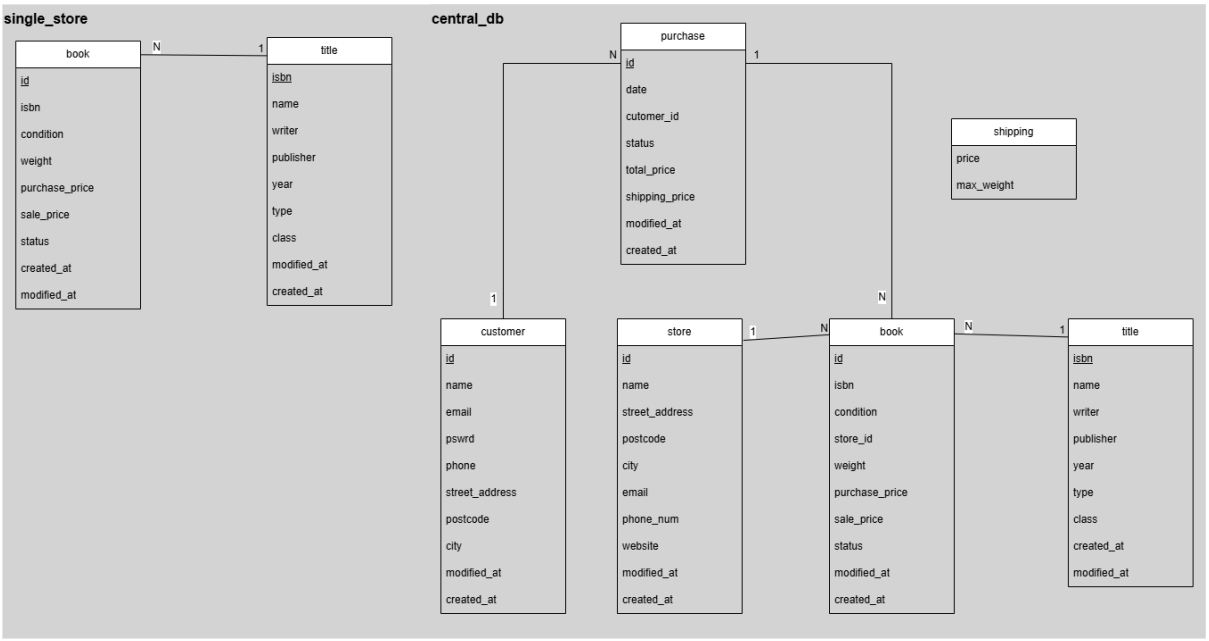
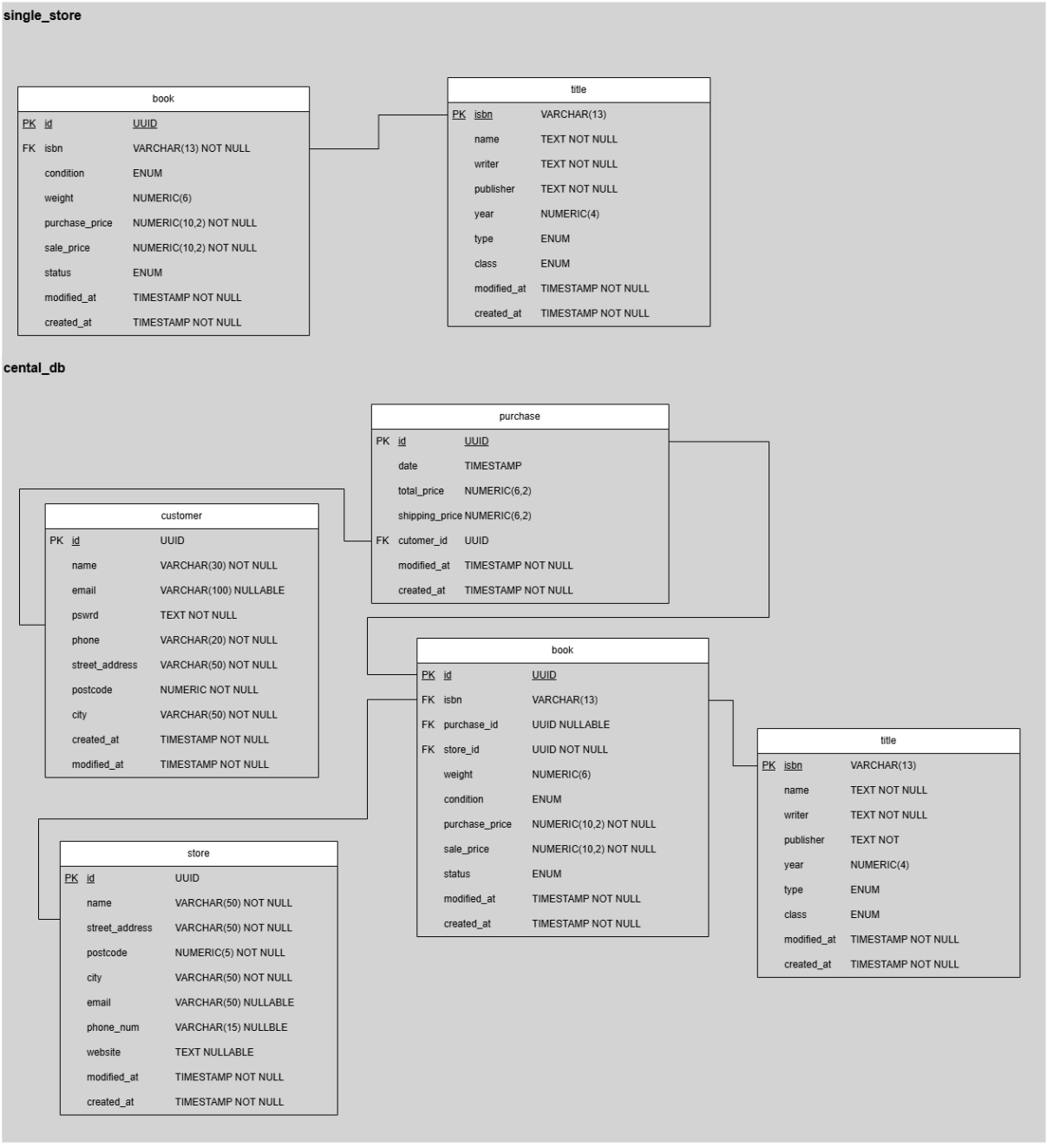
Ryhmä 11 Toivo Knuutinen, Sanni Välimaa, Markus Mattila

