Bazy danych

1. Typy danych w bazie danych PostgreSQL

Name	Size	Range	Comment
boolean	1 bajt	false / true	przechowuje true lub fasle. Może być null
smallint	2 bajty	-32768 do 32767	dla małych zakresów
integer	4 bajty	-2147483648 do 2147483647	najczęściej stosowany
bigint	8 bajty	-9223372036854775 808 do 9223372036854775 807	int o dużym zakresie
real	4 bajty	dokładność 6 cyfr po dziesięnych	the data types real and double precision are inexact, variable-precision numeric types - czyli nie przechowuje dokładnej wartości
numeric(p,s)	zmienna	Do 131072 cyfr przed przecinkiem oraz do 16383 cyfr po przecinku	p - łączna ilość cyfr s - wartość ilość miejsc po przecinku np. (5,2) - oznacza że 3 przed przecinkiem, 2 po przecinku np. 320,12

Typy tekstowe

char(n)	ciąg znaków o długości n. W przypadku niewykorzystania pełnego zakresu reszta zostanie wypełniona spacjami
varchar(n)	tak samo jak char, lecz nie dopełnia spacjami
text	tekst o zmiennej długości

Typy czasowe:

DATE	data
TIME	czas
TIMESTAMP	data i czas
TIMESTAMP WITH TIME ZONE	data i czas z uwzględnieniem strefy

Tworzenie tabeli - DDL - Data Definition Language

Do stworznenia bazy danych musimy wykorzystać zapytanie DDL, która tworzy strukturę tabeli, określając jakie pola mają się w nim znajdować, ich rodzaj oraz inne parametry.

```
CREATE TABLE XYZ (
ID INT PRIMARY KEY NOT NULL,
NAME VARCHAR(20) NOT NULL
);
```

W przypadku tabel zawsze ustawiamy na nich klucz główny (primary key)

Usuwanie tabeli - DROP TABLE nazwaTabeli.

Edytowanie struktury tabeli - ALTER TABLE X ALTER COLUMN

Do zmiany struktury tabelli używamy komendę Alter table np.

```
ALTER TABLE employees
ALTER COLUMN surname DROP NOT NULL;
```

Wkładanie danych do tabeli

Wkładanie danych do tabeli wykonujemy poprzez słowo kluczowe INSERT INTO np.

```
INSERT INTO employees(id,anme,surname,age) values(1,"Michal","Bi
```

Odczyt danych z tabeli - Select

Do odczytywania danych używamy operacji Select np.

```
SELECT * FROM employees;
```

Odczytywanie wyłącznie poszczególnych kolumn:

```
SELECT
    id,
    name,
    surname
FROM employees;
```

Aliasy

Mechnizm aliasów służy do tego, aby kolumy które są wyślwietlane tylko w widoku miały inną nazwę, którą podaliśmy np.

```
Seelect
    id AS my_id,
    name AS my_name
    surname AS my_surname
FROM emloyees;
```

Where

Aby narzucić warunek do wyślwietlanego zapytania możemy użyć klauzuli WHERE, np.

```
SELECT *
    FROM employees
    WHEE name = 'Michal';
```

Łączenie warunnków - operatory loginczne OR/AND

Domemy używać operatorów do łączenia warunków np.

```
SELECT *
    FROM employees
    WHERE name ='Michal'
    AND surname='Bialek';
```

Operatory

W zapytaniach SQL możemy rónież używać operatory matematyczne takie jak:

dodwanie, odejmowanie, mnożenie, dzielenie, modulo.

Dostępne są również operatory porównania takie jak:

1. równość (=)
2. nierówność (!=)
3. różność (<>)
4. mniejszość (<)
5. większość (>)
6. mniejszość lub róność (<=)
7. większość lub równość (>=).

