Лекція 20

Робота з БД

20.1 Створення БД

Інструкція SQL CREATE DATABASE використовується для створення нової бази даних SQL.

Синтаксис

CREATE DATABASE *databasename*;

Приклад:

Наступний оператор SQL створює базу даних під назвою "testDB":

CREATE DATABASE testDB;

Зауваження. Переконайтеся, що у вас є права адміністратора, перш ніж створювати будь-яку базу даних. Після створення бази даних ви можете перевірити її у списку баз даних за допомогою команди SQL: SHOW DATABASES;

20.2 Видалення БД

Інструкція SQL DROP DATABASE використовується для видалення існуючої бази даних SQL.

Синтаксис

DROP DATABASE *databasename*;

Зауваження. Будьте обережні, перш ніж видаляти базу даних. Видалення бази даних призведе до втрати повної інформації, що зберігається в базі даних!

Наступний оператор SQL видаляє існуючу базу даних "testDB":

DROP DATABASE testDB;

Зауваження. Переконайтеся, що у вас є права адміністратора, перш ніж видаляти будь-яку базу даних. Після видалення бази даних ви можете перевірити її відсутність у списку баз даних за допомогою команди SQL: SHOW DATABASES;

20.3 Створення резервної копії БД

Інструкція SQL BACKUP DATABASE використовується в SQL Server для створення повної резервної копії існуючої бази даних SQL.

Синтаксис

BACKUP DATABASE *databasename*
TO DISK = '*filepath*';

20.4 Диференціальне резервне копіювання

Інструкція SQL BACKUP WITH DIFFERENTIAL виконує диференціальне резервне копіювання, тобто створює резервні копії лише тих частин бази даних, які змінилися після останнього повного резервного копіювання бази даних.

Синтаксис

BACKUP DATABASE *databasename*
TO DISK = '*filepath*'
WITH DIFFERENTIAL;

Наступний оператор SQL створює повну резервну копію існуючої бази даних "testDB" на диск D:

BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';

Зауваження. Завжди створюйте резервну копію бази даних на диску, відмінному від фактичної бази даних. Тоді, якщо у вас виникне збій диска, ви не втратите файл резервної копії разом із базою даних.

20.5 Створення таблиці БД

Інструкція SQL CREATE TABLE використовується для створення нової таблиці в базі даних.

Синтаксис

CREATE TABLE *table_name* (
  column1 datatype,
  column2 datatype,

```
    column3 datatype,
   ....
);
```

Параметри стовпців задають імена стовпців таблиці.

Параметр datatype визначає тип даних, які може містити стовпець (наприклад, varchar, integer, date тощо).

*Зауваження***.** Щоб отримати огляд доступних типів даних, перейдіть до повного Довідника типів даних.

Приклад створення таблиці

У наведеному нижче прикладі створюється таблиця Persons, яка містить п’ять стовпців: ідентифікатор особи, прізвище, ім’я, адреса та місто:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Стовпець PersonID має тип int і містить ціле число.

Стовпці Прізвище, Ім’я, Адреса та Місто мають тип varchar і містять символи, а максимальна довжина цих полів становить 255 символів.

Тепер порожня таблиця «Особи» матиме такий вигляд:

PersonID	LastName	FirstName	Address	City

Тепер порожню таблицю можна заповнити даними за допомогою оператора SQL INSERT INTO .

20.6 Створення таблиці за допомогою іншої таблиці

Копію існуючої таблиці також можна створити за допомогою CREATE TABLE.

Нова таблиця отримує ті самі визначення стовпців. Можна вибрати всі стовпці або окремі стовпці.

Якщо ви створюєте нову таблицю, використовуючи вже існуючу таблицю, нова таблиця буде заповнена існуючими записами зі старої таблиці.

Синтаксис

CREATE TABLE *new_table_name* AS
  SELECT *column1, column2,...*
  FROM *existing_table_name*
  WHERE ....;

Наступний SQL створює нову таблицю під назвою «TestTables» (яка є копією таблиці «Customers»):

CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;

20.7 Видалення таблиці

Інструкція SQL DROP TABLE використовується для видалення існуючої таблиці в базі даних.