Tietokannan UML-kaaviot



Tietokantakaavioiden graafinen kuvaus

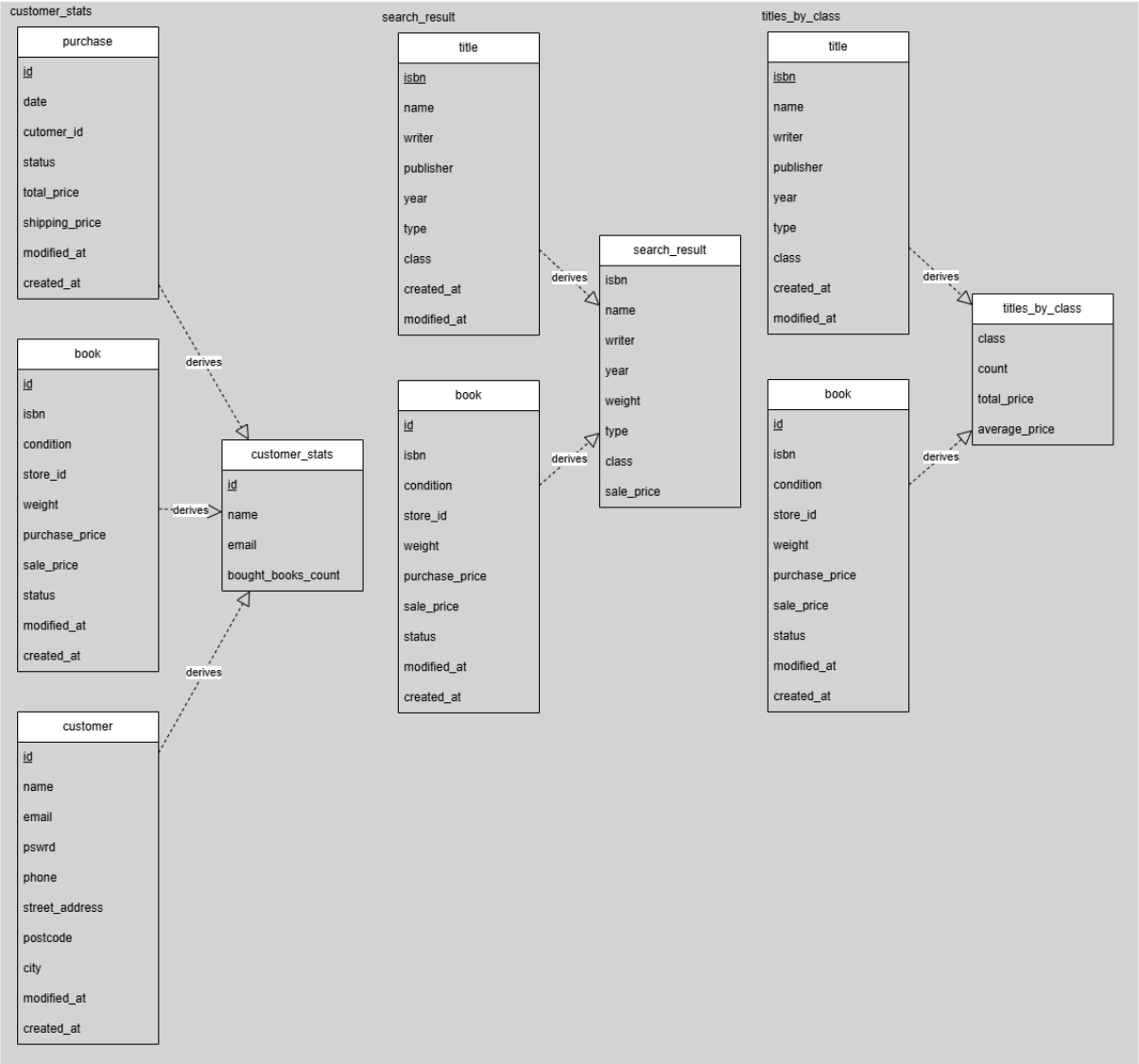


Toteutus ei vaadi välitauluja, vaan suhteet käsitellään suoraan viiteavaimilla. Tämä teki UML-kaavion muunnoksesta tietokantakaavioksi suoraviivaista.

- Avainkentät: UUID-pohjaiset.
- Tekstikentät: VARCHAR ja TEXT, pituus rajoitettu tarpeen mukaan.
- Numerokentät: Rahasummat NUMERIC(6,2), painoarvot NUMERIC(6), postinumerot NUMERIC(5), vuodet NUMERIC(4).
- ENUM-tyypit: Määrittelevät kirjan kunnon, myyntistatuksen, tyypin ja luokan.
- NULL-rajoitukset: NULL-arvoja ei sallita joitain poikkeuksia lukuunottamatta, erityisesti useimmat purchase-taulun kentät, jotka ovat NULL ennen kuin tilaus on vahvistettu.

Toteutus: PostgreSQL, Node.js, uuid-ossp-laajennus.

