Лабораторна робота 4: Проектування фізичної моделі бази даних

Мета:

Навчити студентів перетворювати логічну модель у фізичну модель бази даних, з подальшою реалізацією цієї моделі у конкретній СУБД.

Завдання:

1. Аналіз ER-діаграми:

– Переглянути ER-діаграму, створену в попередній лабораторній роботі.

2. Визначення типів даних:

– Визначити типи даних для атрибутів кожної сутності, відповідно до вимог СУБД PostgreSQL.

3. Реалізація схеми бази даних у СУБД:

– Створити фізичну схему бази даних у СУБД, написавши SQL-запити для створення таблиць, зв'язків між ними, обмежень, а також для введення початкових даних.

4. Тестування бази даних:

– Виконати кілька SQL-запитів для перевірки коректності функціонування бази даних (наприклад, SELECT, INSERT, UPDATE, DELETE).

Результат:

Студенти повинні подати SQL-скрипти, що відображають створення фізичної моделі бази даних, їх опис, а також звіт з результатами тестування.

Теоретичні відомості до виконання лабораторної роботи

Типи даних

У мові *SQL* підтримуються такі основні типи даних (та багато інших в документації):

***CHAR**[ACTER](n)* – текстовий рядок фіксованої довжини в *n* символів. Максимальне значення n – 254 символи. Константа такого типу задається в апострофах (‘). Причому значення ‘‘ дозволяє включити в структуру і сам апостроф (‘).

VARCHAR – текст структури змінної довжини. Її максимальний розмір залежить від СУБД – 256 - 8000 символів.

Аналогічний йому тип ***LONG**[VARCHAR]* – розміром до 16K або 2G, залежно від реалізації. Різні СУБД організують збереження даних цих двох типів або в самій таблиці, або, найчастіше, в *окремому файлі*. А в таблиці зберігаються тільки *посилання на зсув* конкретного значення.

***BIT**(n)* – бітовий рядок *n*-ї довжини. Призначена для збереження двійкових значень як *єдиного цілого* (коди програм, зображення і т.д.).

Сьогодні більшість СУБД замість цього типу даних використовують тип ***BLOB*** – *Binary Large OBject*, який зберігається аналогічно типу *VARCHAR*.

***DEC**[IMAL](n,m)* – десяткове число в *дійсному виді*, тобто з явною десятковою крапкою. *n* – загальна кількість розрядів числа, *m* – кількість розрядів після крапки. Причому $m \le n$. Максимальна кількість розрядів *p* залежить від СУБД, але *p* повинно бути не менше *n*.

***NUMERIC**(n,m)* – аналогічно *DEC*, крім того, що *p* може бути не більше *n*.

***INT**[EGER]* – десяткове ціле. Його максимальний розмір і, відповідно, значення залежать від СУБД.

SMALLINT – підтримується не всіма СУБД. Аналогічний *INT*, але в два рази менше.

***FLOAT**(n)* – десяткове число з плаваючою крапкою, представлене *в експонентній* формі $x \times 10^y$, *де* n – кількість розрядів *числа* x, максимальне значення якого, як і максимальне значення *числа* y, залежить від конкретної реалізації.

REAL – збігається з *FLOAT*, за винятком того, що значення *n* дорівнює

максимальному і встановлюється залежно від реалізації.

DOUBLE – у 2 рази більше *REAL*.

DATE, TIME – назви говорять самі за себе. Характерним є те, що практично всі СУБД дозволяють установлювати формат введення і відображення даних цих типів, хоча кожна СУБД зберігає такі дані у своєму *внутрішньому специфічному форматі*, незалежному від формату відображення.

Домени

Для створення **домену** використовуються такі команди:

CREATE DOMAIN *ім'я_домену* *тип_поля*
[DEFAULT *значення*]
[*список обмежень*];

Змінити або видалити значення за замовчуванням і список обмежень даного домену можна командою, аналогічною наведеній, ALTER DOMAIN.

