelaajamaara
BEGIN

    DECLARE @player VARCHAR(50);
    -- With hakee aina yhden pelaajan loopissa kerrallaan.
    WITH Q1 (COLUMN_NAME, NUM)
    AS
    (
        SELECT COLUMN_NAME, ROW_NUMBER() OVER(ORDER BY COLUMN_NAME ASC) AS NUM
        FROM INFORMATION_SCHEMA.COLUMNS
        WHERE TABLE_NAME = 'pelaajat_kaikki' AND COLUMN_NAME <> 'Ottelu_ID'
    )
    SELECT @player = [COLUMN_NAME]
    FROM Q1
    WHERE NUM = @cnt

    -- Suoritetaan Pelaajatilasto-proseduuri, jolle ainnetaan parametriksi yksi pelaaja.
    EXEC Pelaajatilasto @Player

   SET @cnt = @cnt + 1;
END;


GO
```

Seuraavassa kolmessa kuvassa esitellään MAALI-tauluun rivien syöttäminen, maalin tietojen syöttäminen otteluista, joissa oli tehty vain yksi maali, sekä esimerkki, jolla syötettiin tietoja otteluista, joissa tehtiin useampi maali. Ottelut, joissa oli useampi kuin yksi maali, syötettiin usealla eri tavalla (esim. peliminuutit sisältävät, omia maaleja sisältävät, rangaistuspotkua sisältävät ja etunimiä sisältävät maalitiedot) MAALI-tauluun, mutta samalla koodipohjalla.

Ensimmäisten tietojen syöttäminen MAALI-tauluun. Tämän koodin avulla MAALI-tauluihin saatiin oikea määrä rivejä per ottelu.

```
DECLARE @ottelu INT = 1;

DECLARE @otteluMaara INT;

-- Asetetaan ottelumäärä, jotta looppi pyörii oikean määrän.

SELECT @otteluMaara = COUNT(*)
FROM OTTELUDATA


WHILE @ottelu <= @otteluMaara
BEGIN

    DECLARE @ottelumaalit INT = 0;
    DECLARE @lisatytMaalit INT = 0;

    SELECT @ottelumaalit = HIFK_maalit
    FROM OTTELUDATA
    WHERE OTTELU_ID = @ottelu

    WHILE @lisatytMaalit < @ottelumaalit
    BEGIN
        INSERT INTO MAALI (OTTELU_ID)
        SELECT OTTELU_ID
        FROM OTTELUDATA
        WHERE OTTELU_ID = @ottelu AND HIFK_maalit <> 0

        SET @lisatytMaalit = @lisatytMaalit+1;
        PRINT @lisatytMaalit;
    END;


   SET @ottelu = @ottelu + 1;
END;


GO
```

Maalitiedot siirrettiin ensin ns. välitauluihin, joista data siirretään lopulliseen tauluun niin, että se yhdistetään maalintekijät-sarakkeen pelaajatietoihin sukunimen avulla. Näin saadaan myös haettua pelaajan id MAALI-taulua varten. Maali_valitauluun haettiin maalit otteluista, joissa HIFK oli tehnyt yhden maalin ja Maali_valitaulu_multi -tauluun maalit otteluista, joissa HIFK oli tehnyt vähintään kaksi maalia. Yksimaalisia otteluita, joissa oli maalintekijätietoja, oli 372 ja yli yksimaalisia 942.

```
create table maali_valitaulu (ottelu_id int, pelaaja_id int, peliminuutti int,
                              rangaistuspotku bit, oma_maali bit, maalintekijät varchar(100))
create table maali_valitaulu_multi (ottelu_id int, pelaaja_id int, peliminuutti int,
                              rangaistuspotku bit, oma_maali bit, maalintekijät varchar(100))
```

Maalien lisääminen välitauluun otteluista, joissa oli tehtynä vain yksi maali. Tässä esimerkissä haettiin rangaistuspotkumaaleja, joissa oli mukana peliminuutit. Kaikki muut yksimaaliset maalintekijätiedot haettiin samalla tyylillä, mutta eri ehdoilla.

