МЕТОДИЧНІ ВКАЗІВКИ

для виконання лабораторних робіт

з дисципліни

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Частина 2

2017

Методичні вказівки для самостійної роботи та виконання лабораторних робіт з дисципліни «Організація баз даних та знань» для студентів III курсу денної форми навчання. Спеціальність 122 – «Комп'ютерні науки» / Козловська В.П. – Одеса, ОДЕКУ, 2017. – 45 с.

ЗМІСТ

Лабораторна робота №5 Розробка бази даних 4
Основні теоретичні відомості 4
Завдання на виконання лабораторної роботи 18
Варіанти баз даних 23
Варіанти завдань створення представлень 28
Лабораторна робота №6. Збережені процедури в базі даних 33
Перелік тем лекційного курсу 33
Основні теоретичні відомості 33
Варіанти завдань створення збережених процедур 43
Література 45

Лабораторна робота №5 Розробка бази даних

Мета роботи:

1. Отримання практичних навичок створення бази даних у СКБД MS SQL Server 2000.
2. Отримання практичних навичок створення представлень бази даних у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

1. Типи даних в мові Transact-SQL.
2. Реляційні ключі.
3. Обмеження цілісності.
4. Призначення представлень (view) бази даних
5. Команда створення представлення бази даних

Основні теоретичні відомості

Загальні відомості про архітектуру баз даних MS SQL Server 2000

Базу даних SQL Server 2000 можна розглядати із двох сторін: фізичної й логічної.

Фізична структура БД описує кількість файлів даних і журналу транзакцій, з яких складається БД, їх початковий і поточний розмір, положення на диску, ім'я, розширення й деякі інші параметри. Для користувачів, що працюють із базою даних, у переважній більшості випадків її фізична структура не має значення.

На логічному рівні розглядаються об'єкти, які можна створювати в базі даних, а також різні властивості, які впливають на роботу сервера з базою даних. Під об'єктами тут розуміється не тільки власне об'єкт, яким є таблиця, представлення, збережена процедура й т.д., але також і користувачі, ролі, повнотекстові каталоги. До логічного рівня відносяться й права доступу користувачів і ролей бази даних до створених в ній об'єктів.

Наведемо список власне об'єктів бази даних, які служать для зберігання й обробки інформації:

– **Таблиці** (**tables**). Це єдиний об'єкт бази даних, призначений для зберігання даних користувачів (не системних даних).

– **Представлення (views).** Є *віртуальними таблицями* (virtual tables), які відображають дані, що зберігаються в інших таблицях. Для користувача ж представлення багато в чому нагадують таблиці.

 Індекси (indexes). Об'єкти цього типу призначені для підвищення продуктивності роботи сервера при пошуку потрібних даних у таблицях і представленнях, що досягається шляхом зберігання в упорядкованому стані даних одного або більше стовпців таблиці або представлення. Таким чином, індекси не можуть існувати самі по собі.

 Ключі (keys). Є одним з типів обмеження цілісності. Однак вони грають досить важливу роль у БД і тому розглядаються як окремі об'єкти. Проте, реалізуються вони так само, як і інші обмеження цілісності, які зв'язуються з таблицями.

 Умовчання (defaults). Цей тип об'єктів описує значення, які присвоюються стовпцям таблиці, якщо при додаванні рядка явно не було зазначене значення для відповідного стовпця.

 Правила (rules). Є логічною умовою, що обмежує діапазон можливих значень для стовпця таблиці або обумовленого користувачем типу даних.

 Обмеження цілісності (constraints). Спеціальні керуючі конструкції, що обмежують діапазон можливих значень у стовпці таблиці. Таким чином, обмеження цілісності виявляються нерозривними з таблицями.

Іменування об’єктів

При створенні об'єктів висувається ряд вимог до їхніх імен. Розглянемо ці вимоги.

- Максимально припустима довжина імені об'єкта дорівнює 128 символам. Довжина імені тимчасових таблиць обмежена 116 символами.
- При виборі імені об'єкта варто впевнитися, що воно не є зарезервованим словом, що використовується самим SQL Server 2000.
- Як перший символ імені об'єкта допускається застосування тільки символів національного або латинського алфавітів, а також символу _ (підкреслення). Тобто ім'я об'єкта не може починатися із цифри, символів !, $ і т.д.
- Другий і будь-який з наступних символів імені об'єкта може включати будь-які символи, визначені стандартом Unicode Standard 2.0 — символи національних алфавітів, десяткові цифри й символи @, #, $ і _.
- Не допускається використання в будь-якому місці імені об'єкта пробілів, круглих, квадратних і фігурних дужок, а також символів !, %, ^, &, ~, -, . (крапка), , (кома), \, * і '.

Крім стандартних ідентифікаторів в SQL Server 2000 існують ще й так звані *обмежені ідентифікатори* (Delimited Identifiers). Обмежені ідентифікатори дозволяють обходити деякі з описаних вище правил іменування об'єктів. Для цього в SQL Server 2000 призначені спеціальні обмежувачі – або квадратні дужки, або подвійні лапки. Імена об'єктів, укладені у квадратні дужки або подвійні лапки, і називаються обмеженими ідентифікаторами.

Коли ім'я об'єкта міститься в обмежувачі, у ньому можуть використовуватися будь-які символи, включаючи дужки, пробіли, спеціальні символи, а також зарезервовані слова. Крім того, першим символом імені об'єкта може бути будь-який символ. Однак залишаються вимоги до

унікальності імені об'єкта.