Знищити існуючий домен можна командою

DROP DOMAIN *домен* *опція*;

де *опція* може приймати значення:

RESTRICT – домен не буде знищений, якщо на нього є хоча б одне посилання в таблицях;

CASCADE – команда буде виконана не тільки для самого домена, але й для таблиць з атрибутами, визначеними на його основі: тип таких полів буде перепризначений на базовий тип домену.

Істинно користувальницькі типи даних (UDT)

В СУБД ***PostgreSQL***реалізовано можливість створення *істинно користувальницьких типів* даних (*User Defined Type – UDT*) шляхом перелічення всіх можливих значень типу даних, що створюється:

CREATE TYPE *Користувальницький_тип* AS ENUM (*значення*_1, *значення*_2, …);

де ENUM – означає перелічення якихось констант без вказівки їхнього будь-якого базового типу. Наприклад:

```
CREATE TYPE color AS ENUM ('червоний', 'помаранчевий', 'жовтий', 'зелений',
    'синій', 'фіолетовий');
```

Складені типи даних

CREATE TYPE *Назва_типу* AS (*Назва_атрибуту Тип_атрибуту* [, ...])

Наприклад, для створення типу *Address*, який включає дані про місто, вулицю, будинок та квартиру, можна виконати команду:

```
CREATE TYPE Address AS
    ( City VARCHAR, Street VARCHAR, House SMALLINT, Flat SMALLINT);
```

Тепер до таблиці *Student* можна додати новий атрибут *Address* відповідного типу:

ALTER TABLE Student ADD COLUMN Address ADDRESS;

Для внесення даних до атрибуту складного типу використовується два варіанти:

'(*значення*_1 , *значення*_2 , …)'

або

ROW(*значення*_1 ,*значення*_2 , …),

де *значення*_1, *значення*_2, … – значення *атрибуту*_1, *атрибуту*_2 і т.д. опису складеного типу.

Послідовності

В найпростішому варіанті буде створено таблицю з іменем назва, до єдиного кортежу якої за замовчуванням будуть записані такі значення: в поле Ім'я – назва, поля Крок, Мінімальне і Початкове_значення отримають значення 1, а поле

Максимальне_значення – значення (2^{63}–1) або 9 223 372 036 854 775 807:

CREATE SEQUENCE *назва*;

Якщо необхідно задати інші характеристики послідовності, то для цього використовується більш розширений синтаксис команди CREATE SEQUENCE:

CREATE [TEMP[ORARY]] SEQUENCE *назва*
[INCREMENT [BY] *прирощення*]
[MINVALUE *мінімальне_значення* | NO MINVALUE]
[MAXVALUE *максимальне_значення* | NO MAXVALUE]
[START [WITH] *початкове_значення*];

Таблиці

Команди: створення таблиці – CREATE TABLE, модифікації таблиці – ALTER TABLE, її знищення – DROP TABLE.

Найбільш простий варіант команди створення таблиці:

CREATE TABLE *таблиця*
(*поле* *тип_поля* [DEFAULT *значення*][,
поле *тип_поля* [DEFAULT *значення*][,…]]);

Операнди *таблиця* і *поле* визначають імена відповідних елементів. Як *тип_поля* вказується або ім'я домену, або один з базових типів.

Коли створюється таблиця, можна визначити **обмеження** на значення полів таблиці. В *SQL* існує два типи обмежень: на певні стовпці або просто *на стовпці* і на підмножину стовпців або *на таблицю*. Обмеження на той чи інший стовпець вказуються у визначенні відповідного поля і мають позиційне значення *тільки* в тім розумінні, що розміщуються *після оголошення типу*:

поле *тип_поля* *обмеження*_С

Обмеження на таблицю записуються після визначення останнього поля і відокремлюються від нього комою. Більшість обмежень можна вказувати двома способами, але не всі. Для кожного обмеження буде зазначена можливість накладання як обмеження на таблицю.