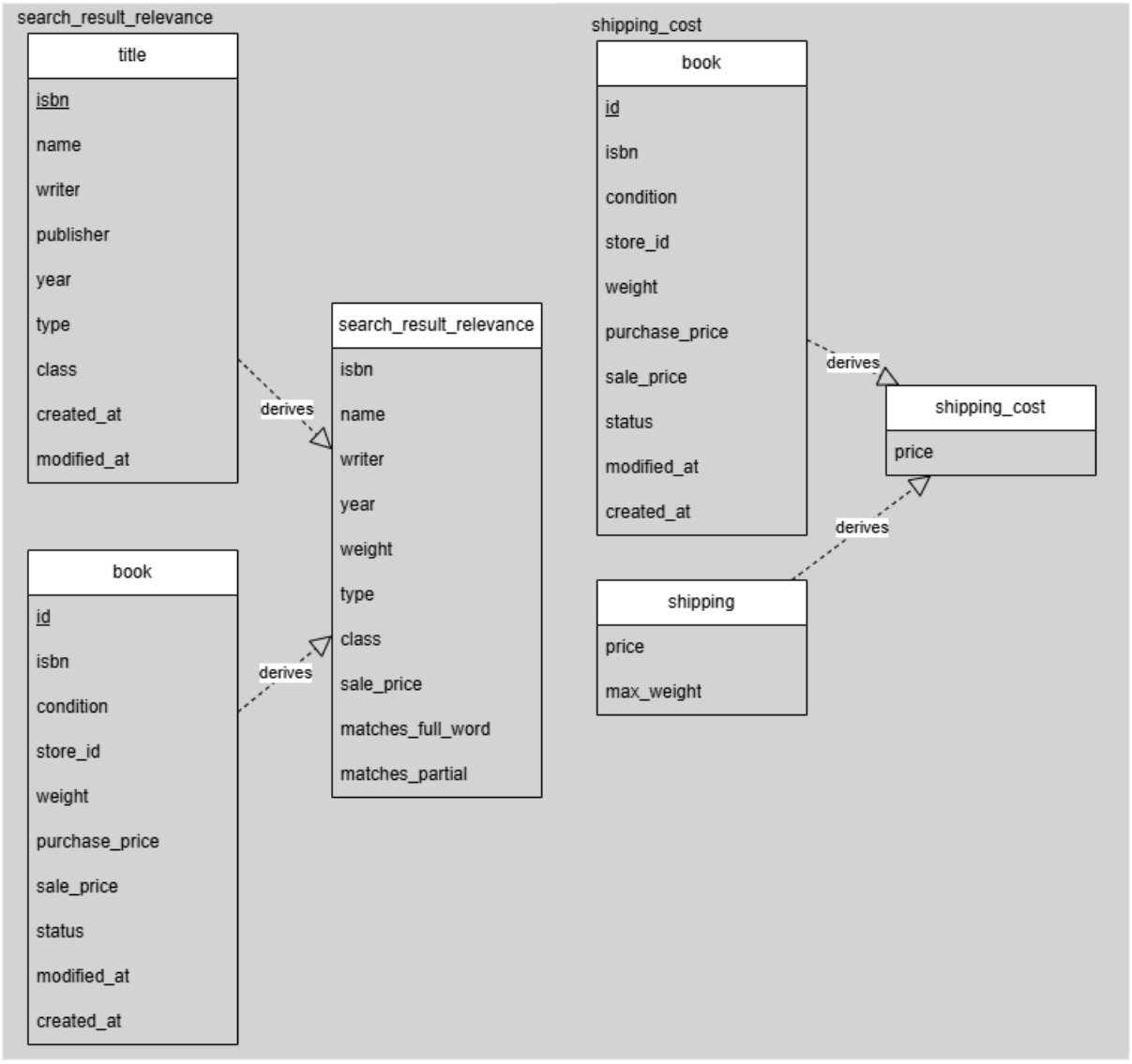
Kuvaus ja kaaviot näkymistä sekä johdettavasta tiedosta



search_result: Haun tulokset haussa, joka voi kohdistua teoksen nimeen, teoksen tekijään, teoksen tyyppiin sekä teoksen luokkaan. Book-taulu yhdistetään title-tauluun isbn perusteella ja valitaan kuvatut sarakkeet.

