

**Національний технічний університет України**

**“Київський політехнічний інститут”**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп’ютерних систем**

**Лабораторна робота №1**

***з дисципліни “* *Бази даних і засоби управління”***

*Тема: “Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Таранич Артем

Перевірив: Павловський В. І.

Київ – 2022

*Метою роботи* є здобуття вмінь проектування бази даних та практичних навичок створення реляційних баз даних за допомогою PostgreSQL.

*Завдання* роботи полягає у наступному:

1. Розробити модель «сутність-зв’язок» предметної галузі, обраної студентом самостійно, відповідно до пункту «Вимоги до ER-моделі».
2. Перетворити розроблену модель у схему бази даних (таблиці) PostgreSQL.
3. Виконати нормалізацію схеми бази даних до третьої нормальної форми (3НФ).
4. Ознайомитись із інструментарієм PostgreSQL та pgAdmin 4 та занести декілька рядків даних у кожну з таблиць засобами pgAdmin 4.

*Вимоги до ER-моделі:*

1. Сутності моделі предметної галузі мають містити зв’язки типу 1:N або N:M.
2. Кількість сутностей у моделі – 3-4. Кількість атрибутів у кожній сутності: від двох до п’яти.
3. Передбачити наявність зв’язку з атрибутом.

Для побудови ER-діаграм використовувати одну із нотацій: Чена, “Пташиної лапки (Crow’s foot)”, UML.

**Частина № 1**

**Опис предметної галузі**

Для виконання лабораторної роботи я обрав тему – сайт для продажу авіаквитків.

Для розробки для даного сайту я видів наступні сутності: відомості про літак (Aircraft), відомості про аеропорт (Airport), відомості про білет (Ticket), відомості про акаунт користувача (User Account).

Атрибути заданих сутностей:

* + - 1. Aircraft: pilot, name, owner
      2. Airport: name, location.
      3. Ticket: price, class, date, flight time
      4. User Account: name, mailbox, bank card

**Опис зв’язків між сутностями предметної області**

На літак може бути придбано якась кількість білетів, тому зв’язок між сутностями «Aircraft» і «Ticket» - 1:N.

На один білет може претендувати тільки один користувач й одночасно з цим користувач може придбати кілька білетів, тому зв’язок між сутностями «User Account» і «Ticket» - 1:N.

Сутність «Aircraft» має зв’язок M:N по відношенню до сутності «Airport», тому що один літак має потрапити в декілька аеропортів (з якого вилітає, куди має прилетіти), в той же на одному аеропорту можуть знаходитися декілька кілька літаків.

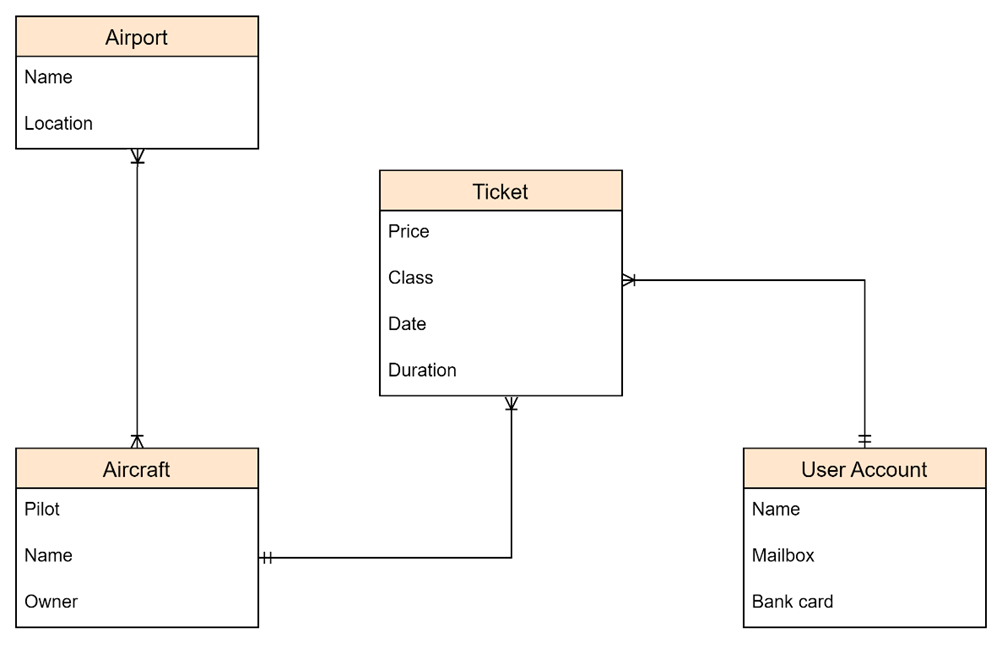


Рисунок 1 - ER-діаграма побудована за нотацією “Пташиної лапки”, побудована у додатку *draw.io*