Для зазначення значення за замовчуванням використовується оператор DEFAULT.

При використанні послідовності (sequence) як обмеження на атрибут вказується наступне:

… ім'я_атрибуту INT PRIMARY KEY DEFAULT NEXTVAL('ім'я_послідовності'), …

Якщо для стовпця не вказано значення за замовчуванням і при додаванні кортежу (рядка) пропущено значення відповідного атрибута, то в комірку таблиці буде введено значення NULL Якщо така ситуація для даного поля неприпустима, то в *обмеження* необхідно ввести оператор NOT NULL.

Іншим варіантом є *обмеження за значенням* – оператор CHECK (можливо накладання як на атрибут, так і на таблицю). Наприклад, може виконуватися перевірка значення на відповідність діапазону. Наприклад:

```
CREATE TABLE     Rating
(kod             INT        CHECK (kod > 0)    NOT NULL,
r_disc1          INT        DEFAULT 0  CHECK(r_disc1 ≥ 0 AND r_disc1 ≤ 100),
r_disc2          INT        DEFAULT 0  CHECK(r_disc2 ≥ 0 AND r_disc1 ≤ 100);
```

Логічні оператори

Результатом їх дії буде значення «істина» або «хиба». До них відносяться оператори порівняння, булеві оператори та спеціальні оператори.

Оператори порівняння аналогічні тим, що застосовуються чи існують у більшості мов програмування: =, >, <, >=, <=, <>.

Булеві оператори в *SQL* реалізують 3 операції булевої алгебри, що утворюють повний базис: AND, OR, NOT.

Спеціальні оператори: IN, BETWEEN, LIKE.

Оператор IN цілком визначає *деяку множину значень*. Наприклад:

... spec CHAR(2) CHECK(spec IN('АП', 'ОС', 'АМ', 'АС')) ...

Оператор BETWEEN разом з оператором AND задає *діапазон значень*. Причому границі діапазону *входять* у число припустимих значень і можуть бути як числами, так і рядками ASCII-символів. В останньому випадку при порівнянні рядків різної довжини більш короткий рядок *доповнюється* пробілами, що мають найменший ASCII-код серед символів алфавіту.

Оператор LIKE застосовуваний *тільки до символьних полів* типу CHAR і VARCHAR і використовується для накладення *шаблонів* на рядки.

Для цього в операторі використовуються спеціальні **символи-шаблони**: символ „підкреслення“ (‘_’), що заміняє *один будь-який* символ, і символ „відсоток“ (‘%’), що заміняє символьний *рядок довільної довжини*. Наприклад, шаблону „*д_м*“ відповідають рядки „*дам*“, „*дим*“, „*дум*“ тощо, а шаблону „*%М%В*“ – в полі прізвища яких закінчуються літерою В, а в середині чи на початку мають літеру М:

SecondName CHAR(60) CHECK(SecondName LIKE ‘%M%B’)

Для того, щоб у шаблоні оператора LIKE використовувати і самі символи ‘_’ та ‘%’ необхідно будь-який символ, наприклад слеш ‘/’, визначити як Escape-символ і випереджати їм кожний з керуючих, у тому числі і самого себе:

… SpName CHAR(60) CHECK(SpName LIKE ‘%/_%/%%//%’ ESCAPE ‘/’), …

У цьому прикладі команда CHECK пропустить будь-які символьні рядки, в яких у будь-якому місці, але послідовно зустрічаються символи ‘_’, ‘%’ і ‘/’. Наприклад:

„У спец_фонд виділити 15% доходу, але не більш 1000 грн/місяць“.

Потенційний ключ:

UNIQUE [(список_атрибутів)]

Це визначення може використовуватися і як обмеження на стовпець, і як обмеження на таблицю. Наприклад, в таблиці Student

... snum INT UNIQUE, …

Первинний ключ:

PRIMARY KEY [(*список_атрибутів*)]

Змінимо попередній приклад:

… PRIMARY KEY (Spec, GNum, SNum));

Це приклад використання первинного ключа як *обмеження на таблицю*. Іноді більш доцільно використовувати *одноатрибутний* привілейований ідентифікатор, тобто використовувати первинний ключ як *обмеження на стовпець*. Наприклад, у таблицю Student можна додати поле

… Kod INT PRIMARY KEY, …

Складений первинний або потенційний ключ можна задавати тільки як обмеження на таблицю.