```
INSERT INTO MAALI_VALITAULU (ottelu_id, pelaaja_id, peliminuutti, maalintekijät, rangaistuspotku)
select otteludata.ottelu_id, pelaaja.PELAAJA_ID,
SUBSTRING(maalintekijät, CHARINDEX('(', maalintekijät)+1
, CHARINDEX('/',maalintekijät) - CHARINDEX('(', maalintekijät) + Len('/') -2) as minuutti, maalintekijät, 1
from otteludata join pelaaja
on otteludata.maalintekijät like pelaaja.sukunimi + '%'
join pelaajatilastot on pelaaja.PELAAJA_ID = pelaajatilastot.PELAAJA_ID
where HIFK_maalit = 1 and maalintekijät is not null
and maalintekijät like '%([rp])%'
and maalintekijät like '%[0-9]%'
AND maalintekijät not like '%,%'
and maalintekijät not like '% om%'
and pelaajatilastot.OTTELU_ID = otteludata.Ottelu_ID
```

Seuraavassa kolmessa kuvassa on kuvattu maalintekijätietojen parsimista sellaisista otteluista, joissa oli maalintekijätietoja yli yhdelle maalille. Tämä tapahtui käytännössä ensin määrittelemällä, millaisia maaleja haluttiin hakea, jonka jälkeen maalintekijätieto parsittiin useampaan muuttujaan, jotka lisättiin lopulta Maali_valitaulu_multi -tauluun. Tässä esimerkissä on haettu maalit, joissa on merkittynä peliminuutit ja useampi kuin yksi maali, mutta vain yksi maalintekijä, eli esimerkiksi ”Virtanen 4 [14, 19, 76, 89]”.

Määritellään ne ottelut, joista maalintekijätietoja haetaan.

```
603  DECLARE @otteluMaara INT;
604  DECLARE @ottelu INT = 1;
605  SELECT @otteluMaara = COUNT(*)
606  from otteludata
607  where HIFK_maalit > 1 and maalintekijät is not null
608  and maalintekijät not like '%rp%'
609  and maalintekijät like '%[[]%'
610  and maalintekijät not like '% om%'
611  and (maalintekijät like '% 2%'
612  or maalintekijät like '% 3%'
613  or maalintekijät like '% 4%'
614  or maalintekijät like '% 6%'
615  or maalintekijät like '% 5%'
616  )
617  and maalintekijät not like '%],%'
```

Ottelukohtaisten muuttujien määrittely.

```
WHILE @ottelu <= @otteluMaara -- Pääsilmukka pyörii kerran jokaista ottelua kohden
BEGIN
    -- Ottelukohtaiset muuttujat:
    DECLARE @OTTELU_ID INT;
    DECLARE @MAALINTEKIJATIETO VARCHAR(100);
    DECLARE @MINUUTTITIETO VARCHAR(100);
    DECLARE @LISÄTTÄVÄTMINUUTIT varchar(100);

    WITH apu AS
    (
        select ottelu_id, HIFK_maalit, maalintekijät, ROW_NUMBER() OVER(ORDER BY ottelu_id) AS Row
        from otteludata
        where HIFK_maalit > 1 and maalintekijät is not null
        and maalintekijät not like '%rp%'
        and maalintekijät like '%[[]%'
        and maalintekijät not like '% om%'
        and (maalintekijät like '% 2%'
        or maalintekijät like '% 3%'
        or maalintekijät like '% 4%'
        or maalintekijät like '% 6%'
        or maalintekijät like '% 5%'
        )
        and maalintekijät not like '%],%'
    )
    SELECT @OTTELU_ID = ottelu_id, @MAALINTEKIJATIETO = SUBSTRING(maalintekijät, 1, CHARINDEX('[', maalintekijät)-2),
           @MINUUTTITIETO = SUBSTRING(maalintekijät, CHARINDEX('[', maalintekijät), LEN(maalintekijät))
    from apu where row = @ottelu
```

Lopullinen maalintekijätietojen lisääminen maaleittain Maali_valitaulu_multi -tauluun.

Tässä vaiheessa siis maali_valitaulu sisälsi valmiit tiedot MAALI-taulua varten, mutta maali_valitaulu_multi -taulusta puuttui vielä pelaajan id, jonka avulla tiedon pystyy siirtämään lopulliseen MAALI-tauluun. Seuraavassa kuvassa haetaan sukunimen perusteella maalintekijät id ja lopulta kahdessa viimeisessä kuvassa valmiit tiedot siirretään MAALI-tauluun.