Подвійні лапки можуть бути й аналогом одинарних лапок, тобто обмежувати символьні рядки. Для визначення ролі подвійних лапок використовується команда:

```
SET QUOTED_IDENTIFIER   {ON | OFF}
```

При виконанні команди з параметром ON подвійні лапки будуть інтерпретуватися як квадратні дужки. Якщо ж використовується аргумент OFF, то укладені в подвійні лапки символи стануть сприйматися як звичайний рядок символів, а подвійні лапки будуть інтерпретуватися як одинарні лапки.

В цій команді (як і в усіх наступних) у фігурних дужках записується варіантна частина команди. Тобто, при завданні команди самих фігурних дужок не буде. Замість них повинен бути записаний один з варіантних параметрів, які перелічені в фігурних дужках. В прикладах синтаксису команд варіанти параметрів відокремлюються один від одного символом "|". Тобто наведений синтаксис команди потрібно читати так:

```
SET QUOTED_IDENTIFIER  ON
```

або

```
SET QUOTED_IDENTIFIER  OFF
```

В синтаксису команд також використовуються квадратні дужки, в які записуються необов'язкові параметри команди (при завданні команди квадратних дужок не буде, а необов'язкові параметри можуть бути або не бути).

Доступ до об'єктів

Кожний об'єкт бази даних SQL Server 2000 повинен мати свого *власника* (owner). Власник об'єкта має *повний контроль* (full control) над своїм об'єктом.

Крім того, що об'єкт належить конкретному користувачеві, він ще й перебуває в певній базі даних. База даних, у свою чергу, розміщається на якомусь сервері. На основі цих відносин складається *повне* ім'я (complete name) або, як його ще називають, *повністю визначене ім'я* (full qualified name) об'єкта. Повне ім'я об'єкта записується у вигляді:

```
[[[server.][database].][owner__name].]object_name
```

На чолі дерева об'єктів перебуває сервер (`server` – ім'я серверу). Наступний рівень ієрархії – це база даних (`database` – назва БД). Потім вказується користувач, якому належить об'єкт (`owner__name` – ідентифікатор користувача), а вже потім ім'я самого об'єкту (`object_name`).

Створення БД в MS SQL Server 2000

Створення бази даних

Створення бази даних засобами Transact-SQL надає адміністраторові максимальні можливості. Для створення бази даних існує наступна команда Transact-SQL:

```
CREATE DATABASE database_name
[ON
[ <filespec> [,.n]]
[, <filegroup>  [,.n]]
]
[LOG ON  {  <filespec>   [   ,...n ]}]
[COLLATE collation_name]
[FOR LOAD   |   FOR ATTACH]
```

Розглянемо призначення основних параметрів команди:

database_name

Цей параметр визначає ім'я, що буде привласнене базі даних. Ім'я бази даних вказується у форматі Unicode і може бути довжиною до 128 символів. Це єдиний обов'язковий параметр команди CREATE DATABASE. Однак для створення бази даних необхідно визначити імена файлів даних і журналу транзакцій, з яких буде складатися база даних. Коли ж команда CREATE DATABASE виконується тільки із вказівкою параметра *database_name*, те сервер створить один файл даних і один файл журналу транзакцій. Генерування імен цих файлів буде виконуватися на основі імені бази даних, до якого буде доданий суфікс `_Data` для файла даних і суфікс `_Log` для файлу журналу транзакцій. Оскільки довжина імен файлів також обмежується 128 символами, то довжина імені бази даних у випадку вказівки тільки параметра *database_name* (точніше, не вказівки імен файлів) обмежується 123 символами.

ON

Присутність цього ключового слова свідчить про те, що далі треба визначення файлів даних, а також груп файлів, з яких буде складатися база даних. Якщо параметр ON опускається, то для бази даних буде створений єдиний файл даних. За замовчуванням розмір цього файла дорівнює 768 Кбайт (0,75 Мбайт). Однак цей розмір можна змінити шляхом внесення відповідних змін у базу даних `Model`.

LOG ON

Після цього ключового слова вказується набір файлів, з яких буде складатися журнал транзакцій бази даних. Якщо наводиться більше одного файлу, то вони повинні бути розділені комою. Як видно із синтаксису, параметр LOG ON може не вказуватися. У цьому випадку журнал транзакцій буде складатися з єдиного файла, розмір якого буде

становити 25% від загального обсягу всіх файлів даних, але не менш 512 Кбайт, і буде мати ім'я, що генерується на основі імені бази даних, до якого додається суфікс Log.

COLLATE *collation_name*

За допомогою цього параметра вказується зіставлення, що буде мати база даних. Зіставлення являється свого роду набором правил, що визначають алгоритми обробки строкових даних:

- порядок сортування для даних не Unicode (char, varchar і text);
- порядок сортування для даних Unicode (nchar, nvarchar і ntext);
- кодова сторінка, що використовується для зберігання даних не Unicode.

Якщо параметр COLLATE опускається, то буде використовуватися зіставлення, визначене на рівні сервера при установці SQL Server 2000.

FOR LOAD

Цей параметр використовується для сумісності з попередніми версіями SQL Server (до SQL Server 7.0).

FOR ATTACH

Параметр FOR ATTACH дозволяє не створювати нової бази даних у повному значенні, а виконати приєднання існуючої бази даних.