Dostępne są operatory logiczne:

1. OR
2. AND
3. IN - sprawdza czy wartość w jest równa jednej z podanej wartości

   ```
   SELECT * FROM employees WHERE name IN ('Andrzej','Michal');
   ```

4. LIKE - operator podobny do String.contains(). Znak % oznacza dowolny ciąg znaku.

   ```
   SELECT * FROM employees WHERE name LIKE '%Mi';
   ```

5. BETWEEN - sprawdz czy wartość znajduje się w przedziale

```
SELECT * FROM employees WHERE age BETWEEN 20 AND 50;
```

6. NOT
7. IS NULL

Sortowanie zwracanego wyniku

```
SELECTY *
FROM employees
ORDER BY age DESC;
```

Możemy również wykonać sortowanie po kilku kolumnach do przecinku podając kolejną kolumne i tryb sortowania

```
ORDER BY age DESC, salay desc;
```

Ograniczanie zwracanego wyniku - LIMIT

Jeżeli chcemy ograniczyć ilość zwracanyc wyników, możemy użyć słowa kluczowego limit np.

```
SELECT *
FROM employees
ORDER BY age ASC
LIMIT 3;
```

Unikalne wartości - DISTINCT

Słowo kluczowe DISTINCT służy do zwracania unikatowych wyników np. imiona w firmie

```
SELECT DISTINCT name FROM employees;
```

Funkcje agregujące i grupowanie

Funkcje agrekujące służą do zamiany elemntów w zdfeinowaną wartość liczbową. Przykładowymi funkcjami agregującymi są:

1. COUNT
2. SUM
3. AVG
4. MIN
5. MAX

Przykładowo, chcemy podliczyć ilość występujących unikalnych imion, które zaczynają się na ‘M’ :

```
SELECT COUNT(DISTINCT name)
FROM employees
WHERE name LIKE 'M%';
```

GROUP BY

Funkcja ta działa podobnie jak grupowanie w stremach w javie. Jeżeli chcemy uzyskać strukturę:

klucz:lista_wartości. np.

```
SELECT age, COUNT(age)
FROM emlpoyees
GROUP BY age;
```

Aktualizowanie bazy danych - UPDATE, SET, WHERE

Przykład:

```
UPDATE employees
SET salary = 10000
WHERE name='Ania' AND surname='Bogata';
```

Usuwanie rekordu w bazie danych DELTE, FROM, . additonal (WHERE)

Do usuwania rekordów w bazie danych służy operacja DELETE ex.

```
DELETE
FROM EMPLOYEES
WHERE ID = 70;
```

Relacyjność baz danych

1. Na czym polega relacyjnośc baz danych i jak ją stosować ?

   Jeżeli chemy przechowywać jakieś dane, to informację, które są/mogłyby być wykorzysytwane w kilku miejscach, powinno się przechowywać w osobnych tabelach, ponieważ każda tabela podawinna odpowiadać tylko za jedną rzecz. Następnie tabele te łączymy powiązaniami, aby uzyskać wymagane dane. Wiązanie odbywa się przy pomocy kluczy.

2. Wiązanie - Do wykonania wiązania używamy zapisku:

```
//Example database:
CREATE TABLE ADDRESSES(
  ID INT NOT NULL,
    ...
  PRIMARY KEY (ID)
);
CREATE TABLE EMPLOYEES(
  ID INT NOT NULL,
    ...
```

```
  ADDRESS_ID INT NOT NULL,
  PRIMARY KEY (ID),

//Zapis powiązania
      CONSTRAINT fk_address
          FOREIGN KEY (ADDRESS_ID)
              REFERENCES ADDRESSES (ID)
);
```

1. Wpisujemy ograniczenie (constraint) fk_adress
2. Klucz obcy (foreign key) odwołujemy się do pola w naszej tabeli, która będzie referenjcą do innej tabeli.
3. Referencja (referenes) - wskazujemy, gdzie powinien klucz obcy się odności (referować)