Синтаксис

DROP TABLE *table_name*;

*Зауваження***.** Видалення таблиці призведе до втрати повної інформації, що зберігається в таблиці.

Наступна інструкція SQL видаляє наявну таблицю " Shippers:

DROP TABLE Shippers;

20.8 Видалення даних з таблиці

Інструкція TRUNCATE TABLE використовується для видалення даних з таблиці, але не самої таблиці.

TRUNCATE TABLE *table_name*;

20.9 Модифікація таблиці

Інструкція ALTER TABLE використовується для додавання, видалення або зміни стовпців у існуючій таблиці.

Інструкція ALTER TABLEтакож використовується для додавання та видалення різноманітних обмежень на існуючу таблицю (завдання властивостей полів таблиці).

Щоб *додати стовпець* у таблицю, використовуйте такий синтаксис:

ALTER TABLE *table_name*
ADD *column_name datatype*;

Наступна SQL інструкція додає стовпець електронної пошти до таблиці клієнтів:

ALTER TABLE Customers
ADD Email varchar(255);

Щоб ***видалити стовпець*** з таблиці слід використати наступний синтаксис (зверніть увагу, що деякі системи баз даних не дозволяють видаляти стовпець):

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

Наступний SQL видаляє стовпець електронної пошти з таблиці клієнтів:

ALTER TABLE Customers
DROP COLUMN Email;

Щоб *перейменувати стовпець* у таблиці слід використати наступний синтаксис:

ALTER TABLE *table_name*
RENAME COLUMN *old_name* to *new_name*;

Щоб *змінити тип даних стовпця* в таблиці слід використати наступний синтаксис:

SQL Server / MS Access:

ALTER TABLE *table_name*
ALTER COLUMN *column_name datatype*;

Mu SQL / Oracle (попередня версія 10G):

ALTER TABLE *table_name*
MODIFY COLUMN *column_name datatype*;

Oracle 10G і новіші версії:

ALTER TABLE *table_name*
MODIFY *column_name datatype*;

Розглянемо використання інструкції SQL ALTER TABLE на прикладі таблиці Persons:

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Потрібно додати стовпець з назвою DateOfBirth у таблицю Persons. Для цього використовуємо оператор SQL:

ALTER TABLE Persons
ADD DateOfBirth date;

Зверніть увагу, що новий стовпець "DateOfBirth" має тип date і міститиме дату. Тип даних визначає, який тип даних може містити стовпець.

Тепер таблиця Persons матиме наступний вигляд:

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Тепер потрібно змінити тип даних стовпця з назвою DateOfBirth у таблиці Persons. Для цього використовуємо наступний оператор SQL:

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

Зверніть увагу, що стовпець DateOfBirth тепер має тип року і буде містити рік у дво- або чотиризначному форматі.

Тепер потрібно видалити стовпець з назвою DateOfBirth у таблиці Persons:

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

Тепер таблиця Persons матиме наступний вигляд:

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

20.10 Властивості полів таблиці SQL

Властивості полів можна вказати під час створення таблиці за допомогою оператора CREATE TABLE або після створення таблиці за допомогою оператора ALTER TABLE.

Синтаксис

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

Обмеження або властивості полів SQL використовуються для визначення умов щодо даних, які вносяться до таблиці. Використовуються для обмеження типу даних, які можуть входити в таблицю. Це забезпечує точність і достовірність даних у таблиці. Якщо існує будь-яке порушення між обмеженням та операцією з даними, операція переривається.

Обмеження можуть бути на рівні стовпця або таблиці.

У SQL зазвичай використовуються такі обмеження:

NOT NULL- Гарантує, що стовпець не може мати значення NULL

UNIQUE- Гарантує, що всі значення в стовпці відрізняються

PRIMARY KEY- Поєднання a NOT NULL та UNIQUE. Поле, за яким можна однозначно ідентифікувати кожен рядок таблиці

FOREIGN KEY - Запобігає діям, які можуть зруйнувати зв'язки між таблицями

CHECK- Перевірка того, що значення в стовпці задовольняють певну умову

DEFAULT- Встановлення значення за замовчуванням для стовпця, якщо значення не вказано користувачем

CREATE INDEX- Використовується для створення та отримання даних з бази даних дуже швидко

Обмеження SQL NOT NULL

За замовчуванням стовпець може містити значення NULL.