Хоча розглянутий варіант команди CREATE DATABASE порівняно невеликий, існує можливість ще більше зменшити розмір коду команд, прийнявши для всіх параметрів, крім імені бази даних, значення по умовчанню:

```
CREATE DATABASE mydatabase
```

Видалення бази даних

Для виконання операції видалення бази даних використовується наступна команда Transact-SQL:

```
DROP DATABASE database_name [ ,...n ].
```

За допомогою єдиного параметра команди вказується ім'я бази даних, яку необхідно видалити. Однак за одну операцію можна видалити множину баз даних, просто перелічивши їхні імена через кому. При видаленні бази даних відбувається видалення рядка таблиці sysdatabases, що описує відповідну базу даних. Також виконується й фізичне видалення всіх файлів, з яких складалася база даних. Проте, маючи резервну копію бази даних, згодом можна відновити її знову

Команда DROP DATABASE повинна виконатися в контексті бази даних Master. Не можна видалити системні бази даних Model, Tempdb, Msdb і Master.

Робота з таблицями

У будь-якій системі керування базами дані таблиці відіграють величезну роль. Таблиці використовуються для зберігання всієї інформації, що користувачі внесли в базу даних. З погляду користувача таблиця являє собою двовимірний масив, кожний рядок якого є екземпляром описуваного в таблиці типу об'єкта. Стовпці масиву являють собою атрибути об'єкту. На перетинанні конкретного рядка й конкретного стовпця знаходиться атрибут конкретного об'єкта. Розглянемо це більш докладно.

Перш ніж почнеться робота з таблицями, їх необхідно створити. Під час цієї операції користувач визначає ім'я таблиці, імена стовпців, тип збережених у них даних, значення за замовчуванням, можливість зберігання невизначених значень, первинний і зовнішній ключі й деякі інші властивості.

Перш ніж приступитися до безпосереднього створення таблиці, необхідно вирішити, які стовпці повинні бути визначені в таблиці, як вона сама буде пов'язана з іншими таблицями, які дані передбачається зберігати в стовпцях таблиці й т.д. Тобто спочатку потрібно розробити логічну модель таблиці, яка б органічно вписувалася в загальну логічну модель бази даних. Питання проектування баз даних вивчаються раніше в курсах «Організація баз даних і знань» і «Системний аналіз і проектування інформаційних систем». Будемо вважати, що користувач має добре сплановану логічну модель таблиці, і залишилося тільки реалізувати її фізично. Розглянемо, як це реалізується в MS SQL Server 2000.

Створення таблиць

При створенні таблиць користувач може для стовпців крім завдання базових властивостей, таких як ім'я, тип даних, розмір і точність, указати обмеження цілісності.

Обмеження цілісності (constraints) — це механізм контролю значень, які можуть зберігатися в полях рядка таблиці. В SQL Server 2000 підтримуються наступні обмеження цілісності:

- **Check** — за допомогою логічних умов накладає обмеження на значення, які можуть зберігатися в стовпці;
- **Null** — задає можливість зберігання невизначених значень;
- **Default** — призначає значення по умовчанню;
- **Unique** — гарантує унікальність значень у стовпці;
- **Primary Key** — визначає первинний ключ;
- **Foreign Key** — визначає зовнішній ключ;
- **No Action** — пропонує не виконувати в залежній таблиці ніяких дій при видаленні або відновленні рядків у головній таблиці;
- **Cascade** — у цьому випадку буде здійснюватися каскадна зміна значень у залежній таблиці при внесенні змін у головну таблицю.

Однієї з основних характеристик стовпця є *тип даних* (data type). Тип даних визначає діапазон значень, які можна буде зберігати в стовпці. Основні типи даних перелічені в таблиці 1.

Таблиця 1 – Список типів даних, що використовуються для стовпців

Тип даних	**Короткий опис**
bigint	Цілочисельний тип даних, що займає 8 байт
binary	Двійкові дані фіксованої довжини до 8000 байт
bit	Один біт, приймає значення або 0, або 1
char	Символьні дані не Unicode фіксованої довжини до 8000 символів
datetime	Дата й час високої точності (8-байтовий) від 1 січня 1753 року до 31 грудня 9999 року, час визначається з точністю до 3.33 мілісекунд
decimal	Нецілочисельний тип даних фіксованої точності для якого вказується загальна кількість цифр в числі, та скільки з них цифр дробової частини, наприклад, decimal (7,2)
float	Нецілочисельний тип даних приблизної точності з діапазоном значень від 1.79E + 308 до 1.79E + 308
image	Двійкові дані довжиною до 2 Гбайт
int	Цілочисельний тип даних, що займає 4 байти
money	Грошовий тип даних високої точності (8-байтовий)
nchar	Символьні дані Unicode фіксованої довжини до 4000 символів
ntext	Текстові дані Unicode довжиною до 1 Гбайта
nvarchar	Символьні дані Unicode змінної довжини до 4000 символів
numeric	Нецілочисельний тип даних фіксованої точності (аналогічний decimal)
real	Нецілочисельний тип даних приблизної точності з діапазоном значень від –3.40E + 38 до 3.40E + 38; займає 4 байти
smalldatetime	Дата й час низької точності (4-байтовий) від 1 січня 1 1900 року до 6 червня 2079 року, час визначається з точністю до половини хвилини. В smalldatetime значення секунд не більше 29.998 відкидаються; значення секунд від 29.999 – додає ще одну хвилину до часу
smallint	Цілочисельний тип даних, що займає 2 байти

Тип даних	Короткий опис
smallmoney	Грошовий тип даних низкою точності (4-байтовий)
text	Текстові дані не Unicode довжиною до 2 Гбайт
timestamp	Часовий штамп або версія рядка
tinyint	Цілочисельний тип даних, що займає 1 байт
varbinary	Двійкові дані змінної довжини до 8000 байт
varchar	Символьні дані не Unicode змінної довжини до 8000 символів