titles_by_class: Myynnissä olevat teokset ryhmiteltynä luokan mukaan. Ensin book-taulu yhdistetään title-tauluun isbn perusteella ja sitten tulokset ryhmitellään luokan mukaan. Näkymästä näkyy kunkin teosluokan kokonaismyyntihinta ja keskihinta.

customer_stats: Asiakkaat ja näiden viime vuonna ostamien teosten lukumäärä. Customer-taulu yhdistetään purchase-tauluun customer_id perusteella ja purchase-taulu yhdistetään book-tauluun purchase_id perusteella. Ostot rajataan viimeiseen kalenterivuoteen purchase-taulun date-kentän ja nykyhetken perusteella. Valitaan kuvatut sarakkeet ja lasketaan bought_books_count ostettujen kirjojen määrästä.



search_result_by_relevance: Hakutulos teoksen nimeen kohdistuvasta haussa, jossa hakutulokset järjestellään relevanssin perusteella. Sarakkeet valitaan tauluista samalla tavalla kuin useisiin tietoihin kohdistuvassa haussa search_result. Taulussa ”matches_full_word” ilmoitetaan, kuinka moni hakusana esiintyy teoksen nimessä ja ”matches_partial” ilmoitetaan osittaiset osumat. Näiden perusteella voidaan järjestää hakutulokset, ensisijaisesti laskevaan järjestykseen matches_full_word perusteella ja toissijaisesti laskevaan järjestykseen matches_partial perusteella.

shipping: Tilauksen kokonaispaino lasketaan yhteenlaskemalla kaikkien book-taulussa olevien kirjojen weight-arvot. Oikea toimituskulu valitaan shipping-taulusta etsimällä ensimmäinen rivi, jossa max_weight on suurempi tai yhtä suuri kuin tilauksen paino.

Tarvittavien tapahtumien kuvaukset

Asiakas rekisteröityy järjestelmään

- kirjoita asiakkaan antamat tiedot keksutietokannan *customer* relaatioon

Lisätään yksittäisen teoksen (jota ei ole aikaisemmin tallennettu) tiedot divarin D1 tietokantaan ja keskustietokantaan

- kirjoita nimikkeen tarvittavat tiedot divarin D1 ja keskustietokannan *title* sekä *book* relaatioon

Lisätään yksittäinen teos divarille D2 tietokantaan (siis keskusdivarin tietokantaan), jonka teostiedot jo löytyvät tietokannasta. (käyttäjäroolina divarin D2 ylläpitäjä)

- lue teoksen *isbn* keskusdivarin *title* relaatiosta
- kirjoita teoksen tiedot (ml. *isbn*) ja divarin D2 *store_id* keskustietokannan *book* relaatioon

Asiakas tekee yksittäisen kirjan tilauksen (eli tilaus on maksettu onnistuneesti)

- lue keskusdivarin relaatiosta *customer* asiakkaan tiedot (ainakin *id* ja osoitetiedot)
- lue asiakkaan valitseman kirjan tiedot keskusdivarin relaatiosta *book* (ainakin *id*, *store_id*, *weight*, *sale_price* ja *status*)
- kirjoita keskusdivarin ja mahdollisesti divarin D1, jossa kirja on, *book* relaatioon valitun kirjan *status* sarakkeeseen ’reserved’
- lue *shipping* relaatiosta kirjan painon mukainen kuljetusmaksu
- kirjoita keskustietokannan *purchase* relaatioon ostotapahtuman tiedot kun asiakas on maksanut
- kirjoita keskustietokannan ja mahdollisesti sen divarin, mistä kirja tilattiin *book* relaatioihin kyseisen kirjan *status* sarakkeeseen ’sold’ ja *purchase_id* sarakkeeseen luodun ostotapahtuman *id*