Зовнішній ключ:

Як обмеження на таблицю:

FOREIGN KEY *список_атрибутів*
REFERENCES *базова_таблиця* [*список_атрибутів*]

Якщо в зовнішньому ключі присутній *тільки один* атрибут, то синтаксис буде трохи

інший (обмеження на атрибут):

… Spec CHAR(2) REFERENCES Speciality (Spec), …

Для модифікації таблиці в *SQL* існує команда

ALTER TABLE *таблиця операція об'єкт*[,
таблиця операція об'єкт[, …]];

де *таблиця* – ім'я таблиці, *операція* – ADD (*додати*) або DROP (*видалити*), *об'єкт* – стовпець (при додаванні указуються всі необхідні параметри) або обмеження типу UNIQUE, PRIMARY KEY, CHECK і т.д.

Знищується таблиця командою

DROP TABLE таблиця опція;

Опціями може бути RESTRICT або CASCADE. Якщо при видаленні таблиці встановлюється опція RESTRICT, а сама таблиця містить батьківські ключі, на які *існують посилання*, то команда буде *відкинута*. Якщо ж у такій ситуації встановлена опція CASCADE, то команда буде виконана, а всі *посилання* на неї будуть *знищені* (замінені NULL або DEFAULT).

При створенні таблиці-нащадка на основі таблиці-предка використовується оператор INHERITS:

CREATE TABLE *таблиця-нащадок* (…) INHERITS (*таблиця-предок* [*атрибут*_1, *атрибут*_2, …])

Усі *нові кортежі* в *SQL* **вводяться** в таблиці за допомогою команди INSERT:

INSERT INTO *таблиця* VALUES(*значення*, *значення*, …);

У такому варіанті команди число значень повинне відповідати числу атрибутів таблиці. Значення при цьому мають *позиційне значення* і не можуть містити *вирази*.

Якщо ж необхідно „пропустити" значення якого-небудь поля при введенні, це можна реалізувати *двома способами*:

— уставити *NULL-значення* у відповідну позицію:

INSERT INTO Student VALUES(111, 'Іванов, 'Іван', NULL, 12, 'AM', 823, NULL, NULL);

Це можливо тільки в тому випадку, якщо в даному полі *припустимі* такі значення;

— *явно вказати імена* стовпців, до котрих будуть уведені нові значення. Пропущеним атрибутам будуть надані або NULL-значення, якщо вони припустимі, або значення за замовчуванням, якщо вони існують (задані).

При використанні послідовності (sequence) значення первинного ключа не заповнюється.

Рядки з таблиці можна **виключити** за допомогою команди

DELETE FROM *таблиця* [*умова*];

Якщо *не* вказана *умова*, то команда виконує „очищення" таблиці:

DELETE FROM Rating;

Використання умови дозволяє видаляти *кілька кортежів* таблиці:

DELETE FROM Student WHERE GNum < 941;

Якщо в умові вказане ім'я *первинного ключа*, то це забезпечить видалення *одного єдиного* кортежу.