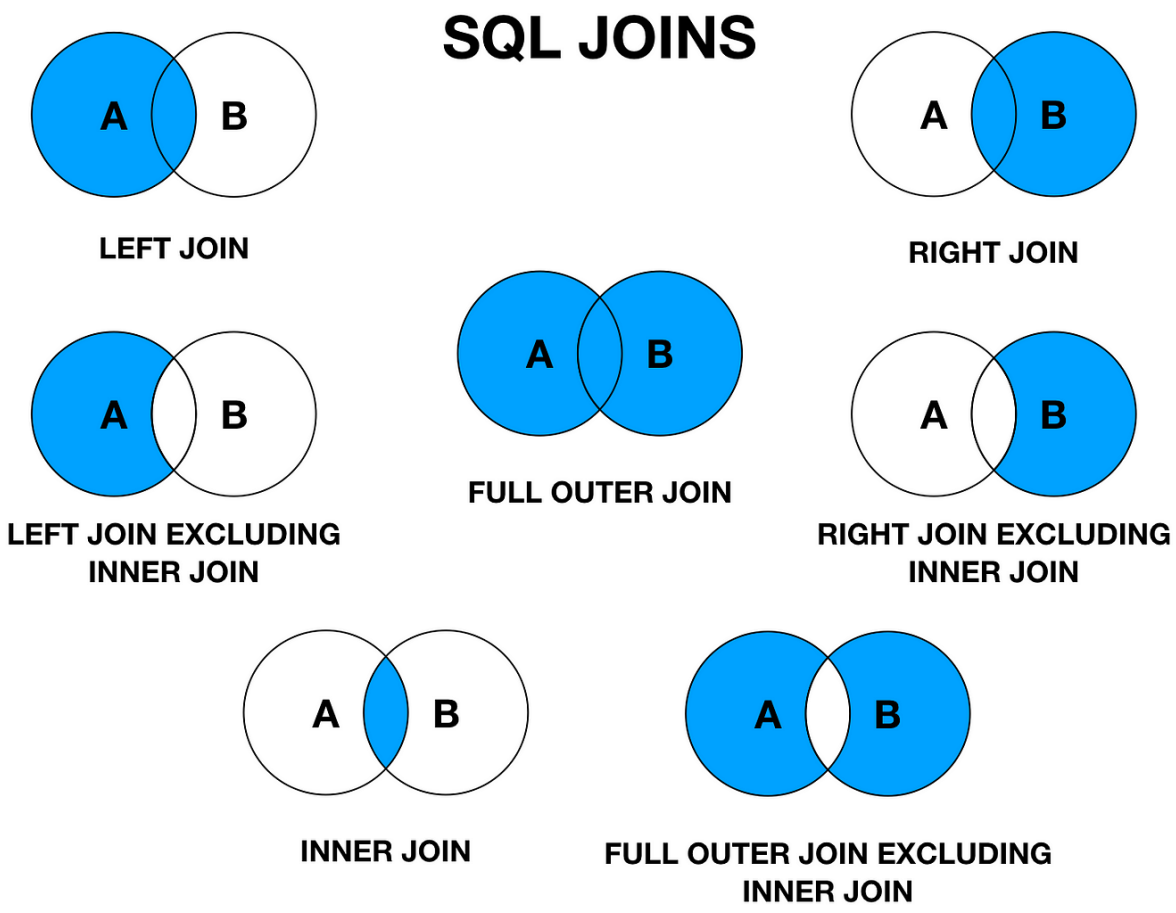
Uwagi

Jeżeli nasz klucz obcy jest not null, to nie można włożyć rekordudo tablicy, bez wcześniejszego utworzenia rekordu w tabeli do której sie odwołujemy.

Klucze obce wymuszają na nas stosowanie pewnej kolenności dodwania i usuwania elementów z bazy danych.

Joiny

Rozróżniamy kilka rodzajów joinów:



Do czego służą joiny ?

Joiny służą do łączenia tabel przy wykorzystaniu kluczy obcych.

Uwaga - przy joinowaniu mamy id tabeli pierwszej, oraz następnych tabel, dlatego odnosząc się do id musimy określić, do id z jakiej tabeli się odwołujemy. np. poprzez employees.id - przykład:

```
SELECT emp.id, name, surname, age, salary, city, street // Jakie
    FROM employees AS emp  // określamy z jakiej tablicy
        INNER JOIN adress adr ON emp.adress_id = adr.id // jaką
        // +on + klucz obcy z tabeli = id z tabeli
```

Podstawowe Joiny

1. Inner join (inaczej zwykły join / przecięcie zbiorów) - bierzemy te, gdzie jest dokładne dopasowanie do siebie, czyli gdzie rekord z lewej odpowiada prawemi i vice versa.
2. Full join - zwracamy rekordy z obu tabel, nawet jeżeli ani lewa, ani prawa strona nie ma dowiązania do siebie.
3. Left join - zwrócimy wszystkie elementy z lewej strony, nwaet jeżeli nie mają dowiązania z prawej, a z prawej wyświetlimy tylko te, co mają dowiązania do lewej.
4. Right join - odwrotna sytuacja do left joina.