Обмеження NOT NULLзмушує стовпець НЕ приймати значення NULL.

Це означає, що поле завжди містить значення, тобто не можна вставити новий запис або оновити запис, не додавши значення до цього поля.

Приклад

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

Приклад

SQL Server / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN Age int NOT NULL;
```

Мій SQL / Oracle (попередня версія 10G):

```
ALTER TABLE Persons
MODIFY COLUMN Age int NOT NULL;
```

Oracle 10G і новіші версії:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

Обмеження SQL UNIQUE

Обмеження UNIQUE гарантує, що всі значення в стовпці відрізняються.

Обмеження, UNIQUE та PRIMARY KEY забезпечують гарантію унікальності для стовпця або набору стовпців.

Поле, яке має обмеження PRIMARY KEY автоматично має й обмеження UNIQUE.

Однак можна встановити багато обмежень UNIQUE на поля таблиці, але лише одне обмеження PRIMARY KEY.

Приклад

Наступний оператор SQL створює обмеження UNIQUE для стовпця ID під час створення таблиці Persons:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

Для визначення обмеження UNIQUE для кількох стовпців, використовується наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

Для визначення обмеження UNIQUE для стовпця ID після створення таблиці використовується наступна SQL інструкція:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

Для визначення обмеження UNIQUE для кількох стовпців, використовується наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

20.11 Зняття обмежень (скасування властивостей)

Для відміни дії обмеження UNIQUE використовється наступна SQL інструкція:

MySQL:

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Person;
```

20.12 Властивість первинного ключа SQL

За даними поля, яке має властивість PRIMARY KEY можна однозначно ідентифікувати кожен запис у таблиці.

Первинні ключі повинні містити значення UNIQUE і не можуть містити значення NULL.

Таблиця може мати лише ОДИН первинний ключ, який може складатися з одного або кількох стовпців (полів).

Приклад

Наступний SQL створює стовпець ID, з властивістю PRIMARY KEY під час створення таблиці Persons:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Для завдання властивості PRIMARY KEY для кількох стовпців, використовується наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
```

```
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Зауваження. У наведеному вище прикладі лише ОДИН PRIMARY KEY(PK_Person). Однак ЗНАЧЕННЯ первинного ключа складається з ДВОХ СТОВПЦІВ (ID + Прізвище).

Приклад

Для завдання властивості PRIMARY KEY для стовпця ID після створення таблиці слід використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Для завдання властивості PRIMARY KEY для кількох стовпців, використовується наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

*Зауваження. Я*кщо ви використовуєте оператор ALTER TABLE для додавання первинного ключа, стовпці первинного ключа повинні бути оголошені такими, що не містять значень NULL (коли таблицю було створено вперше).

Для скасування дії обмеження PRIMARY KEY використовується наступний синтаксис SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

20.13 Властивість зовнішнього ключа SQL

Властивість FOREIGN KEY використовується для запобігання діям, які руйнують зв’язки між таблицями.

Поле з властивістю FOREIGN KEY – це поле (або сукупність полів) в одній таблиці, яке посилається на поле з властивістю PRIMARY KEY в іншій таблиці.

Таблиця із зовнішнім ключем називається дочірньою таблицею, а таблиця з первинним ключем називається батьківською таблицею.

Розглянемо цю властивість на прикладі наступних двох таблиць:

Таблиця Person

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Таблиця Orders

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Зверніть увагу, що поле PersonID у таблиці Orders відповідає полю PersonID у таблиці Persons. Поле PersonID таблиці Persons є первинним ключем (PRIMARY KEY) для таблиці Persons. Поле PersonID таблиці Orders є зовнішнім ключем (FOREIGN KEY) для таблиці Orders.