Наступна команда МВД *SQL* дозволяє **змінювати** значення **вже існуючих** полів таблиці:

UPDATE *таблиця* SET *поле = значення*[, *поле = значення*[, …]] [*умова*];

Коротка «шпаргалка»

Операторі між символами «[ ]» – не є обов'язковими.
Для створення таблиць:
CREATE TABLE *ім'я_таблиці* (
{*ім'я_поля тип_даних* [DEFAULT *значення за замовчуванням*] [*обмеження поля*]}
[,{…}]
обмеження таблиці
);

Обмежень поля може бути декілька, записаних для поля через пробіл:
NOT NULL – не пусте
NULL – пусті значення дозволені (задається за замовчуванням)
UNIQUE – значення поля унікальні (потенційний ключ)
PRIMARY KEY – первинний ключ
CHECK (*вираз*) – обмеження на значення
REFERENCES *зв'язана_таблиця* [(*зв'язане_поле*)] [ON DELETE *action*] [ON UPDATE *action*] – визначення зв'язку між таблицями через зовнішній ключ
action (NO ACTION; RESTRICT; CASCADE; SET NULL; SET DEFAULT)

Обмежень таблиці може бути декілька, записанних для поля через пробіл:
UNIQUE (*ім'я_поля* [, ...]) – унікальне значення (потенційний ключ)
PRIMARY KEY (*ім'я_поля* [, ...]) – первинний ключ (може бути складеним)
CHECK (*вираз*) – обмеження на значення
FOREIGN KEY (*ім'я_поля* [, ...]) REFERENCES *зв'язана_таблиця* [(*зв'язане_поле* [, ...])] [ON DELETE *action*] [ON UPDATE *action*] – визначення зв'язку між таблицями через зовнішній ключ

Для додавання записів в таблицю:
INSERT INTO *ім'я_таблиці* [(*ім'я_поля* [, ...])]
VALUES (*значення_поля* [, ...])

Для зміни запису:
UPDATE *ім'я_таблиці* SET *ім'я*_поля = *значення_поля*
[WHERE *ім'я*_поля = умова]

Для видалення запису:
DELETE FROM *ім'я_таблиці* [WHERE *ім'я*_поля = умова]

Приклад рішення завдання до лабораторної роботи 4

Запишіть *SQL*-запити для створення фізичної моделі даних на основі логічної моделі, що спроектована у лабораторній роботі 3.

П.1. Обґрунтуйте задані типи полів.

П.2. Запишіть *SQL*-запити для створення домену операцій з квитками.

П.3. Задайте та обґрунтуйте обмеження значень всіх полів. Зокрема:

- обмеження на поля створюйте з використанням спеціальних логічних операторів;
- реалізуйте зв'язки між таблицями через обмеження зовнішніх ключів.

Розв'язання.

Зверніть увагу, що в цьому прикладі *SQL*-запити, які створено згідно завданню, включені до *SQL*-сценарію (*SQL-script*). Це текстовий файл, з *SQL*-запитами, які будуть *послідовно виконані*. Тому їхня позиція дуже важлива саме при створені БД: спочатку прописуються команди створення користувальницьких типів даних, далі – доменів, потім –

базових таблиць з батьківськими ключами і лише за ними – команди створення деталізованих таблиць із зовнішніми ключами.