Przykłady

1. Inner Join (przecięcie)

```
SELECT id,name,seurname,salary,city,postal_code,street
    FROM employees as emp
        INNER JOIN adress adr on emp.adress_id = adr.id;
```

2. Full Join (wszystko)

```
SELECT id,name,seurname,salary,city,postal_code,street
    FROM employees as emp
        FULL JOIN adress adr on emp.adress_id = adr.id;
```

3. Left

```
SELECT id,name,seurname,salary,city,postal_code,street
    FROM employees as emp
        LEFT JOIN adress adr on emp.adress_id = adr.id;
```

4. Right

```
SELECT id,name,seurname,salary,city,postal_code,street
    FROM employees as emp
```

```
        RIGHT JOIN adress adr on emp.adress_id = adr.id;
```

Resources: nieinformatyk

1. DBMS - Database Management System, czyli system zarządzania bazami danych, gdzie oprócz przechowywania, dodawania oraz wyszukiwania elementów w bazie, posiada również funkcjonalności optymalizacyjne, tworzenia kopii zapasowych itp.
2. Rodzaje baz danych - główne odzielenie polega na bazy relacyjne SQL, oraz nierelacyjne bazy danych noSQL. Bazy danych noSQL dzielimy na różne rodzaje np. kary key-value, column-familiy, graph, document. każda baza danych ma swojego dostawcę (providera) i w przypadku relacyjnych ba danych pogą to być: PostgreSQL, MySql, MSsql server, Oracle database.każdy provider ma swoją implementację, która różni się swoją specyfikacją. Natomias dla noSQL są to mongoDB, amazon dybamoDB, redis itp.
3. Co to jest SQL (Structural Query Language) - jeżyk wykorzystywany do operacji na tabelach relacyjnych.

Elementy baz danych

1. Serwer bazy danych - jest to urządzenie, na którym funkcjonuje baza danych. Jeżeli jest to produkcyjna baza danych, to serwer który powinien być dostępny 24/7 i serwować reści dla klientów. Rozróaniamy podkategorie jak rozproszona baza danych, gdzie kilka urządzeń współpracuje ze sobą tworząc logiczną całość, bazy danych w chmurze, albo rozwiązania hybrydowe.
2. Oprogramowanie bazy danych - program zarządzające plikami bazodanowymi (DBMS).Są one wersjonowane.
3. Instancja bazy danych - Na instancje składa się pamęć RAM oraz ROM. Dane przechowywane w pamięci trwałej są przenoszone do pamięci wenętrznej. Za przenoszenie odpowiadają procesy pracujące w tle. Instancja DB jest przypadna, jeżeli chcemy wiedzieć o szczegółach działnia bazy danych.

Czym kierować się przy wyborze implementacji/drivera bazo db.

1. Przy wyborze konkretnej implementacji najważniejsze są kryterium funkcjonalności, możliwość łączenia danej implementacji z innymi narzędziami, finanse, znajomość przez pracowników danej bazy danych.

Co to jest normalizacja baz danych

1. Normalizacja jest to podział dużych table na mniejsze. Robimy to ze względu to, że brak nomalizacji może powodować błędy :
    a. anomalia wstawiania- przy dużej tabeli nie idzie wstawić np. nowego działu firmy, ponieważ tabela wymaga również podania innych not null danych
    b. anomalia usuwania - usuwając 1 rekord możemy doprowadzić do sytuacji, gdzie zanika kompletnie informacja o innej tabeli, np. jeżeli mamy tylko 1 informatyka i go usuniemy, to usuniemy wszystkie informacje na temat istnienia działu IT w firmie.
    c. anomalia modyfikacji, anomalia czytania, które mogą doprowadzić do niespójności danych.

W przypadku normalizacji ważna jest wiedza jakie są powiązania. np. co jest do czego powiązane:

Postać nie normalna:

IMIE	NAZWISKO	PESEL	ADRES	DZIAŁ	KIEROWNIK DZIAŁU	PENSJA
Magda	Kowalska	78052000653	ul. Kwiatowa 13/16 02-200 Wieluń	Sprzedaż	Wojciechowski Jan	4800
Jakub	Nowak	89122431862	ul. Aksamitna 115 17-682 Łuków	Sprzedaż	Wojciechowski Jan	5500
Paweł	Iksiński	98090500695	ul. Wojskowa 1/15 78-963 Katowice	IT	Renata Kadłubek	6000