Часто набір полів таблиці є незручним для використання його як первинний ключ. Наприклад, у таблиці з описом людини в якості первинного ключа можна вибрати стовпець із номером паспорта і його серією. Однак іноді по тих або інших причинах людина міняє паспорт. У цьому випадку необхідно виправити значення не тільки в головній таблиці, але й у залежних таблицях. Ці й деякі інші причини змушують багатьох розроблювачів створювати в таблиці додатковий стовпець, єдине призначення якого – служити ідентифікаційним номером рядка в таблиці. Для цих полів зручно користуватися автоматичною нумерацією – під час додавання рядка в таблицю для поля з автонумерацією значення не вводиться, а SQL Server сам визначає наступне значення в стовпці.

Щоб дозволити поле таблиці для автоматичної нумерації, для нього необхідно встановити властивість IDENTITY (ID entity, ідентифікаційний номер сутності). При цьому користувач повинен задати початкове значення й крок приросту – якщо їх не задавати, то і для початкового значення і для кроку змінення призначається значення 1. Перший рядок, що вставляє в таблицю, буде мати значення, зазначене як початкове. У кожній таблиці тільки для одного стовпця може бути встановлена властивість IDENTITY.

Створення таблиць в SQL Server 2000 виконується за допомогою команди Transact-SQL CREATE TABLE. Розглянемо докладно синтаксис і використання цієї команди.

```
CREATE TABLE
   [ database_name.[ owner ] . | owner. ] table_name
   ( { < column_definition >
      | column_name AS computed_column_expression
      | < table_constraint > ::= [ CONSTRAINT constraint_name ] }

                  | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
   )

[ ON { filegroup | DEFAULT } ]
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]
```

Параметри команди:

database_name

Ім'я бази даних, у якій буде створена таблиця. Зазначена база даних повинна існувати. Якщо її ім'я не задається, то таблиця буде створена в поточній базі даних. Необхідно переконатися, що обліковий запис користувача має доступ до бази даних, у якій повинна бути створена таблиця, і що цей користувач має права на створення таблиць.

owner

Ім'я користувача бази даних, що буде вважатися власником створюваної таблиці. Власник таблиці може виконувати будь-які дії з нею, у т.ч. дозволяти доступ до цієї таблиці іншим користувачам. Якщо ім'я користувача не вказується, то власником таблиці є користувач, що створив її.

table_name

Ім'я, що буде привласнено таблиці. Для зберігання імені таблиці приділяється 256 байт, однак, тому що ім'я вказується у форматі Unicode, воно не повинне перевищувати 128 символів. Комбінація імені таблиці й імені її власника повинна бути унікальна в межах бази даних.

<*column_definition*>

Ця конструкція визначає властивості стовпця. Синтаксис цієї конструкції і її використання буде розглянуто далі.

column_name AS computed_column_expression

За допомогою цього аргументу можна створити стовпці, що *обчислюють* (computed). Значення таких стовпців обчислюються щораз заново при звертанні до них. Наприклад, якщо в таблиці є рядок, у якому існують стовпці із ціною товару (`price`) і його кількість (`count`), то стовпець із загальною ціною товару (`cost`) може бути обчислений автоматично. Для цього при створенні таблиці необхідно використати наступну конструкцію: `cost AS count*price`

У структурі таблиці зберігається тільки формула, по якій відбувається обчислення значень, тоді як самі значення не зберігаються. Тому неможлива вставка або зміна значень в полях, що обчислюють. При створенні стовпця, що обчислюється, необхідно вказати його ім'я (аргумент *column_name*) і після ключового слова AS вираз (аргумент *computed_column_expression*), по якому буде обчислюватися значення поля.

Крім того поля, що обчислюють, не дозволено змінювати, для них не можна встановлювати обмеження цілісності UNIQUE, PRIMARY KEY, FOREIGN KEY і DEFAULT.

<*table_constraint*>

За допомогою цієї конструкції визначаються обмеження цілісності рівня таблиці. Докладно синтаксис і застосування розглянутої конструкції будуть наведені далі.

[,*...n*]

Дана конструкція говорить про те, що через кому може бути зазначена множина визначень стовпців або обмежень цілісності.

Звичайні (не ті, що обчислюються) стовпці визначаються за допомогою конструкції <column_definition>, що має синтаксис:

```
< column_definition > ::= { column_name data_type }
  [ COLLATE < collation_name > ]
  [ [ DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed , increment ) [ NOT FOR REPLICATION ] ] ]
  ]
  [ ROWGUIDCOL]
  [ < column_constraint > ] [ ...n ]
```

Розглянемо призначення й використання параметрів команди.

column_name

Ім'я, що буде мати стовпець. Ім'я повинне бути унікальним у межах таблиці. В імені стовпця допускається вказівка російських символів. Якщо в імені стовпця застосовуються заборонені символи, такі як пробіл, %, * і т.д., або ім'я стовпця збігається із зарезервованими словами, то ім'я стовпця при створенні повинне бути укладене у квадратні дужки.

data_type

Після імені стовпця через пробіл вказується тип даних, що будуть мати збережені в стовпці значення. Дозволяється застосування як стандартних типів даних SQL Server 2000, так і *користувальницьких типів даних* (UDDT, User Defined Data Type).