**Частина № 2**

**Перетворення концептуальної моделі у схему баз даних**

Для кожної сутності створюється таблиця. Причому кожному атрибуту сутності відповідає стовпець таблиці. В даній моделі перетворення в схему баз даних відбувалося за такими правилами:

Якщо зв'язок типу 1: N і клас приналежності сутності на стороні N є обов'язковим, то необхідно побудувати таблицю для кожної сутності. Первинний ключ сутності повинен бути первинним ключем відповідної таблиці. Первинний ключ сутності на стороні 1 додається як атрибут в таблицю для сутності на стороні N.

Якщо зв'язок типу N: M, то необхідно побудувати три таблиці - по одній для кожної сутності і одну для зв'язку. Первинний ключ сутності повинен бути первинним ключем відповідної таблиці. Таблиця для зв'язку серед своїх атрибутів повинна мати ключі обох сутностей.

Сутність «User Account» перетворено у таблицю «user\_account».

Сутність «Ticket» перетворено у таблицю «ticket».

Зв’язок 1:N між сутностями «User Account» та «Ticket» та зв’язок 1:N між сутностями «Aircraft» та «Ticket» зумовив появу 2 зовнішніх ключів: aircraft\_id та user\_acc\_id у таблицю «ticket».

Сутність «Aircraft» перетворено у таблицю «aircraft».

Сутність «Airport» перетворено у таблицю «airport».

Зв’язок M:N між сутністю «Aircraft» та «Airport» зумовив появу додаткової таблицю «airport/aircraft» із зовнішніми ключами aircraft\_id та airport\_id.

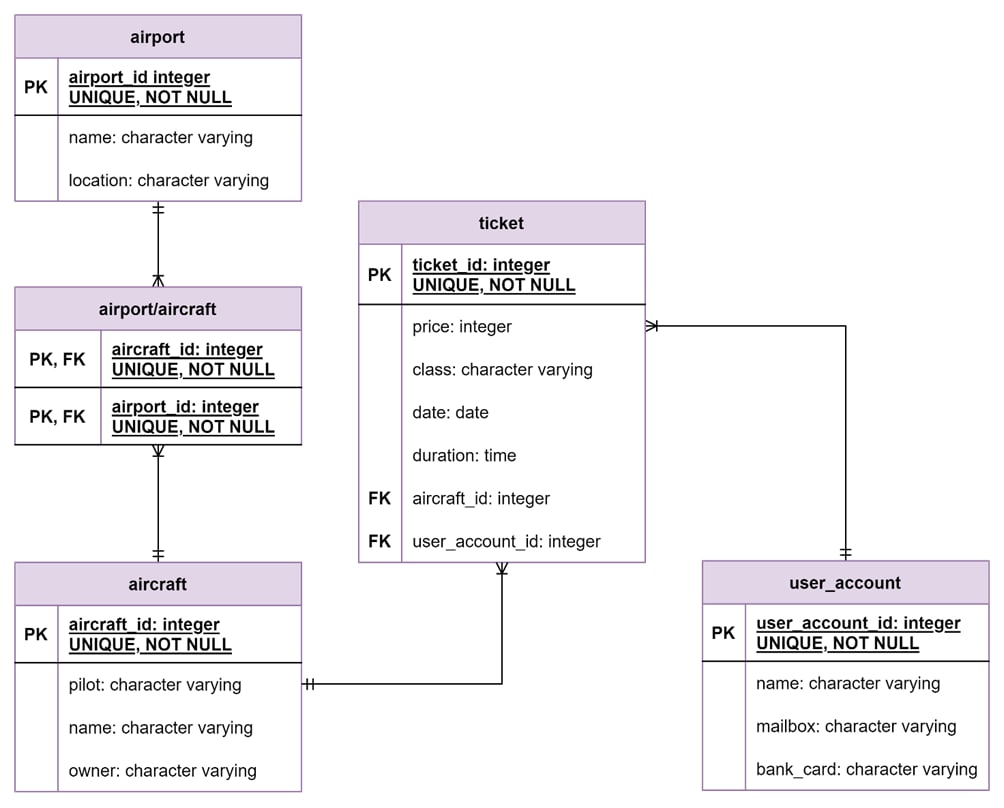


Рисунок 2 - Схема бази даних, побудована у додатку draw.io

**Частина № 3**

**Відповідність схеми бази даних до третьої нормальної форми**

Схема відповідає 1НФ, тому що:

* Значення в кожній комірці є атомарним;
* В таблиці немає дубльованих рядків.
* В кожному стовпці зберігаються дані одного типу.