Asiakas tekee tilauksen, jonka paino ylittää 2000 grammaa (käyttäjäroolina asiakas). Lähetys joudutaan siis jakamaan useaan erään. Käytännön yksi tilaus voi kohdistua usean divarin teoksiin. Tätä ei kuitenkaan työssä huomioida vaan lasketaan postituskulut siten, että ne kohdistuisivat saman divarin teoksiin

- lue keskusdivarin relaatiosta *customer* asiakkaan tiedot (ainakin *id* ja osoitetiedot)
- lue asiakkaan valitsemien kirjojen tiedot keskusdivarin relaatiosta *book* (ainakin *id*, *store_id*, *weight*, *sale_price* ja *status*)
- kirjoita keskusdivarin ja divarin D1 *book* relaatioon valittujen kirjojen *status* sarakkeeseen ’reserved’
- lue *shipping* relaatiosta kirjojen painojen mukaiset hinnat (tilaus jakautuu kahtia ja painolaskuri nollataan kun tilaus ylittää suurimman *shipping* relaation maksimipainon)
- kirjoita keskusdivarin relaatioon *purchase* uusi rivi jokaisesta muodostuneesta tilauksesta kun asiakas on maksanut tilaukset
- asiakkaan maksettua tilauksen kirjoita keskusdivarin ja divarin D1 *book* relaatioon valittujen kirjojen *status* sarakkeeseen ’sold’ ja niiden *purchase_id* sarakkeeseen niille kuuluvan ostotapahtuman *id*

Toteuta triggeri, joka päivittää keskusdivarin automaattisesti, kun divarin omaan tietokantaan tuodaan uusi myyntikappale. Oletetaan, että teoksen yleiset tiedot on talletettu ennen lisäystä molempiin tietokantoihin. Toteuta tätä varten kolmannen divarin D3 tietokanta, joka voi rakenteellisesti noudattaa divarin D1 kantaa

- triggeri asetetaan kuuntelemaan divarin D1 *book* relaatiota
- triggeri huomaa muutoksen relaatiossa
- lue D1 ja D3 taulujen *book* relaatiot
- kirjoita keskusdivarin *book* relaatioon ne rivit, jotka löytyivät D1 taulusta mutta ei D3 taulusta

Oletetaan, että Divarin tietokantaan on lisätty kirjoja, joita ei löydy keskustietokannasta. (Simuloi tilannetta lisäämällä D1:n pari uutta teosta.) Päivitä keskusdivarin tiedot. Tässä siis siirretään divarin D1 tietokannasta (kaavioista) uusien teosten tiedot keskustietokantaan. Tämän voi toteuttaa vertaamalla tietokannoissa olevia tietoja tai esimerkiksi pitämällä yllä tietoa, että minkäkin teoksien tiedot ovat muuttuneet viimeisen päivityksen jälkeen

- lue D1 *title* ja *book* relaatioista ne rivit, joita ei löydy keskustietokannasta
- kirjoita löydetyt rivit keskustietokannan *book* ja *title* relaatioihin

Divari D4 haluaa liittyä keskusdivariin. Heidän datansa on XML-muodossa, jonka rakenne noudattaa liitteessä 2 annettua DTD:tä. Toteuta toiminto, joka konfiguroi ja siirtää datan keskusdivarin tietokantaan. Tee esimerkkiaineistoon vähintään kaksi teosta, joissa kaksi nidettä. Muita kuin kuvattuja tietoja ei tarvitse huomioida.