DEFAULT *constant_expression*

За допомогою цього аргументу можна визначити значення за замовчуванням, що буде привласнюватися відповідному полю рядка, якщо при її вставці користувач явно не вказав конкретне значення. Значення за замовчуванням не може бути застосоване до стовпців з типом даних timestamp або із установленою властивістю IDENTITY. Як значення за замовчуванням можуть застосовуватися константи, системні змінні й функції, а також будь-які вирази, побудовані на їхній основі. Використання посилань на інші стовпці заборонено.

IDENTITY [(*seed* , *increment*)]

Зазначення цього аргументу пропонує створити стовпець із підтримкою автоматичної нумерації. При вставці нового рядка в таблицю SQL Server 2000 автоматично забезпечує вставку в поле IDENTITY унікального значення, що монотонно збільшується при вставці кожного нового рядка.

Властивість IDENTITY може бути встановлено тільки для стовпців з типом даних int, smallint, tinyint, decimal (p, 0) і numeric (p, 0). У межах однієї таблиці можна створити тільки один стовпець із установленою властивістю IDENTITY. По умовчанню *seed* **=1** та *increment*=1.

<column_constraint>

Ця конструкція визначає обмеження цілісності на рівні стовпця. Синтаксис і використання цієї конструкції розглянуто далі.

[...*n*]

Дана конструкція говорить про те, що для одного стовпця може бути визначена декілька обмежень цілісності, які повинні бути перераховані через пробіл.

Як видно з синтаксису команди, для опису стовпця таблиці досить визначити його ім'я та тип даних. Інші аргументи є необов'язковими.

Обмеження цілісності на рівні стовпців

Як було сказано вище, обмеження цілісності для стовпця визначаються за допомогою конструкції <column_constraint>. Для одного стовпця може бути визначена множина обмежень цілісності або не визначено жодного. Конструкція <column_constraint> має наступний синтаксис:

```
< column_constraint > ::= [ CONSTRAINT constraint_name ]
  { [ NULL | NOT NULL ]
    | [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [ WITH FILLFACTOR = fillfactor ]
      [ON {filegroup | DEFAULT} ] ]
    ]
    | [ [ FOREIGN KEY ] REFERENCES ref_table [ ( ref_column ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
      [ NOT FOR REPLICATION ]
    ]
    | CHECK [ NOT FOR REPLICATION ]
    ( logical_expression )
  }
```

Розглянемо призначення та використання основних аргументів:

CONSTRAINT *constraint_name*

Ключове слово CONSTRAINT вказує на те, що далі треба опис обмежень цілісності. За допомогою аргументу *constraint_name* вказується ім'я, що буде привласнено обмеженню цілісності. Ім'я обмеження цілісності повинне бути унікально в межах бази даних. Зазначення конструкції CONSTRAINT *constraint_name* не обов'язково.

NULL | NOT NULL

За допомогою цих опцій визначається, чи буде можливим зберігання в стовпці невизначених значень, чи ні. Для одного стовпця допускається застосування тільки одного з аргументів. При вказівці NULL зберігання невизначених значень дозволено, тоді як при вказівці NOT NULL забороняється. Якщо при створенні стовпця не було явно зазначене, буде він зберігати значення NULL чи ні, то для обмеження цілісності береться значення по умовчанню, обумовлене за допомогою команд ANSI_NULL_DFLT_ON і ANSI_NULL_DFLT_OFF.

PRIMARY KEY

При створенні стовпця із цим параметром буде створений первинний ключ. У таблиці дозволяється створювати тільки один первинний ключ, у який може бути включене більше одного стовпця.

UNIQUE

Це ключове слово забезпечує створення для стовпця обмеження цілісності UNIQUE, що забезпечує унікальність, значень. Це обмеження цілісності може бути визначене на основі більш ніж одного стовпця.

Зауваження: Для того самого стовпця не можуть бути одночасно встановлені обмеження цілісності Primary Key і Unique.

FOREIGN KEY REFERENCES *ref_table* [(*ref_column*)]

За допомогою цієї конструкції визначається зовнішній ключ таблиці. Ключові слова FOREIGN KEY можуть не використовуватися. Після ключового слова REFERENCES вказується ім'я таблиці, зі стовпцем якої буде зв'язуватися обмеження цілісності FOREIGN KEY. Це обмеження цілісності зв'язується з одним стовпцем головної таблиці. Для стовпця головної таблиці, з яким зв'язується обмеження цілісності FOREIGN KEY, повинне бути встановлене обмеження цілісності PRIMARY KEY або UNIQUE.

Ім'я таблиці, з якої зв'язується зовнішній ключ, вказується за допомогою аргументу *ref_table*. За замовчуванням зовнішній ключ зв'язується з первинним ключем. Але можна зв'язати зовнішній ключ із будь-яким іншим стовпцем. Для цього за допомогою аргументу *ref_column* вказується ім'я потрібного стовпця.

ON DELETE {CASCADE | NO ACTION}

Пропонує використовувати для відповідного зовнішнього ключа обмеження цілісності CASCADE або NO ACTION на видалення рядків у головній таблиці. Таким чином, видалення рядків у головній таблиці буде приводити до видалення відповідних рядків (при вказівці CASCADE) у створюваній таблиці або скасуванні операції видалення рядка головної таблиці (при вказівці NO ACTION).

ON UPDATE {CASCADE | NO ACTION}

Пропонує використовувати для відповідного зовнішнього ключа обмеження цілісності CASCADE або NO ACTION на зміну рядків у головній таблиці. Таким чином, зміна первинного ключа в головній таблиці буде приводити до модифікації відповідних рядків (при вказівці CASCADE) у створюваній таблиці або скасуванні операції зміни рядка головної таблиці (при вказівці NO ACTION).