Властивість FOREIGN KEY запобігає додаванню даних у стовпець зовнішнього ключа, яких немає в стовпці первинного ключа пов’язаної таблиці, оскільки це має бути одне зі значень, що містяться у полі первинного ключа батьківської таблиці.

Наступний приклад ілюструє створення поля з властивістю FOREIGN KEY (PersonID) під час створення таблиці Orders:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

Для завдання властивості FOREIGN для кількох стовпців слід використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
```

```
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

Для завдання властивості FOREIGN KEY для поля PersonID після створення таблиці потрібно використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Для завдання властивості FOREIGN KEY для кількох стовпців після створення таблиці потрібно використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Для скасування властивості FOREIGN KEY потрібно використати наступний синтаксис SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

20.14 Властивість діапазону значень

Властивість CHECK використовується для обмеження діапазону значень, які можна розміщувати в полі.

Завдання властивості CHECK для певного поля дозволяє обмежити діапазон даних в цьому полю певним набором значень.

Завдання властивості CHECK для таблиці дозволяє обмежувати значення в певних полях на основі значень в інших полях запису.

Наступний приклад ілюструє завдання властивості CHECK для поля Age під час створення таблиці Persons. Властивість CHECK вимагає щоб вік особи мав значення від 18 років та старше:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

Для завдання властивості CHECK для кількох стовпців слід використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
```

```
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

Для завдання властивості CHECK для поля Age після створення таблиці слід використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

Для завдання властивості CHECK для кількох стовпців після створення таблиці слід використати наступний синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandn
es');
```

Для скасування властивості CHECK слід використати наступний синтаксис SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

20.15 Властивість значення за умовчанням

Властивість DEFAULT використовується для встановлення значення за замовчуванням для певного поля. Значення за замовчуванням буде додано до всіх нових записів, якщо не вказано інше значення.

Наступний приклад інструкції SQL встановлює значення за умовчанням для поля City під час створення таблиці Persons:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

Властивість DEFAULT також можна використовувати для вставки системних значень за допомогою таких функцій, як GETDATE()

```
CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT GETDATE()
);
```

Для встановлення властивості DEFAULT для поля вже після створення таблиці використовується наступний синтаксис SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT 'Sandnes';
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Для скасування властивості DEFAULT використовується наступний синтаксис SQL:

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

SQL Server:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

20.16 Оператор створення індексів SQL

Інструкція CREATE INDEX використовується для створення індексів у таблицях.

Індекси використовуються для швидшого отримання даних із бази даних. Користувачі не можуть бачити індекси, вони просто використовуються для прискорення пошуку та сортування запитів.

Зауваження. Оновлення таблиці з індексами займає більше часу, ніж оновлення таблиці без індексів, оскільки індекси також потребують оновлення. Тому створювати індекси потрібно лише для стовпців, за якими часто здійснюватиметься пошук або сортування записів.

Наступний синтаксис інструкції CREATE INDEX призначений для створення індекса у таблиці, за якого допускаються повторювані значення:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Синтаксис CREATE UNIQUE INDEX призначений для створення унікальних індексів, за яких повторювані значення не допускаються:

CREATE UNIQUE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

Зауваження: Синтаксис для створення індексів відрізняється в різних базах даних. Тому слід перевіряти синтаксис для створення індексів у конкретній базі даних.

Наведемо приклади використання оператора CREATE INDEX

Наведений нижче оператор SQL створює індекс із назвою idx_lastname у полі LastName таблиці Persons:

CREATE INDEX idx_lastname
ON Persons (LastName);

Для створення індексу на комбінації полів потрібно вказати назви цих полів у круглих дужках, розділивши їх комами:

CREATE INDEX idx_pname
ON Persons (LastName, FirstName);

Інструкція DROP INDEX використовується для видалення індексу в таблиці.

MS Access:

DROP INDEX *index_name* ON *table_name*;

SQL Server:

DROP INDEX *table_name*.*index_name*;

DB2/Oracle:

DROP INDEX *index_name*;

MySQL:

ALTER TABLE *table_name*
DROP INDEX *index_name*;

20.17 Поля з автоінркементом

Автоматичне збільшення дозволяє автоматично генерувати унікальний номер, коли новий запис вставляється в таблицю.

Часто це поле первинного ключа, нове значення якого слід створювати автоматично кожного разу, коли вставляється новий запис.

Наведемо приклад використання цієї інструкції для **MySQL**.

Наступний оператор SQL визначає стовпець Personid як поле первинного ключа з автоматичним збільшенням у таблиці Persons.

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

MySQL використовує ключове слово AUTO_INCREMENT для виконання функції автоматичного збільшення. За замовчуванням початкове значення AUTO_INCREMENTдорівнює 1, і воно буде збільшуватися на 1 для кожного нового запису.

Для того, щоб послідовність AUTO_INCREMENT починалася з іншого значення слід скористатися наступним оператором SQL:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Для додавання нового запису у таблицю Persons вже не потрібно буде вказувати значення для стовпця Personid: унікальне значення буде додано автоматично:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

Наведена інструкція SQL під час додавання нового запису у таблицю полю Personid автоматичне привласнює унікальне значення. В поле FirstName буде встановлено значення 'Lars', а в поле LastName буде встановлено значення 'Monsen'.

Наведемо приклад використання цієї інструкції для **SQL Server**