Отже, *приклад сценарію для побудови таблиць*:

```
CONNECT  c:\work\sql\lectures  USER GMG  PASSWORD <password>;
CREATE SEQUENCE s_aircompany;
CREATE SEQUENCE s_voyage;
CREATE SEQUENCE s_class;
CREATE SEQUENCE s_passenger;
CREATE SEQUENCE s_employee;
CREATE SEQUENCE s_ticket;
/* Domain: operation, Owner: GMG */
CREATE DOMAIN operation CHAR(7) DEFAULT 'покупка'
                                    CHECK (VALUE IN ('покупка', 'бронь'));
/* Table: aircompany, Owner: GMG */
CREATE TABLE aircompany
      (id_aircompany INT PRIMARY KEY DEFAULT NEXTVAL('s_aircompany'),
       name CHAR (50),
       country CHAR (30));
/* Table: voyage, Owner: GMG */
CREATE TABLE voyage
      (id_voyage INT PRIMARY KEY DEFAULT NEXTVAL('s_voyage'),
       number CHAR(20),
       destination CHAR (30),
       date_departure DATE,
       time_departure TIME,
       date_arrival DATE,
       time_arrival TIME,
       type_aircraft CHAR(30),
       aircompany INT REFERENCES aircompany(id_aircompany));
/* Table: class, Owner: GMG */
CREATE TABLE class
      (id_class INT PRIMARY KEY DEFAULT NEXTVAL('s_class'),
       name CHAR (30),
       voyage INT REFERENCES voyage(id_voyage),
       price DECIMAL(50,2),
       quantity_place INT);
/* Table: passenger, Owner: GMG */
CREATE TABLE passenger
      (id_passenger INT PRIMARY KEY DEFAULT NEXTVAL('s_passenger'),
       full_name CHAR (30),
       sex CHAR (1) CHECK(sex IN('м','ж')),
       passport CHAR (30));
/* Table: employee, Owner: GMG */
CREATE TABLE employee
      (id_employee INT PRIMARY KEY DEFAULT NEXTVAL('s_employee'),
       full_name CHAR(30),
       position CHAR(30),
       date_birth DATE,
       passport CHAR(10),
       identification_code FLOAT,
       adress CHAR(55),
       telefon CHAR(15));
```

```
/* Table: ticket, Owner: GMG */
    CREATE TABLE ticket
        (id_ticket INT PRIMARY KEY DEFAULT NEXTVAL('s_ticket'),
         passenger INT REFERENCES passenger(id_passenger),
         class INT REFERENCES class(id_class),
         place INT,
         employee INT REFERENCES employee(id_employee),
         operation operation);
    EXIT;
```

П.4. Складіть *SQL*-запити щодо введення нової інформації до таблиць бази даних, що створені у попередньому пункті. При цьому визначте та відстежте послідовність наповнення таблиць для зберігання посилальної цілісності.

Розв'язання.

Згідно схемі даних таблиці *aircompany*, *passenger* та *employee* містять батьківські ключі для таблиць *voyage* та *ticket* і не мають зовнішніх ключів. Тому вони створювались першими, і до них в першу чергу повинні вноситись нові кортежі. Таблиця *voyage* містить батьківський ключ для таблиці *class*, і до неї новий кортеж буде внесений наступним. Потім – у таблицю *class* з батьківським ключем для таблиці *ticket*, яку і створено останньою, і нові дані до неї можна буде внести тільки при наявності відповідних значень у таблицях *class*, *passenger* та *employee*.

```
INSERT INTO    aircompany VALUES (NEXTVAL('s_aircompany'), 'Lufthansa',
        'Німеччина');
```

Для цього прикладу можна далі заповнити повністю кожну з таблиць за означеною чергою, а можна, як в реальній системі: до відповідної таблиці(-ць) внести кортеж(і) з батьківським(и) ключем(-ами), а потім – всі кортежі з залежними від нього(них) зовнішніми ключами.

```
INSERT INTO    voyage VALUES (NEXTVAL('s_voyage'), 'RK 4578', 'Одеса – Нью-
        Йорк', '2012-08-09', '12:30:00', '2012-08-10', '03:40:00', 'Boeing 747', 1);
INSERT INTO    class      VALUES (NEXTVAL('s_class'), 'бізнес', 1, 1000.00, 50);
INSERT INTO    passenger VALUES (NEXTVAL('s_passenger'), 'Чугунов А.А.', 'м',
        'КН 123456');
INSERT INTO    employee VALUES (NEXTVAL('s_employee'), 'Арсірій О.О.',
        'касир', '1966-03-06', 'КМ 555555', 56788765349876, 'м. Одеса, вул. Блакитна,
        б. 6, кв. 35', 2345678);
INSERT INTO    ticket     VALUES (NEXTVAL('s_ticket'), 4, 6, 10, 2, 'покупка');
```

Проконтролювати введені кортежі можна за допомогою відповідних команд SELECT * FROM *таблиця*;