Обмеження цілісності на рівні таблиці

У попередньому розділі були описані обмеження цілісності на рівні окремого стовпця. Однак іноді буває необхідно визначити деякі обмеження цілісності на рівні таблиці. Наприклад, при визначенні обмежень цілісності PRIMARY KEY або UNIQUE на рівні стовпця можна використати тільки один стовпець. Однак іноді буває необхідно визначити обмеження цілісності на основі декількох стовпців. Наприклад, в базі даних житлового фонду є таблиці «Вулиці» і «Будинки». Кожний будинок визначається парою ідентифікаторів: номер вулиці (ідентифікатор вулиці в таблиці) та номер будинку на вулиці, – таким чином первинний ключ таблиці «Будинки» буде складатися з двох атрибутів. Для визначення обмежень цілісності PRIMARY KEY (або UNIQUE), FOREIGN KEY, CHECK на основі декількох стовпців потрібно реалізувати ці обмеження цілісності на рівні таблиці.

Для визначення обмежень цілісності на рівні таблиці існує конструкція <table_constraint>, що має синтаксис:

```
< table_constraint > ::= [ CONSTRAINT constraint_name ]
   { [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      { ( column [ ASC | DESC ] [ ,...n ] ) }
      [ WITH FILLFACTOR = fillfactor ]
      [ ON { filegroup | DEFAULT } ]
   ]
   | FOREIGN KEY
      [ ( column [ ,...n ] ) ]
      REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ]
      ( search_conditions )
   }
```

Параметри команди були розглянуті вище.

Видалення таблиці з БД

Для цього використовується команда Transact-SQL:

DROP TABLE *table_name*

Тут *table_name* – ім'я таблиці, що усувається з бази даних.

Якщо первинний ключ таблиці використовується як зовнішній ключ іншої таблиці, то її буде заборонено видаляти. Наприклад, у БД `City` таблиці можна усувати у наступному порядку: спочатку таблиця `Flat`, потім `House`, потім `Street`. Тобто, спочатку видаляються підлеглі таблиці.

Додавання записів в таблиці

Для цього використовується команда Transact-SQL INSERT:

INSERT [INTO] *table_name* [**(** *column_list* **)**]
VALUES **(** { DEFAULT | NULL | *expression* } [**,***...n*] **)**

Параметри команди:

INTO

Необов'язкове ключове слово.

table_name

Ім'я таблиці, в яку додаються записи.

column_list

Перелік одного чи декількох атрибутів, для яких вводяться дані в новому записі таблиці. Імена атрибутів в переліку відокремлюються комою (як і в інших переліках). SQL Server автоматично забезпечує значення для наступних атрибутів таблиць:

- Тих, що мають властивість IDENTITY. Вираховується наступне значення.
- Тих, що мають значення за замовчуванням (DEFAULT). Використовується це задане значення.
- Тих, що можуть мати пусті значення (NULL). Використовується значення NULL.

Атрибут з властивістю IDENTITY не повинен бути у переліку *column_list*. Атрибути, що мають обмеження цілісності DEFAULT або NULL можуть бути у переліку. Тоді їм можна або надавати будь-які (припустимі для їх типу) значення, або вказати в якості значення відповідне ключове слово (DEFAULT або NULL).

Іноді виникає необхідність додавати записи з деяким заданим значенням для атрибуту, що має властивість IDENTITY. SQL Server 2000 надає цю можливість за допомогою команди SET IDENTITY_INSERT, яка змінює властивість IDENTITY_INSERT таблиці бази даних:

SET IDENTITY_INSERT [*database***.** [*owner***.**]] { *table* } { ON | OFF }

По умовчанню для всіх таблиць для властивісті IDENTITY_INSERT встановлено значення OFF; у будь-який момент часу лише для одної таблиці бази даних можна встановити значення ON для властивісті IDENTITY_INSERT. Тобто, якщо в декілька таблиць потрібно вставити записи з заданими значеннями атрибутів з властивістю IDENTITY, то потрібно по черзі надавати таблицям цю можливість.

VALUES

Ключове слово, яке передує переліку значень, що вводяться. Після нього у круглих дужках наводиться перелік значень для всіх атрибутів таблиці, крім атрибуту з властивістю IDENTITY – якщо немає переліку *column_list*, – або наводиться перелік значень для всіх перелічених у *column_list* атрибутів. Якщо немає переліку *column_list*, то значення для атрибутів вказуються в тому порядку, який відповідає опису таблиці в команді CREATE TABLE.

expression

Константа, змінна, або вираз, значення якого використовується для завдання значення атрибуту.

Завдання на виконання лабораторної роботи

Для бази даних, вказаній в варіанті завдання, треба проаналізувати сутності БД та зв'язки між ними, визначити домен та тип даних для кожного атрибуту. Визначити структуру таблиць БД, створити логічну схему БД, фізичну схему БД (створення БД командами CREATE), та заповнити таблиці вказаною кількістю записів.

Для створення таблиць бази даних потрібно визначити назви та типи даних для атрибутів усіх таблиць, та встановити обмеження цілісності: первинний ключ таблиці, зовнішній ключ, атрибути з унікальними значеннями, можливість порожніх значень.

Для визначення типів даних спочатку визначається для кожного атрибуту його домен – область припустимих значень. Для визначення типів даних для числових атрибутів треба визначити діапазон їх можливих значень.