```
CREATE TABLE Persons (
    Personid int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MS SQL Server використовує ключове слово IDENTITY для виконання функції автоматичного збільшення.

У наведеному вище прикладі початкове значення IDENTITY дорівнює 1, і воно буде збільшуватися на 1 для кожного нового запису.

Для того, щоб указати, що значення поля Personid мають починатися зі значення 10 і збільшуватися на 5 оператор повинен бути наступним: IDENTITY(10,5).

Тоді під час додавання нового запису у таблицю Persons вже не потрібно буде вказувати значення для стовпця Personid, йому буде присвласнено унікальне значення:

```
INSERT INTO Persons (FirstName,LastName)
```

Наведемо приклад використання цієї інструкції для **Access**

Наступний оператор SQL визначає поле Personid як поле первинного ключа з автоматичним збільшенням:

```
CREATE TABLE Persons (
    Personid AUTOINCREMENT PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MS Access використовує ключове слово AUTOINCREMENT для виконання функції автоматичного збільшення.

За замовчуванням початкове значення AUTOINCREMENT дорівнює 1, і воно буде збільшуватися на 1 для кожного нового запису.

Для того, щоб указати, що значення поля Personid мають починатися зі значення 10 і збільшуватися на 5 слід додати інструкцію AUTOINCREMENT(10,5).

Тоді під час додавання нового запису у таблицю Persons вже не потрібно буде вказувати значення для стовпця Personid, йому буде присвласнено унікальне значення:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

Наведемо приклад використання цієї інструкції для **Oracle**

Для використання властивості автоматичного збільшення значення в Oracle потрібно створити поле автоінкременту з об’єктом послідовності, який генерує послідовність чисел, за допомогою оператора CREATE SEQUENCE за наступного синтаксису:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

Наведений вище код створює об’єкт послідовності під назвою seq_person, який починається з 1 і буде збільшуватися на 1. Він також кешує до 10 значень для продуктивності. Опція кешу визначає, скільки значень послідовності буде збережено в пам’яті для швидшого доступу.

Щоб вставити новий запис у таблицю Persons доведеться скористатися функцією nextval, яка отримує наступне значення з послідовності seq_person:

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Наведена вище інструкція SQL додає новий запис у таблицю Persons, а полю Personid буде привласнено тнаступний номер із послідовності seq_person.

20.17 Робота з даними типу дата в SQL

Найскладніше під час роботи з датами – переконатися, що формат дати, яку ви намагаєтеся додати, відповідає формату поля дати в базі даних.

MySQL поставляється з такими типами даних для зберігання дати або значення дати/часу в базі даних:

DATE- формат РРРР-ММ-ДД

DATETIME- формат: РРРР-ММ-ДД ГГ:МІ:СС

TIMESTAMP- формат: РРРР-ММ-ДД ГГ:МІ:СС

YEAR- формат РРРР або РР

SQL Server поставляється з такими типами даних для зберігання дати або значення дати/часу в базі даних:

DATE- формат РРРР-ММ-ДД

DATETIME- формат: РРРР-ММ-ДД ГГ:МІ:СС

SMALLDATETIME- формат: РРРР-ММ-ДД ГГ:МІ:СС

TIMESTAMP- формат: унікальний номер

Зауваження. Типи дати обирається для поля під час створення нової таблиці у базі даних.

Наприклад є таблиця замовлень з полем OrderDate, яке має тип дати і містить дату замовлення.

Таблиця замовлень

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09

3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Потрібно вибрати записи, в яких поле OrderDate має значення "2008-11-11". Для цього слід використати наступну інструкцію SELECT:

SELECT * FROM Orders WHERE OrderDate='2008-11-11'

Набір результатів буде виглядати наступним чином:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Дві дати можна легко порівняти, якщо немає компонента часу.

Тепер припустімо, що таблиця «Замовлення» містить поле OrderDate, яке має тип дати і містить дату та час замовлення.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Тепер наведена вище інструкція SELECT:

SELECT * FROM Orders WHERE OrderDate='2008-11-11'

не враховує компоненти часу і не видає потрібного результату. Тому щоб запити були простими та легкими для обслуговування, не слід використовувати компоненти часу у датах без зайвої необхідності.

20.18 Створення представлення в SQL

У SQL представлення – це віртуальна таблиця, заснована на наборі результатів оператора SQL.

Представлення містить рядки та стовпці, як і справжня таблиця. Поля в представленні — це поля з однієї чи кількох реальних таблиць у базі даних.

Представлення створюється командою CREATE VIEW .

CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

Зауваження. Представлення завжди відображає актуальні дані. Механізм бази даних відтворює представлення кожного разу, коли користувач запитує його.

Наведемо приклади використання SQL команди CREATE VIEW

Наступний синтаксис SQL створює представлення, яке показує всіх клієнтів з Бразилії:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Переглянути це представлення можна наступним чином:

```
SELECT * FROM [Brazil Customers];
```

Наступний SQL створює подання, яке вибирає кожен продукт у таблиці Products з ціною, вищою за середню:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

Переглянути це представлення можна наступним чином:

```
SELECT * FROM [Products Above Average Price];
```

Представлення можна *оновити* за допомогою команди CREATE OR REPLACE VIEW.

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Наступний синтаксис SQL додає стовпець «Місто» до представлення Brazil Customers:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

Для видалення представлення використовується команда DROP VIEW.

```
DROP VIEW view_name;
```

Наступний синтаксис SQL видаляє представлення Brazil Customers:

```
DROP VIEW [Brazil Customers];
```

20.19 Ін'єкції SQL

SQL-ін’єкція – це техніка впровадження коду, яка може знищити вашу базу даних. SQL-ін’єкція є одним із найпоширеніших методів веб-злому.

SQL-ін’єкція – це розміщення зловмисного коду в операторах SQL за допомогою введення веб-сторінки.

SQL-ін’єкція зазвичай відбувається, коли ви просите користувача ввести дані, (ім’я користувача, ідентифікатор користувача), і замість даних користувач вводить оператор SQL, який
ви несвідомо запускаєте у своїй базі даних.

Подивіться на наступний приклад, у якому створюється оператор SELECT шляхом додавання змінної (txtUserId) до рядка вибору. Змінна отримується за введення користувача (getRequestString):

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

SQL-ін’єкція на основі 1=1 завжди вірна

Подивіться на приклад вище ще раз. Початковою метою коду було створити оператор SQL для вибору користувача з заданим ідентифікатором користувача.

Якщо ніщо не заважає користувачеві ввести «неправильне» значення, користувач може ввести якесь «розумне» значення, наприклад:

Ідентифікатор користувача: 105 OR 1=1

Тоді оператор SQL виглядатиме наступним чином:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

Наведений вище SQL дійсний і повертатиме ВСІ рядки з таблиці Users, оскільки АБО 1=1 завжди ІСТИНА.

Чи виглядає наведений вище приклад небезпечним? Що робити, якщо таблиця Users містить імена та паролі?

Наведений вище оператор SQL майже такий самий, як цей:

```
SELECT UserId, Name, Password FROM Users WHERE UserId
= 105 or 1=1;
```

Таким чином, хакер може отримати доступ до всіх імен користувачів і паролів у базі даних, просто вставивши 105 АБО 1=1 у поле введення.

SQL-ін’єкція на основі ""="" завжди вірна

Ось приклад входу користувача на веб-сайт:

Ім'я користувача:

John Doe

Пароль:

myPass

приклад

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ='" + uName + "' AND Pass ='" + uPass + ""
```

Результат

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

Хакер може отримати доступ до імен користувачів і паролів у базі даних, просто вставивши " АБО ""=" у текстове поле імені користувача або пароля:

Ім'я користувача:

" or ""="

Пароль:

" or ""="

Код на сервері створить наступний оператор SQL:

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

Наведений вище SQL є дійсним і поверне всі рядки з таблиці Users, оскільки АБО ""="" завжди має значення TRUE.

20.20 Ін’єкції SQL на основі пакетних інструкцій SQL

Більшість баз даних підтримують пакетні інструкції SQL.

Пакет інструкцій SQL — це група з двох або більше інструкцій SQL, розділених крапкою з комою.

Наведений нижче оператор SQL поверне всі рядки з таблиці Users, а потім видалить таблицю Suppliers.

SELECT * FROM Users; DROP TABLE Suppliers

Повернемося до прикладу введення ідентифікатора користувача:

txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

Та сформулюємо наступний рядок введення:

Ідентифікатор користувача: 105; DROP TABLE Suppliers

Тоді оператор SQL виглядатиме наступним чином:

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

20.21 Використання параметрів SQL для захисту

Щоб захистити веб-сайт від впливу SQL ін’єкцій потрібно використовувати параметри SQL.

Параметри SQL – це значення, які контрольованим чином додаються до запиту SQL під час виконання.

Наприклад **ASP.NET Razor**

txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);

Зверніть увагу, що параметри представлені в операторі SQL маркером @.

Механізм SQL перевіряє кожен параметр, щоб переконатися втому, що його значення є правильним для відповідного поля та розглядається буквально, а не як частина SQL інструкції, яку потрібно виконати.

Інший приклад

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City)
Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

У наступних прикладах показано, як створювати параметризовані запити деякими поширеними веб-мовами.

Оператор SELECT в ASP.NET:

```
txtUserId = getRequestString("UserId");
sql = "SELECT * FROM Customers WHERE CustomerId = @0";
command = new SqlCommand(sql);
command.Parameters.AddWithValue("@0",txtUserId);
command.ExecuteReader();
```

Оператор INSERT INTO в ASP.NET:

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City)
Values(@0,@1,@2)";
command = new SqlCommand(txtSQL);
command.Parameters.AddWithValue("@0",txtNam);
command.Parameters.AddWithValue("@1",txtAdd);
command.Parameters.AddWithValue("@2",txtCit);
command.ExecuteNonQuery();
```

Оператор INSERT INTO у PHP:

```
$stmt = $dbh->prepare("INSERT INTO Customers
(CustomerName,Address,City)
VALUES (:nam, :add, :cit)");
$stmt->bindParam(':nam', $txtNam);
```

```
$stmt->bindParam(':add', $txtAdd);
$stmt->bindParam(':cit', $txtCit);
$stmt->execute();
```

20.22 Хостинг SQL

Для того, щоб веб-сайт мав змогу зберігати та отримувати дані з бази даних, ваш веб-сервер повинен мати доступ до системи баз даних, яка використовує мову SQL.

Якщо ваш веб-сервер розміщено в Інтернет-провайдера (ISP), вам доведеться шукати варіанти хостингу SQL.

Найпоширенішими базами даних для розміщення SQL є MS SQL Server, Oracle, MySQL і MS Access.

20.23 Отримання інформації про базу даних

У випадку, якщо потрібно дізнатися назву бази даних або таблиці, або структуру певної таблиці (наприклад, з яких стовпців вона складається), MySQL дозволяє вирішити цю задачу за допомогою команд, які надають інформацію про бази даних і таблиці, які вона підтримує.

Виконання команди SHOW DATABASE надає список баз даних, які управляються даним сервером. Для визначення того, яку базу даних вибрано в даний момент слід скористатися функцією DATABASE().

Якщо ще не вибрано жодної бази даних, результат буде NULL.

Для визначення, які саме таблиці містить дана база даних (наприклад, якщо слід вточнити назву таблиці), слід скористатися запитом SHOW TABLES.

Назва стовпчика, яку буде надано у відповідь на цей запит завжди буде мати синтаксис:

```
Tables_in_db_name,
```

де db_name — назва бази даних.

Для визначення структури таблиці слід скористатися командою DESCRIBE, результатом якої є опис полів таблиці. Такий опис є таблицею, яка містить:

Field показує ім’я поля,

Type показує тип поля,

NULL показує чи може поле містити значення NULL,

Key показує чи є поле індексом або ключем,

Default показує яке значення приймає дане поле за умовчанням,

Extra показує додаткову інформацію про поле, наприклад, чи має поле властивість AUTO_INCREMENT.

У разі, якщо таблиця має індексні поля їх перегляд можна здійснити за допомогою команди SHOW INDEX FROM tbl_name.

20.24 Створення представлень

https://www.w3schools.com/mysql/mysql_view.asp

В базах даних на персональних комп’ютерах представлення застосовуються для зручності та спрощення запитів до бази даних. В промислових базах даних представлення відіграє роль під час створення структури бази даних для кожного користувача та забезпечення її безпеки.

Основні переваги представлень наступні:

• Безпека. Кожному користувачеві можна надати доступ лише до певної кількості представлень, які містять лише ту інформацію, яку має знати даний користувач. Таким чином можна здійснити обмеження доступа користувачів до інформації, яка зберігається в базі.

• Простота запитів. За допомогою представлень можна отримати дані з кількох таблиць та подати їх як одну таблицю, перетворюючи таким

чином запит до багатьох таблиць в однотабличний запти до представлення.

• Простота структури. За допомогою представлень для кожного користувача можна створити власну структуру бази даних та визначити її як множину віртуальних таблиць, які досяжні для користувача.

• Захист від змін. Представлення може повертати несуперечливий та незмінний образ структури бази даних, навіть якщо початкові таблиці поділяються, реструктуруються або перейменовуються. Проте за зміни назв початкових таблиць або полів, що входять до його складу, визначення представлення має бути обновлено.

• Цілісність даних. Якщо доступ до даних або введення даних відбувається за допомогою представлення, СУБД може автоматично перевіряти виконання певних умов цілісності.

Представлення мають наступні недоліки

• Продуктивність. Представлення створює лише відімість існування відповідної таблиці, тому СУБД доводиться перетворювати запит до представлення у запит до початкових таблиць. Якщо представлення відбиває багатотабличний запит, то простий запит до представлення стає складним об’єднанням і тому на його виконання може бути затрачено багато часу. Проте це не завжди пов’язано із запитом що звертається до представлення, будь-який погано сконструйований запит може створювати проблеми продиктивності.

• Управляємість. Представлення, так само як і всі інші об’єкти баз даних, мають підлягати управлінню. Якщо розробники та користувачі баз даних зможуть на власний розсуд створювати будь-які представлення, то робота адміністратора бази даних стане значно складнішою. Це особливо видно у випадку, коли створюються представлення, в основі яких є інші представлення, які, в свою чергу, можуть бути сконструйовані на інших представленнях. Таким чином, чим більше рівнів між базовими таблицями та представленнями, тим складніше вирішувати проблеми з представленнями, які виникають в такій системі.

• Обмеження на оновлення. Коли користувач намагається обновити рядки представлення, СУБД має перетворити запит в запит на обновлення рядків початкових таблиць. Такий процес є можливим для простих представлень; складніші представлення обновлювати не можна, вони є досяжними лише для вибірки.