1. Pierwsza postać normalna:

ID	IMIE	NAZWISKO	PESEL	ULICA	NUMER BUDYNKU	NUMER MIESZKANIA	KOD_POCZTOWY	MIEJSCOWOŚĆ	DZIAŁ	KIEROWNIK DZIAŁU	PENSJA
1	Magda	Kowalska	78052000653	Kwiatowa	13	16	02-200	Wieluń	Sprzedaż	Wojciechowski Jan	4800
2	Jakub	Nowak	89122431862	Aksamitna	115		17-682	Łuków	Sprzedaż	Wojciechowski Jan	5500
3	Paweł	Iksiński	98090500695	Wojskowa	1	15	78-963	Katowice	IT	Renata Kadłubek	6000

1. Tabela posiada ID

2. Wszystkie kolumny są jedno wartościowe - np. adres nie jest przechowywany pod jednom kolumną, lecz jest rozdzielony na ulice,numer doku, mieszkania, miejscowość itp.
3. Druga postać normalna

ID_PRACOWNIKA	IMIE	NAZWISKO	PESEL	ULICA	NUMER BUDYNKU	NUMER MIESZKANIA	KOD_POCZTOWY	MIEJSCOWOŚĆ	PENSJA	ID_DZIAŁU
1	Magda	Kowalska	78052000653	Kwiatowa	13	16	02-200	Wieluń	4800	1
2	Jakub	Nowak	89122431862	Aksamitna	115		17-682	Łuków	5500	1
3	Paweł	Iksiński	98090500695	Wojskowa	1	15	78-963	Katowice	6000	2

ID_DZIAŁU	NAZWA_DZIAŁU	KIEROWNIK DZIAŁU
1	Sprzedaż	Wojciechowski Jan
2	IT	Renata Kadłubek

    1. Wszystkie kolumny nie kluczowe zależa od klucza głównego
4. Trzecia postać normalna

Tranzakcja

Tranzakcyjność jest to funkcjonalność, któa jest wymagana do realizacji potrzeb biznesowych aplikacji. Uwzględnia ona zasadę kolejkowania/dostępności do danych. Poprawnie zaimplementowana tranzakcyjność powinna zachowywać zasady ACID, czyli Atomicity, Consistency, Isolation, Durability. W zależności od stopnia izolacji tranzakcji możemy spodziewać się różnych negatywnych efektów współbierzności tranzakcji. Są to np. dirty read, phantom read, non-repeatable read.

Relacyjność bazy danych

Relacja to tabela. Co to jest relacyjność baz danych definiuje 12 zasad Edgara Codd'a.

1. Dane są przechowywane są w tabeli.
2. Dostęp do danych uzyskujemy poprzez podanie tabeli, kolumny i klucza głównego
3. Istnieje wartość NULL
4. W bazie danych istnieją metadane

5. **Istnieje technologia zapytań SQL**
6. **Widoki pozwalają na operacjach DML (Data manupulation language)**
7. Oprócz czytania można też modyfikować dane
8. Fizyczna niezależność danych
9. Logiczna niezależność danych
10. **DOstępne więzy spójności (nut null, fk, uniqe)**
11. Niezaloeżność dystrybucyjna
12. Nie idzie ominąć reguł DB operacjami niższego poziomu.

Tworzenie bazy danych

1. Infrastruktura bazy danych (sprze, sieć, OS)
2. Specyfika przetwarzania (baza rozproszona/scentralizowana, OLTP/OLAP ?)
3. Implementacja RDBMS - Wybór silnika bazodanowego, instalacja i konfiguracja
4. przogramy BI, systemy raportujące itp.
5. Architektura i kod (tworzenie aplikacji i optymalizacji)

Modelowanie/Projektowanie bazy danych

Potrzebna jest wiedza na temat danych oraz ich połączeń, w celu doboru odpowiedniej dla nas bazy danych. Projektowanie te składa się z 3 etapów:

1. Tworzenie modelu konceptualnego (analiza biznesu, związku i zależności miedzy danymi, czyli głównie biznes)
2. Tworzenie modelu loginczego (znajomość modelu relacyjnego, diagram związków encji - np. ERD, oraz normalizacja)
3. Tworzenie modelu fizycznego - transformacja na kod SQL, tworzenie kolumn technicznych itp.
4. Narzędzia przyadtne do tworzenia map:
    a. Developer Data Modeler

b. Draw.io
c. StarUML

Funkcje analityczne i funkcje agregujące i porównanie

1. Zastosowanie funkcji analitycznych jest często bardziej optymalne, ponieważ w porównaniu do gunkcji agregującej, dane nie są grupowane

Jak działa WITH w sql → CTE (Common Table Expression)

Jest to bardziej złożone zapytanie SELECT. Klauzula WITH pomaga.