Схема відповідає 2НФ, тому що:

* Вона відповідає 1НФ.
* Кожен неключовий атрибут функціонально залежить від всього ключа, а не від його частини.

Схема відповідає 3НФ, тому що:

* Вона відповідає 2НФ.
* Кожен не простий атрибут, є функціонально залежним від головних ключів.

**Частина № 4**

Фізична модель БД «Cайт для продажу авіаквитків» у pgAdmin4

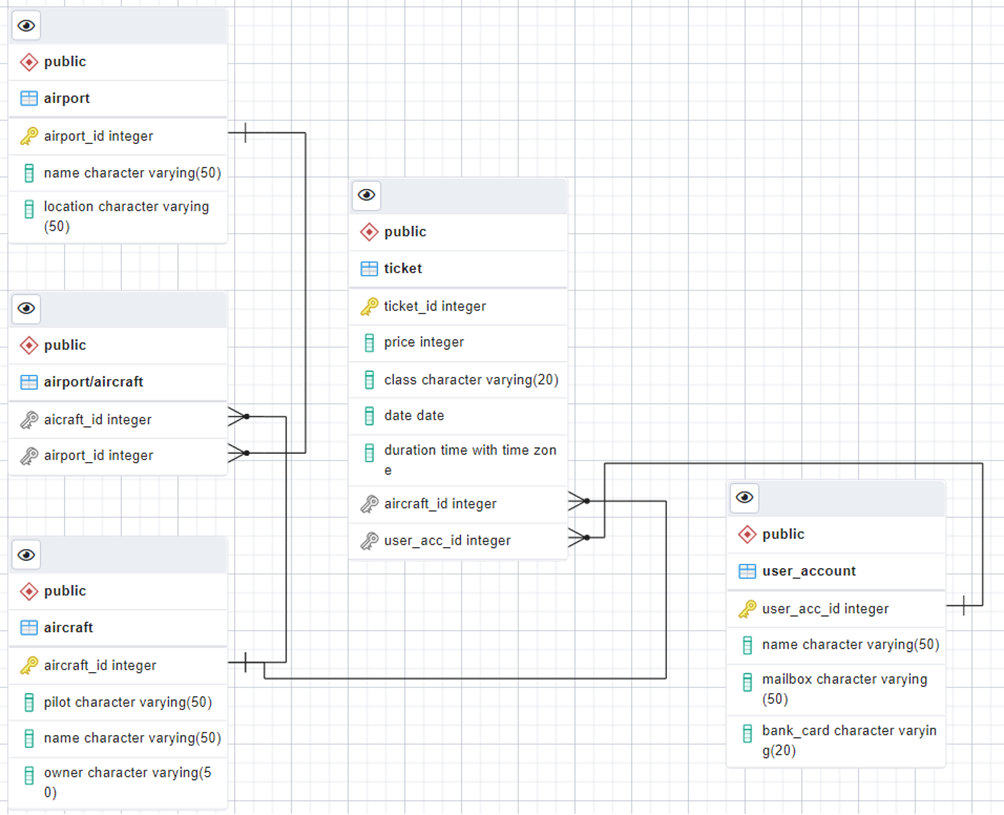