Наприклад.

```
SELECT * FROM aircompany;
```

id_aircompany	name	country
1	Lufthansa	Німеччина
2	Аеросвіт	Україна
3	Трансаэро	Україна
4	Днеправіа	Україна
5	Turkish Airlines	Турція

```
SELECT * FROM class;
```

id_class	Name	voyage	price	quantity_place
1	Бізнес	1	1000.00	50
2	економ покращений	1	800.00	50
3	Економ	1	500.00	100
4	Бізнес	2	1200.00	50

id_class	Name	voyage	price	quantity_place
5	економ покращений	2	600.00	50
6	Економ	2	300.00	100
7	бізнес	3	1500.00	50
8	економ покращений	3	1400.00	50
9	економ	3	1100.00	100
10	бізнес	4	2000.00	50
11	економ	4	1800.00	50
12	економ	4	1500.00	100

SELECT * FROM passenger;

id_passenger	full_name	sex	passport
1	Малахов Є.В.	м	КМ 123456
2	Чугунов А.А.	м	КН 123456
3	Альохін О.Б.	м	КМ 654321
4	Філиппова С.В.	ж	КЕ 123456
5	Журан О.А.	ж	КМ 234567
6	Лінгур Л.М.	ж	КМ 345678
7	Андрієнко В.М.	ж	КМ 456789

SELECT * FROM voyage;

id_ voyage	number	destination	date_ departure	time_ departure	date_ arrival	time_ arrival	type_ aircraft	aircom-pany
1	RK 4578	Одеса – Нью-Йорк	2012-08-09	12:30:00	2012-08-10	03:40:00	Boeing 747	1
2	RK 3467	Одеса – Лондон	2009-06-11	18:40:00	2009-06-12	02:50:00	A310	1
3	RK 3478	Одеса – Бухарест	2009-03-19	17:40:00	2009-03-19	17:20:00	АН 158	2
4	RK 2398	Одеса – Пекін	2009-05-05	12:10:00	2009-05-06	15:50:00	Boeing 747	2
5	RK 3490	Одеса – Рим	2009-06-12	18:45:00	2009-06-13	16:20:00	ИЛ 76	2

SELECT * FROM employee;

id_ emp	full_name	position	date_birth	passport	identification_ code	adress	telefon
1	Шевченко Г.В.	Касир	1956-10-04	КМ 777777	23458765349876	м. Одеса, вул. Незнайкіна, б. 56, кв. 45	1234567
2	Некрасов О.О.	Касир	1966-03-06	КМ 555555	56788765349876	м. Одеса, вул. Блакитна, б. 6, кв. 35	2345678
3	Погорецька В.Я.	Касир	1974-09-12	КМ 444111	23458456349876	м. Одеса, пл. Рожева, б. 2, кв. 32	3456789
4	Івченко І.Ю.	Касир	1984-05-10	КМ 222333	85474776534987	м. Одеса, вул. Біла, б. 23, кв. 4	7654321

SELECT * FROM ticket;

id_ticket	passenger	class	place	employee	operation
1	4	6	10	2	покупка
2	1	5	12	2	бронь
3	3	4	11	4	покупка

id_ticket	passenger	class	place	employee	operation
4	5	2	14	4	покупка
5	2	2	21	4	покупка
6	7	2	6	4	покупка
7	6	2	4	4	бронь

Структура звіту до лабораторної роботи

1. Копія логічної моделі, яку створену в лабораторній роботі 3.
2. SQL-код створення об’єктів (домени, послідовності, UDT, таблиці) бази даних. Порядок створення об’єктів повинен бути збережений. Обов’язково прописати всі зв’язки між таблицями, ключі, обмеження.
3. SQL-код додавання записів в таблиці. Як мінімум по 10 кортежів в кожну таблицю. Порядок заповнення таблиць повинен бути збережений.
4. Приклади SQL-коду видалення записів з таблиць з словесним описом дії (мінімум 5).
5. Приклади SQL-коду зміни даних в таблицях з словесним описом дії (мінімум 5).
6. Скріншоти створених та заповнених таблиць.

Звіт до лабораторної роботи 4 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.