Shopping database project

List of tables:

1. Customer
   a. ID
   b. EMAIL
   c. NAME
   d. SURNAME
   e. BIRTHD_DATE
   f. PHONE_NUMBER
2. Producer
   a. ID
   b. PRODUCER_NAME
   c. BUSINES_IDENTIFIER
   d. COUNTRY

    e. CITY
    f. STREET
    g. POSTAL_CODE
3. Product
    a. ID
    b. PRODUCT_INTERNAL_CODE
    c. EAN_CODE
    d. PRODUCT_PRICE
    e. ADULTS_ONLY
    f. DESCRIPTION
    g. PRODUCER_ID
4. Purchase
    a. ID
    b. CUSTOMER_ID
    c. PRODUCT_ID
    d. QUANTITY
    e. DATE_TIME
5. Opinion
    a. ID
    b. CUSTOMER_ID
    c. PRODUCT_ID
    d. STARS
    e. COMMENT
    f. DATE_TIME

Excercise:

1. Create presenteddatabase and provide mock data (for instance by website" mockaroo.com)
2. Delete every opinion that's under and equal to 3 stars.
3. Show distinct product ean code for products that were bought in last year (2022)
4. Show most bought 5 products - show ean code and quantity of transaction where appears those product's
5. Show all client's that bought products intended for adults
6. Show at what age are person that bought most often products for adults
7. Input 20% discount for all products form XXX company
8. Search person that at least one give 1 star opinion
9. Search company that their products are most selling in our shop
10. Show second most expensive product (we can use OFFSET)
11. Show list of 10 most reviewed products
12. Advanced - Count earnings for every single month in our company. We can use DATE_TRUNC('month', date_time);

Solutions

Database

```
DROP TABLE IF EXISTS customer CASCADE;
DROP TABLE IF EXISTS PRODUCER CASCADE;
DROP TABLE IF EXISTS product CASCADE;
DROP TABLE IF EXISTS purchase CASCADE;
DROP TABLE IF EXISTS opinion CASCADE;


CREATE TABLE customer
(
```

```
    id            INT          NOT NULL PRIMARY KEY,
    email         VARCHAR(255) NOT NULL UNIQUE,
    name          VARCHAR(64)  NOT NULL,
    surname       VARCHAR(64)  NOT NULL,
    date_of_birth DATE         NOT NULL,
    phone_number  varchar(12)  NOT NULL,

    CONSTRAINT phone_number_validation CHECK ( phone_number ~ '^
    CONSTRAINT email_validation CHECK (email ~ '^[A-Za-z0-9._%+-
);

CREATE TABLE producer
(
    id                  INT          NOT NULL PRIMARY KEY,
    producer_name       VARCHAR(128) NOT NULL,
    business_identifier VARCHAR(9)   NOT NULL,
    country             VARCHAR(64)  NOT NULL,
    city                VARCHAR(64)  NOT NULL,
    street              VARCHAR(128) NOT NULL,
    building_number     VARCHAR(6)   NOT NULL,
    postal_code         varchar(6)   NOT NULL,

    CONSTRAINT postal_code_validation CHECK ( postal_code ~ '^[0
    CONSTRAINT business_identifier_validation CHECK ( business_i
);

CREATE TABLE product
(
    id                   INT           NOT NULL PRIMARY KEY,
    product_name         VARCHAR(32)   NOT NULL,
    product_internal_code VARCHAR(9)    NOT NULL,
    ean_code             VARCHAR(13)   NOT NULL,
    product_price        NUMERIC(9, 2) NOT NULL,
    adults_only          BOOLEAN       NOT NULL,
    description          TEXT          NOT NULL,
    producer_id          INT           NOT NULL,
```

```
    CONSTRAINT product_internal_code_validation CHECK ( product_
    CONSTRAINT ean_code_validation CHECK (ean_code ~ '^\d{8}$|^\

    CONSTRAINT fk_producer
        FOREIGN KEY (producer_id)
            REFERENCES producer (id)

);

CREATE TABLE purchase
(
    id          INT NOT NULL PRIMARY KEY,
    customer_id INT NOT NULL,
    product_id  INT NOT NULL,
    quantity    INT NOT NULL,
    date_time   TIMESTAMP WITH TIME ZONE,

    CONSTRAINT fk_customer
        FOREIGN KEY (customer_id)
            REFERENCES customer (id),
    CONSTRAINT fk_product
        FOREIGN KEY (product_id)
            REFERENCES product (id)

);

CREATE TABLE opinion
(
    id          INT      NOT NULL PRIMARY KEY,
    customer_id INT      NOT NULL,
    product_id  INT      NOT NULL,
    stars       SMALLINT NOT NULL,
    comment     VARCHAR  NOT NULL,
    date_time   TIMESTAMP WITH TIME ZONE,
    CONSTRAINT stars_validation CHECK ( stars BETWEEN 1 AND 5),
```

```
    CONSTRAINT fk_customer
        FOREIGN KEY (customer_id)
            REFERENCES customer (id),
    CONSTRAINT fk_product
        FOREIGN KEY (product_id)
            REFERENCES product (id)
);
```