Таким чином отримуємо наступну структуру чотирьох з 5 таблиць бази даних «Персонал підприємства», яка використовувалась у лабораторних роботах №1 - №4:

Таблиця **Otdel** (Відділ)

Атрибут	Домен	Тип даних	Призначення
`Id_otdel`	Ціле число менше 100	`tinyint`	Ідентифікатор відділу, первинний ключ
`Name_otdel`	Рядок до 30 символів	`varchar(30)`	Назва відділу

Таблиця **Post** (Посада)

Атрибут	Домен	Тип даних	Призначення
Id_post	Ціле число менше 100	tinyint	Ідентифікатор посади, первинний ключ
Name_post	Рядок до 30 символів	varchar(30)	Назва посади
Post_money	Дійсне число до 99999 с 2 цифрами дробової частини	numeric(7,2)	Зарплатня
Days	Ціле число менше 100	tinyint	Кількість днів відпустки
Count_post	Ціле число менше 100	tinyint	Кількість місць (ставок) даної посади

Таблиця **Persona** (Персона)

Атрибут	Домен	Тип даних	Призначення
Id_man	Ціле число менше 10000	smallint	Ідентифікатор працівника, первинний ключ
PassNo	Рядок з 8 символів	char(8)	Серія і номер паспорту (унікальний)
Surname	Рядок до 25 символів	varchar(25)	Прізвище працівника
Name1	Рядок до 12 символів	varchar(12)	Ім’я працівника
Name2	Рядок до 15 символів	varchar(15)	По батькові
Birthdate	Дата 20-21 століття	datetime	Дата народження
Sex	Один символ	char(1)	Стать
Education	Рядок до 15 символів	varchar(15)	Освіта
Married	Логічне (так, ні)	bit	Сімейне положення
Children	Ціле число менше 100	tinyint	Кількість дітей

Таблиця **Worker** (Працівник)

Атрибут	Домен	Тип даних	Призначення
Id_man	Ціле число менше 10000	smallint	Ідентифікатор працівника, зовнішній ключ
Id_post	Ціле число менше 100	tinyint	Ідентифікатор посади, зовнішній ключ
Id_otdel	Ціле число менше 100	tinyint	Ідентифікатор відділу, зовнішній ключ
Date_work	Дата 20-21 століття	datetime	Дата прийому на роботу
Main_work	Логічне (так, ні)	bit	Чи є основним співробітником
Stavka	Дробове число не більше одиниці с 2 цифрами дробової частини	numeric(4,2)	Частина ставки (повна ставка це stavka=1)

Для створення бази даних потрібно завантажити програму MS SQL Server 2000/2008/2012. Робота с програмою Query Analyzer для MS SQL Server 2000 описана у методичних вказівках для виконання контрольної роботи. Якщо студент встановив на свій комп'ютер SQL Server версій 2008 або 2012, то з роботою програми він знайомиться самостійно. Вказані програми мають русифіковані версії, що спрощує ознайомлення.

Після створення БД, тобто виконання команд CREATE DATABASE та CREATE TABLE, потрібно створити діаграму БД, яка є логічною моделлю реляційної БД. Приклад цієї діаграми для бази даних «Персонал підприємства» наведено на рис.1.

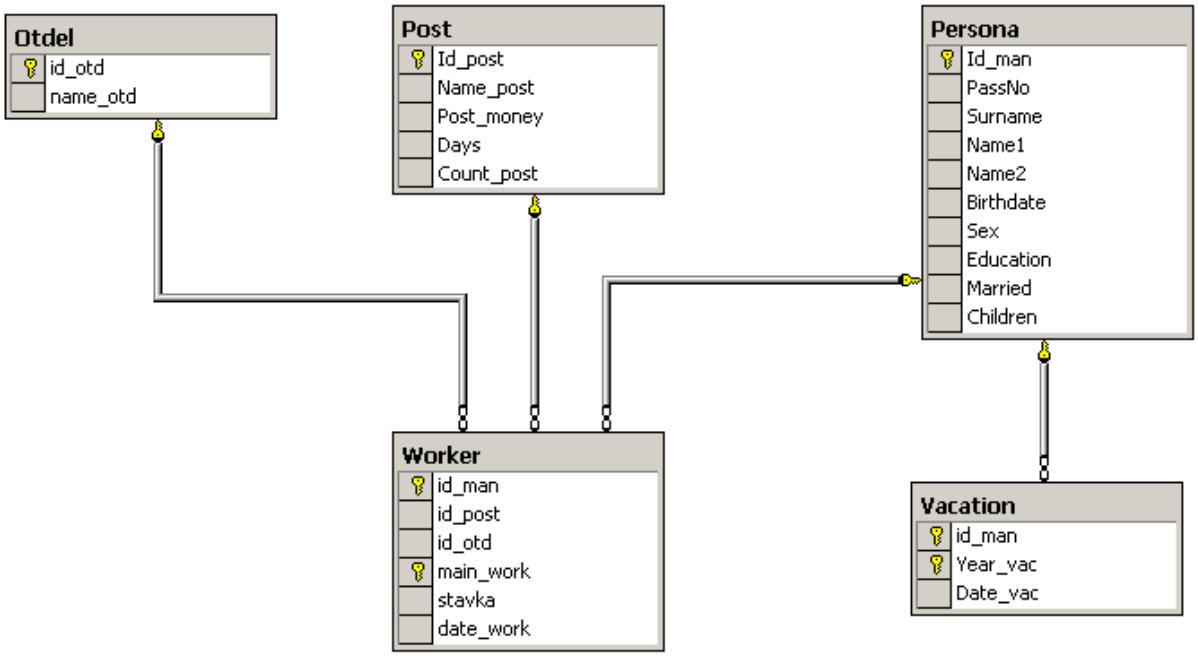

Рисунок 1 – Логічна схема БД Personal

Після створення БД потрібно внести інформацію в таблиці.