- kirjoita keskusdivarin *store* relaatioon uuden divarin tiedot
- luo uusi D4 tietokanta
- kirjoita divarista D4 löytyvien nimikkeiden ja kirjojen tiedot keskusdivarin *book* ja *title* relaatioihin ja merkitse niiden *store_id* sarakkeeseen divarin D4 *id*
- kirjoita divarista D4 löytyvien nimikkeiden ja kirjojen tiedot uuden D4 tietokannan *book* ja *title* relaatioihin

Tietokantojen luontilauseet

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- book condition
CREATE TYPE condition_enum AS ENUM (
  'NEW',
  'GOOD',
  'FAIR',
  'POOR'
);

-- sale status
CREATE TYPE status_enum AS ENUM (
  'AVAILABLE',
  'RESERVED',
  'SOLD'
);

-- book type
CREATE TYPE type_enum AS ENUM (
  'HARDCOVER',
  'PAPERBACK',
  'CARTOON'
  'OTHER'
);

-- book genre / class
CREATE TYPE class_enum AS ENUM (
  'FICTION',
  'NONFICTION',
  'COMIC',
  'OTHER'
);


-- central_db

CREATE TABLE IF NOT EXISTS customer (
  id              UUID              PRIMARY KEY DEFAULT uuid_generate_v4(),
  name            VARCHAR(30)       NOT NULL,
  email           VARCHAR(100),
  passwrd         TEXT              NOT NULL,
  phone           VARCHAR(20)       NOT NULL,
  street_address  VARCHAR(50)       NOT NULL,
  postcode        NUMERIC           NOT NULL,
  city            VARCHAR(50)       NOT NULL,
  created_at      TIMESTAMP         NOT NULL,
  modified_at     TIMESTAMP         NOT NULL
);

CREATE TABLE IF NOT EXISTS store (
  id              UUID              PRIMARY KEY DEFAULT uuid_generate_v4(),
  name            VARCHAR(50)       NOT NULL,
  street_address  VARCHAR(50)       NOT NULL,
  postcode        NUMERIC(5)        NOT NULL,
  city            VARCHAR(50)       NOT NULL,
  email           VARCHAR(50),
  phone_num       VARCHAR(15),
  website         TEXT,
  created_at      TIMESTAMP         NOT NULL,
```

```
  modified_at     TIMESTAMP         NOT NULL
);

CREATE TABLE IF NOT EXISTS purchase (
  id                UUID            PRIMARY KEY DEFAULT uuid_generate_v4(),
  date              TIMESTAMP       NOT NULL,
  total_price       NUMERIC(6,2)    NOT NULL,
  shipping_price    NUMERIC(6,2),
  customer_id       UUID            NOT NULL,
  created_at        TIMESTAMP       NOT NULL,
  modified_at       TIMESTAMP       NOT NULL

  CONSTRAINT fk_purchase_customer
    FOREIGN KEY (customer_id)
    REFERENCES customer (id)
    ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS title (
  isbn          VARCHAR(13)     PRIMARY KEY,
  name          TEXT            NOT NULL,
  writer        TEXT            NOT NULL,
  publisher     TEXT            NOT NULL,
  year          NUMERIC(4),
  type          type_enum       NOT NULL,
  class         class_enum      NOT NULL,
  created_at    TIMESTAMP       NOT NULL,
  modified_at   TIMESTAMP       NOT NULL
);

CREATE TABLE IF NOT EXISTS book (
  id              UUID            PRIMARY KEY DEFAULT uuid_generate_v4(),
  isbn            VARCHAR(13)     NOT NULL,
  purchase_id     UUID,
  store_id        UUID            NOT NULL,
  weight          NUMERIC(6)      NOT NULL,
  condition       condition_enum  NOT NULL,
  purchase_price  NUMERIC(10,2)   NOT NULL,
  sale_price      NUMERIC(10,2)   NOT NULL,
  status          status_enum     NOT NULL,
  created_at      TIMESTAMP       NOT NULL,
  modified_at     TIMESTAMP       NOT NULL,

  CONSTRAINT fk_book_isbn
    FOREIGN KEY (isbn)
    REFERENCES title (isbn)
    ON DELETE RESTRICT,