```
WITH X (OTTELU_ID, PELAAJA_ID, SUKUNIMI, MAALINTEKIJÄT)
AS
(
SELECT M.OTTELU_ID, P.PELAAJA_ID, P.SUKUNIMI, M.MAALINTEKIJÄT
FROM PELAAJA P JOIN MAALI_VALITAULU_MULTI M ON M.MAALINTEKIJÄT LIKE P.SUKUNIMI + '%'
WHERE M.MAALINTEKIJÄT NOT LIKE '%,%' AND M.MAALINTEKIJÄT <> '???'
AND M.OTTELU_ID IN (SELECT OTTELU_ID FROM OTTELUDATA WHERE YEAR(PVM) BETWEEN 1973 AND 1978)
AND M.OTTELU_ID NOT IN (SELECT A.OTTELU_ID FROM TESTI4 A JOIN TESTI5 B ON A.OTTELU_ID = B.OTTELU_ID
WHERE A.MAALIT > B.MAALIT)
AND MAALINTEKIJÄT <> 'Laine'
)
UPDATE MAALI_VALITAULU_MULTI SET PELAAJA_ID = (SELECT DISTINCT PELAAJA_ID FROM X
WHERE X.OTTELU_ID = MAALI_VALITAULU_MULTI.OTTELU_ID AND
MAALI_VALITAULU_MULTI.MAALINTEKIJÄT LIKE X.SUKUNIMI + '%' AND
MAALI_VALITAULU_MULTI.OMA_MAALI IS NULL AND
MAALI_VALITAULU_MULTI.PELAAJA_ID IS NULL
)
WHERE PELAAJA_ID IS NULL
;
```

Maali_valitaulu_multi- ja MAALI-tauluihin lisättiin sarake MAALI_JARJESTYS, joka kertoo maalin järjestysluvun otteluittain.

```
-- Tehdään maali_valitaulu_multi_jarjestys:
SELECT OTTELU_ID, PELAAJA_ID, PELIMINUUTTI, RANGAISTUSPOTKU, OMA_MAALI, MAALINTEKIJÄT,
       ROW_NUMBER() OVER(PARTITION BY OTTELU_ID ORDER BY PELIMINUUTTI) AS MAALI_JARJESTYS
       INTO MAALI_VALITAULU_MULTI_JARJESTYS
FROM MAALI_VALITAULU_MULTI

-- Lisätään sama tieto MAALI-tauluun:
ALTER TABLE MAALI
ADD MAALI_JARJESTYS INT;

UPDATE MAALI
SET MAALI.MAALI_JARJESTYS = R.MAALI_JARJESTYS
FROM (SELECT MAALI_ID, ROW_NUMBER() OVER(PARTITION BY OTTELU_ID ORDER BY PELIMINUUTTI) AS MAALI_JARJESTYS
      FROM MAALI S) R
WHERE MAALI.MAALI_ID = R.MAALI_ID
```

Viimeinen vaihe, jossa maalintekijätiedot siirrettiin yksitellen MAALI-tauluun ottelun id:n ja maalin järjestystiedon avulla.

