ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №3

Тема: «Обмеження цілісності даних в SQL даних»

Виконав:

Ст. *Лук'янчук Денис*

Група ПМІ-23

2025

Тема: «Обмеження цілісності даних в SQL даних».

Мета роботи: Ознайомлення з поняттями обмеження цілісності даних в SQL, їх створення і використання.

Завдання лабораторної роботи:

1. Опрацювати теоретичний матеріал.

2. Проаналізувати наявні обмеження цілісності даних в створених таблицях та додати відсутні (особливо звернувши увагу на природні ключі). Проілюструвати знімками екрану з командами створення обмежень.

Теоретичний матеріал

1. Обмеження цілісності даних в SQL

Обмеження цілісності даних у SQL використовуються для забезпечення коректності та узгодженості даних у базі даних. Вони визначають правила, які повинні виконуватися для стовпців або таблиць. Основні типи обмежень:

- CHECK: Перевіряє умову для значень у стовпці.
- NOT NULL: Забороняє зберігати NULL-значення.
- UNIQUE: Гарантує унікальність значень у стовпці.
- PRIMARY KEY: Визначає первинний ключ (унікальний і не NULL).
- FOREIGN KEY: Зв’язує таблиці, посилаючись на первинний ключ іншої таблиці.

Ці обмеження допомагають уникнути помилок, таких як дублювання записів, введення негативних цін чи порушення зв’язків між таблицями.

1.1. Обмеження-перевірки CHECK

Обмеження CHECK дозволяє задавати умову, яку повинні виконувати значення у стовпці. Наприклад, ціна товару не може бути від’ємною.

1.2. Обмеження NOT NULL

Обмеження NOT NULL вимагає, щоб стовпець завжди мав значення, відмінне від NULL.

1.3. Обмеження унікальності UNIQUE

Обмеження UNIQUE гарантує, що всі значення в стовпці будуть унікальними (дозволяється NULL, але лише один раз).

1.4. Первинні ключі PRIMARY KEY

Первинний ключ (PRIMARY KEY) унікально ідентифікує кожен рядок у таблиці і не може містити NULL. Як правило, це природній або синтетичний ключ.

1.5. Зовнішні ключі FOREIGN KEY

Зовнішній ключ (FOREIGN KEY) встановлює зв’язок між таблицями, посилаючись на первинний ключ іншої таблиці.

```sql
-- Table: public.characteristic

-- DROP TABLE IF EXISTS public.characteristic;

CREATE TABLE IF NOT EXISTS public.characteristic
(
    id_characteristic integer NOT NULL DEFAULT nextval('characteristic_id_characteristic_seq'::regclass),
    color character varying(50) COLLATE pg_catalog."default",
    material character varying(50) COLLATE pg_catalog."default" NOT NULL,
    weight numeric(10,2),
    size character varying(20) COLLATE pg_catalog."default",
    unit unit_type COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT characteristic_pkey PRIMARY KEY (id_characteristic),
    CONSTRAINT check_weight_positive CHECK (weight > 0::numeric)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.characteristic
    OWNER to postgres;
```


```sql
-- Table: public.characteristic

-- DROP TABLE IF EXISTS public.characteristic;

CREATE TABLE IF NOT EXISTS public.characteristic
(
    id_characteristic integer NOT NULL DEFAULT nextval('characteristic_id_characteristic_seq'::regclass),
    color character varying(50) COLLATE pg_catalog."default",
    material character varying(50) COLLATE pg_catalog."default" NOT NULL,
    weight numeric(10,2),
    size character varying(20) COLLATE pg_catalog."default",
    unit unit_type COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT characteristic_pkey PRIMARY KEY (id_characteristic),
    CONSTRAINT check_weight_positive CHECK (weight > 0::numeric)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.characteristic
    OWNER to postgres;
```


```sql
-- Table: public.appuser

-- DROP TABLE IF EXISTS public.appuser;

CREATE TABLE IF NOT EXISTS public.appuser
(
    id_user integer NOT NULL DEFAULT nextval('appuser_id_user_seq'::regclass),
    username name_type COLLATE pg_catalog."default" NOT NULL,
    email email_type COLLATE pg_catalog."default",
    last_name name_type COLLATE pg_catalog."default",
    phone_number phone_type COLLATE pg_catalog."default",
    registration_date date_type DEFAULT CURRENT_DATE,
    address character varying(255) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT appuser_pkey PRIMARY KEY (id_user),
    CONSTRAINT appuser_email_key UNIQUE (email),
    CONSTRAINT unique_phone_number UNIQUE (phone_number)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.appuser
    OWNER to postgres;
```

Висновок: У ході роботи з теми «Обмеження цілісності даних в SQL» досягнуто мети ознайомлення з їх створенням і використанням. Проаналізовано базу `PartsTradeDB` і додано відсутні обмеження: `NOT NULL` (наприклад, `Country`, `Address`), `UNIQUE` (наприклад, `Phone_Number`, `Contact_Information`), `CHECK` (наприклад, `Weight > 0`, `Price > 0`). Це підвищило цілісність даних, підтверджено знімками екрану в pgAdmin.

Відповіді на контрольні питання

1. Перерахуйте відомі вам обмеження цілісності даних.

- `PRIMARY KEY` — унікальний ідентифікатор рядка.
- `FOREIGN KEY` — зв’язок між таблицями через первинний ключ іншої таблиці.
- `UNIQUE` — гарантує унікальність значень у стовпці.
- `NOT NULL` — забороняє NULL значення у стовпці.
- `CHECK` — задає логічні умови для значень (наприклад, `Price > 0`).

2. Що означає обмеження цілісності даних первинний ключ (Primary Key)?

`PRIMARY KEY` — це обмеження, яке визначає стовпець (або комбінацію стовпців) як унікальний ідентифікатор кожного рядка в таблиці. Воно не дозволяє дублювати значення і не допускає `NULL`. Наприклад, `ID_User` у таблиці `AppUser` унікально ідентифікує кожного користувача.

3. Що означає обмеження цілісності даних зовнішній ключ (Foreign Key)?

`FOREIGN KEY` — це обмеження, яке встановлює зв’язок між двома таблицями, посилаючись на `PRIMARY KEY` іншої таблиці. Воно забезпечує, що значення у стовпці відповідають існуючим значенням у пов’язаному стовпці. Наприклад, `ID_User` у `Orders` посилається на `ID_User` у `AppUser`, гарантуючи, що замовлення пов’язане з існуючим користувачем.

4. Що означає обмеження цілісності даних UNIQUE?

`UNIQUE` — це обмеження, яке гарантує, що всі значення у стовпці (або комбінації стовпців) є унікальними, тобто не повторюються. Наприклад, `Email` у `AppUser` має бути унікальним для кожного користувача, але на відміну від `PRIMARY KEY`, може приймати `NULL`.

5. Поясніть обмеження цілісності даних NOT NULL.

`NOT NULL` — це обмеження, яке забороняє стовпцю приймати значення `NULL`, тобто стовпець завжди має містити дійсне значення. Наприклад,

`Username` у `AppUser` із `NOT NULL` означає, що кожен користувач мусить мати ім’я.

6. Наведіть приклад обмеження цілісності даних CHECK.

Обмеження `CHECK` задає логічну умову для значень у стовпці. Наприклад:

CREATE TABLE Part (

Price DECIMAL(10, 2) CHECK (Price > 0)

);

Це обмеження гарантує, що ціна товару (`Price`) у таблиці `Part` завжди буде додатною (більшою за 0). Якщо спробувати вставити `Price = -10`, виникне помилка.