  CONSTRAINT fk_book_purchase
    FOREIGN KEY (purchase_id)
    REFERENCES purchase (id)
    ON DELETE SET NULL,

  CONSTRAINT fk_book_store
    FOREIGN KEY (store_id)
    REFERENCES store (id)
    ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS shipping (
  max_weight   NUMERIC(6)      PRIMARY KEY,
```

```
  price         NUMERIC(4,2)     NOT NULL
);


-- single_db

CREATE TABLE IF NOT EXISTS title (
  isbn          VARCHAR(13)      PRIMARY KEY,
  name          TEXT             NOT NULL,
  writer        TEXT             NOT NULL,
  publisher     TEXT             NOT NULL,
  year          NUMERIC(4),
  type          type_enum        NOT NULL,
  class         class_enum       NOT NULL,
  created_at    TIMESTAMP        NOT NULL,
  modified_at   TIMESTAMP        NOT NULL
);

CREATE TABLE IF NOT EXISTS book (
  id              UUID             PRIMARY KEY DEFAULT uuid_generate_v4(),
  isbn            VARCHAR(13)      NOT NULL,
  condition       condition_enum   NOT NULL,
  weight          NUMERIC(6)       NOT NULL,
  purchase_price  NUMERIC(10,2)    NOT NULL,
  sale_price      NUMERIC(10,2)    NOT NULL,
  status          status_enum      NOT NULL,
  created_at      TIMESTAMP        NOT NULL,
  modified_at     TIMESTAMP        NOT NULL,

  CONSTRAINT fk_book_isbn
    FOREIGN KEY (isbn)
    REFERENCES title (isbn)
    ON DELETE RESTRICT
);
```

Raporttien luontilauseita näkyminä

```
CREATE TYPE book_condition AS ENUM ('new', 'used', 'damaged');
CREATE TYPE book_type AS ENUM ('novel', 'picture_book', 'comic', 'guide',
'nonfiction');
CREATE TYPE book_class AS ENUM ('romance', 'adventure', 'humor', 'history',
'detective', 'manual');

CREATE VIEW search_result AS
SELECT
    b.isbn,
    t.name,
    t.writer,
    t.year,
    b.weight,
    t.type,
    t.class,
    b.sale_price
FROM book b
JOIN title t ON b.isbn = t.isbn;


CREATE VIEW titles_by_class AS
SELECT
    t.class,
    COUNT(b.id) AS count,
    SUM(b.sale_price) AS total_price,
    AVG(b.sale_price) AS average_price
FROM book b
JOIN title t ON b.isbn = t.isbn
GROUP BY t.class;

CREATE VIEW customer_stats AS
SELECT
    c.id,
    c.name,
    c.email,
    COUNT(p.id) AS bought_books_count
FROM customer c
LEFT JOIN purchase p ON c.id = p.customer_id
GROUP BY c.id, c.name, c.email;


CREATE VIEW search_result_relevance AS
SELECT
    b.isbn,
    t.name,
    t.writer,
    t.year,
    b.weight,
    t.type,
    t.class,
    b.sale_price,
    -- Lasketaan kuinka monta kertaa hakusana esiintyy nimessä
    LENGTH(LOWER(t.name)) - LENGTH(REPLACE(LOWER(t.name), 'hakusana', ''))
AS matches_full_word,
    -- Osittainen täsmäys
    CASE
        WHEN t.name ILIKE '%hakusana%' THEN 1
        ELSE 0
    END AS matches_partial
```

```
FROM book b
JOIN title t ON b.isbn = t.isbn;


CREATE VIEW shipping_cost AS
SELECT
    b.id AS book_id,
    b.isbn,
    b.weight,
    s.price AS shipping_price
FROM book b
JOIN shipping s ON b.weight <= s.max_weight
ORDER BY s.max_weight ASC
LIMIT 1;
```