```
UPDATE MAALI
SET PELAAJA_ID = (
    SELECT PELAAJA_ID
    FROM MAALI_VALITAULU_MULTI_JARJESTYS B
    WHERE B.OTTELU_ID = MAALI.OTTELU_ID
    AND B.MAALI_JARJESTYS = MAALI.MAALI_JARJESTYS
)
WHERE PELAAJA_ID IS NULL


UPDATE MAALI
SET PELIMINUUTTI = (
    SELECT PELIMINUUTTI
    FROM MAALI_VALITAULU_MULTI_JARJESTYS B
    WHERE B.OTTELU_ID = MAALI.OTTELU_ID
    AND B.MAALI_JARJESTYS = MAALI.MAALI_JARJESTYS
)
WHERE PELIMINUUTTI IS NULL

UPDATE MAALI
SET RANGAISTUSPOTKU = (
    SELECT RANGAISTUSPOTKU
    FROM MAALI_VALITAULU_MULTI_JARJESTYS B
    WHERE B.OTTELU_ID = MAALI.OTTELU_ID
    AND B.MAALI_JARJESTYS = MAALI.MAALI_JARJESTYS
)
WHERE RANGAISTUSPOTKU IS NULL

UPDATE MAALI
SET OMA_MAALI = (
    SELECT OMA_MAALI
    FROM MAALI_VALITAULU_MULTI_JARJESTYS B
    WHERE B.OTTELU_ID = MAALI.OTTELU_ID
    AND B.MAALI_JARJESTYS = MAALI.MAALI_JARJESTYS
)
WHERE OMA_MAALI IS NULL
```

Viimeinen taulu, jälkeenpäin lisätty TILASTO79_87, on rakenteeltaan erilainen, koska se sisältää kyseisten kausien maalimäärät pelaajittain sekä hyvin puutteelliset ottelumäärät pelaajittain. Näin ollen tietoja ei voi lisätä esimerkiksi MAALI-tauluun, koska otteluista, joissa maalit ovat syntyneet, ei ole tietoa. Pelaajien tilastoinnin vuoksi tiedot haluttiin kuitenkin tietokantaan, joten ne lisättiin omaan tauluunsa, joka sisältää tietueen jokaiselle pelaajalle.

Kausien 1979–1987 pelaajatilastot haettiin jälkikäteen omasta csv-tiedostosta, jossa tietoja oli rajatusti saatavilla. Tiedot vietiin Pelaaja-tauluun sekä näiden kausien omaan tilastotauluun, jossa on kerrottu vain maali- ja ottelumäärät pelaajittain. Osa pelaajista oli pelannut muillakin kausilla, joten heidän tietojansa oli jo tietokannassa.

```
select distinct etunimi, sukunimi, pelipaikka,
sum(cast(Ottelut as float)) as ottelut, sum(cast(Maalit as float)) as maalit into VALITIETO79_87
from pelaajat79_87
group by etunimi, sukunimi, pelipaikka
order by sukunimi, etunimi, pelipaikka

-- Osa pelaajista (36/127) on jo pelaaja-taulussa:
select v.etunimi+v.sukunimi
from valitieto79_87 v join pelaaja p on v.etunimi = p.etunimi and v.sukunimi = p.sukunimi


-- Lisätään loputkin:
insert into pelaaja (etunimi, sukunimi, ensisijainen_pelipaikka)
select etunimi, sukunimi, pelipaikka
from valitieto79_87
where etunimi+sukunimi not in (
    select v.etunimi+v.sukunimi
    from valitieto79_87 v join pelaaja p on v.etunimi = p.etunimi and v.sukunimi = p.sukunimi
)

-- Viedään tiedot TILASTO79_87 -tauluun:

INSERT INTO TILASTO79_87
SELECT PELAAJA_ID, OTTELUT, MAALIT
FROM VALITIETO79_87 V JOIN PELAAJA P ON V.ETUNIMI = P.ETUNIMI AND V.SUKUNIMI =P.SUKUNIMI

```