Рисунок 3 - Схема бази даних, побудована у pgAdmin 4

Airport

Graphical user interface, application, table

Description automatically generated

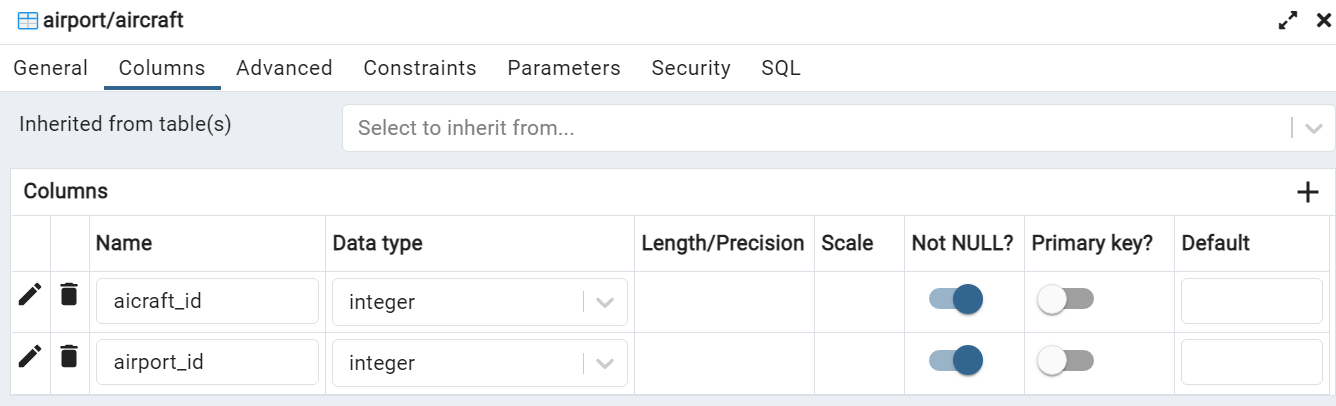
Graphical user interface, application

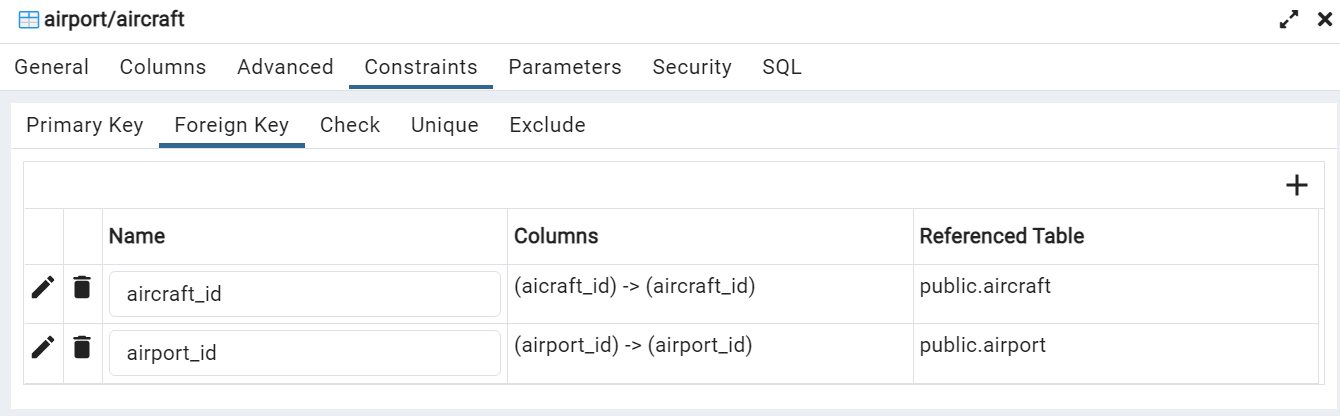
Description automatically generated

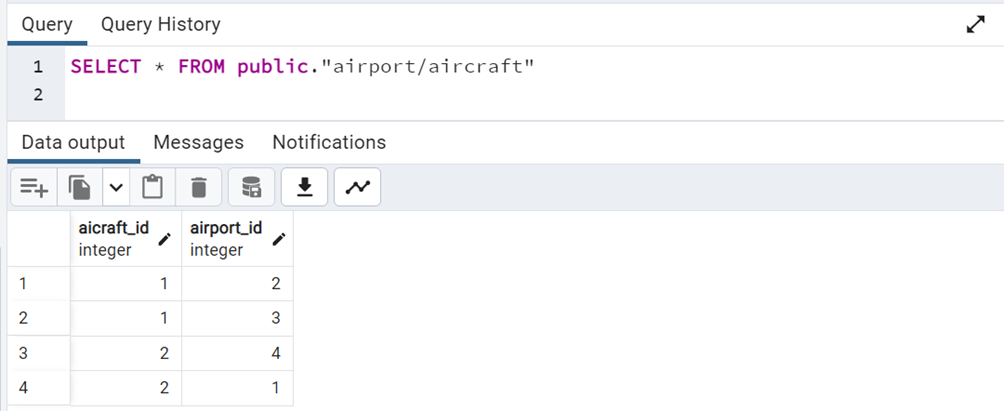
Graphical user interface, text, application