Excercises

```
-- Excercise 1 - Delete every opinion that’s under and equal to
DELETE
FROM opinion
WHERE stars BETWEEN 1 AND 3;
-- Excercise 2 - Show distinct product ean code for products tha

SELECT PRODUCT.ean_code, purchase.date_time
FROM purchase
         INNER JOIN PRODUCT ON PURCHASE.PRODUCT_ID = PRODUCT_ID
WHERE purchase.date_time >= '2023-01-1'
  AND purchase.date_time <= '2023-12-31';
-- Exercise 3 - Show most bought 5 products  - show ean code and

SELECT pr.product_internal_code, count(purchase.id) as quantityO
from purchase
         inner join public.product pr on pr.id = purchase.produc
group by pr.product_internal_code
order by quantityOfTransaction desc
limit 5;

-- Exercise 4 - Show all client’s that bought products intended
SELECT c.name, c.surname, c.phone_number
FROM purchase
         INNER JOIN public.customer c on c.id = purchase.custome
```

```
         INNER JOIN public.product p on p.id = purchase.product_
WHERE p.adults_only = 'true';
;
-- Exercise 5 - Show at what age are person that bought most oft

    SELECT EXTRACT(YEAR FROM age(CURRENT_DATE, c.date_of_birth))
           count(p.id) AS sum_of_bought
    FROM purchase
        INNER JOIN public.product p on p.id = purchase.product_i
    INNER JOIN public.customer c on c.id = purchase.customer_id
    where p.adults_only='true'
    group by c.date_of_birth
    order by sum_of_bought desc;

-- Exercise 6 - Input 20% discount for all products form Twitter
-- Selecting
SELECT product_name, product_price FROM producer
 inner join public.product p on producer.id = p.producer_id
    WHERE producer_name = 'Twitterwire';
-- Updating
    UPDATE product
    SET product_price = product_price*0.8
    FROM producer
        WHERE product.producer_id=producer.id
    AND producer_name = 'Twitterwire';
-- 111, 291,489

-- Exercise 7 - Search person that at least one give 1 star opin
SELECT c.name ,c.surname from opinion
    inner join public.customer c on c.id = opinion.customer_id
        where stars = 1;
-- Exercise 8 - Search company that their products are most sell

SELECT producer_name, count(product_id) as number_of_sold_produc
    FROM purchase
        INNER JOIN public.product p on p.id = purchase.product_i
```

```
        INNER JOIN public.producer p2 on p2.id = p.producer_id
    GROUP BY producer_name
    ORDER BY number_of_sold_products desc;

-- Exercise 9 - Show second most expensive product

    SELECT product_name, product_price
    FROM product
    ORDER BY product_price desc
    OFFSET 1
    LIMIT 1;

-- Exercise 10 - Show list of 10 most reviewed products
    SELECT product_name, count(o.product_id) as number_of_opinio
    FROM product
        inner join public.opinion o on product.id = o.product_id
        GROUP BY product_name
        ORDER BY number_of_opinions DESC;

-- Exercise 11 - Advanced - Count earnings for every single mont

    WITH TMP AS(
        SELECT
            date_trunc('month',pur.date_time) as date_time,
            pur.quantity,
            pr.product_price,
            pur.quantity * pr.product_price AS income_per_produc
        FROM purchase pur
            inner join public.product pr on pur.product_id = pr
        order by date_time, income_per_product
    )
    SELECT TMP.date_time, sum(tmp.income_per_product) as income
        FROM TMP
    GROUP BY TMP.date_time
    ORDER BY income desc;
```