Description automatically generated

airport/aircraft







Aircraft

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

User account

Table

Description automatically generated

Graphical user interface, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Ticket

Table

Description automatically generated

Graphical user interface, table

Description automatically generated with medium confidence

Graphical user interface, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

SQL TEXT

-- Table: public.airport

-- DROP TABLE IF EXISTS public.airport;

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **public**.airport

(

airport\_id **integer** **NOT** **NULL**,

name **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

location **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

**CONSTRAINT** airport\_pkey **PRIMARY** **KEY** (airport\_id)

)

**TABLESPACE** pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** **public**.airport

OWNER **to** postgres;

-- Table: public.aircraft

-- DROP TABLE IF EXISTS public.aircraft;

**CREATE** **TABLE** **IF NOT EXISTS** **public**.aircraft

(

aircraft\_id **integer NOT NULL**,

pilot **character varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

name **character varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

owner **character varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

**CONSTRAINT** aircraft\_pkey **PRIMARY** KEY (aircraft\_id)

)

**TABLESPACE** pg\_default;

**ALTER TABLE IF EXISTS** **public**.aircraft

OWNER to postgres;

-- Table: public.airport

-- DROP TABLE IF EXISTS public.airport;

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **public**.airport

(

airport\_id **integer** **NOT** **NULL**,

name **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

location **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

**CONSTRAINT** airport\_pkey **PRIMARY** **KEY** (airport\_id)

)

**TABLESPACE** pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** **public**.airport

OWNER **to** postgres;

-- Table: public.airport/aircraft

-- DROP TABLE IF EXISTS public."airport/aircraft";

**CREATE TABLE IF NOT EXISTS public**."airport/aircraft"

(

aicraft\_id **integer NOT NULL**,

airport\_id **integer NOT NULL**,

**CONSTRAINT** aircraft\_id **FOREIGN** **KEY** (aicraft\_id)

**REFERENCES** **public**.aircraft (aircraft\_id) **MATCH SIMPLE**

O**N** **UPDATE NO ACTION**

**ON DELETE NO ACTION**,

**CONSTRAINT** airport\_id **FOREIGN** **KEY** (airport\_id)

**REFERENCES** **public**.airport (airport\_id) **MATCH** **SIMPLE**

**ON UPDATE NO ACTION**

**ON DELETE NO ACTION**

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."airport/aircraft"

OWNER to postgres;

-- Table: public.ticket

-- DROP TABLE IF EXISTS public.ticket;

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **public**.ticket

(

ticket\_id **integer** **NOT** **NULL**,

price **integer** **NOT** **NULL**,

**class** **character** **varying**(20) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

**date** **date** **NOT** **NULL**,

duration **time** **with** **time** **zone** **NOT** **NULL**,

aircraft\_id **integer** **NOT** **NULL**,

user\_acc\_id **integer** **NOT** **NULL**,

**CONSTRAINT** ticket\_pkey **PRIMARY** **KEY** (ticket\_id),

**CONSTRAINT** aircraft\_id **FOREIGN** **KEY** (aircraft\_id)

**REFERENCES** **public**.aircraft (aircraft\_id) **MATCH** SIMPLE

**ON** **UPDATE** **NO** **ACTION**

**ON** **DELETE** **NO** **ACTION**

**NOT** VALID,

**CONSTRAINT** user\_acc\_id **FOREIGN** **KEY** (user\_acc\_id)

**REFERENCES** **public**.user\_account (user\_acc\_id) **MATCH** SIMPLE

**ON** **UPDATE** **NO** **ACTION**

**ON** **DELETE** **NO** **ACTION**

**NOT** VALID

)

**TABLESPACE** pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** **public**.ticket

OWNER **to** postgres;

-- Table: public.user\_account

-- DROP TABLE IF EXISTS public.user\_account;

**CREATE** **TABLE** **IF** **NOT** **EXISTS** **public**.user\_account

(

user\_acc\_id **integer** **NOT** **NULL**,

name **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

mailbox **character** **varying**(50) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

bank\_card **character** **varying**(20) **COLLATE** pg\_catalog."default" **NOT** **NULL**,

**CONSTRAINT** user\_account\_pkey **PRIMARY** **KEY** (user\_acc\_id)

)

**TABLESPACE** pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** **public**.user\_account

OWNER **to** postgres;