Заповнення таблиць БД відбувається тільки командами INSERT !

Таблиці з зовнішніми ключами повинні заповнюватись після заповнення таблиць з відповідними первинними ключами. Якщо всі команди заповнення бази даних записані в одному файлі, то бажано відокремити команди заповнення підлеглих таблиць командою `GO`, яка розділяє пакети команд мови Transact-SQL.

Після заповнення таблиць бази даних потрібно вивести на екран вміст кожної таблиці та зробити Screenshot. Виводити на екран вміст таблиці потрібно командою:

SELECT * FROM *TableName*

Тут *TableName* – назва таблиці.

Команда створення представлень для бази даних в СКБД MS SQL Server

Команда створення представлення має декілька опцій, не всі з яких використовуються однаково часто. Найчастіше ця команда використовується у наступному форматі:

CREATE VIEW [< *database_name* > **.**] [< *owner* > **.**] *view_name*
AS
select_statement

Аргументи:

view_name

Імя (назва) представлення.

select_statement

Це є оператор SELECT, який визначає представлення. Він може використовувати більше однієї таблиці та інших представлень. Щоб вибрати дані з об'єктів, на які посилається команда SELECT створеного представлення, необхідно мати відповідні дозволи.

Представлення не повинно бути лише простою підмножиною рядків і стовпчиків однієї конкретної таблиці. Воно може бути створене за допомогою декількох таблиць або інших представлень з опцією SELECT будь-якої складності.

Є кілька обмежень для елементів SELECT у визначенні вигляду. Команда CREATE VIEW не може:

- Включати опції COMPUTE або COMPUTE BY.
- Включати опцію ORDER BY, якщо немає опції TOP у списку вибору (*select_list*) команди SELECT.
- Включати ключове слово INTO.
- Посилатись на тимчасову таблицю або табличну змінну.

У розділі *select_statement* допустиме використання функцій.

select_statement може бути множиною команд SELECT, які відокремлюються одна від одної ключовими совами UNION або UNION ALL.

Приклади команди створення представлення.

Для таблиці Worker бази даних Personal створити представлення, що не може оновлюватись, в якому ідентифікатор працівника заміюється на його прізвище з ініціалами, ідентифікатор відділу заміюється на його назву, ідентифікатор посади заміюється на її назву, значення атрибуту main_work представляється словами «так» і «ні», дата прийому на роботу представляється у конвертованому виді у форматі «дд/мм/гггг»:

```
CREATE VIEW ViewWorker1 AS
SELECT Працівник = surname+' '+LEFT(name1,1)+ '.'+LEFT(name2,1)+ '.',
    Відділ = name_otd, Посада = name_post, Ставка = stavka,
    [Основна робота] = when main_work=1 then 'так' else 'ні' end,
    [Дата прийому на роботу] = CONVERT(char(10),date_work,103)
FROM Persona p JOIN Worker w ON p.id_man=w.id_man JOIN Post
    ON Post.id_post=w.id_post JOIN Otdel ON Otdel.id_otd=w.id_otd
```

Для таблиці Worker бази даних Personal створити представлення, що може оновлюватись та використовуватись у клієнтських програмах, в якому ідентифікатор працівника доповнюється його прізвищем з ініціалами, ідентифікатор відділу доповнюється його назвою, ідентифікатор посади доповнюється її назвою, значення атрибуту main_work доповнюється колонкою зі словами «так» і «ні», дата прийому на роботу доповнюється конвертованим виглядом у форматі «дд/мм/гггг»:

```
CREATE VIEW ViewWorker2 AS
SELECT FIO = surname+' '+LEFT(name1,1)+ '.'+LEFT(name2,1)+ '.',
    name_otd, name_post,
    MainWork = when main_work=1 then 'так' else 'ні' end,
    DateWork = CONVERT(char(10),date_work,103), w.*
FROM Persona p JOIN Worker w ON p.id_man=w.id_man JOIN Post
    ON Post.id_post=w.id_post JOIN Otdel ON Otdel.id_otd=w.id_otd
```

В представленні `ViewWorker2` збережені всі атрибути таблиці Worker (`w.*`). Коли за допомогою цього представлення дані з таблиці Worker будуть виводитись на форму програми, всі атрибути таблиці Worker, крім атрибуту stavka, будуть зроблені невидимими. Замість цих невидимих атрибутів в табличному компоненті на формі прогами будуть видимі атрибути, які їх представляють у вигляді символьних рядків:

id_man → `FIO`

id_otd → `name_otd`

id_post → `name_post`

main_work → `MainWork`

date_work → `DateWork`

Але ці атрибути доступні в програмі, хоча і є невидимими. Їх значення можна використовувати, наприклад, в запитах на додавання, оновлення, видалення записів в таблиці Worker.

В варіантах завдань студентам потрібно створити саме представлення, що можуть оновлюватись та використовуватись у клієнтських програмах.

Варіанти баз даних

Варіант 1 – БД «Деканат»

Створити базу даних деканату, в якій повинні бути виділені наступні базові сутності: **Студент**, **Викладач**, **Предмет, Відмітка**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Студент (ідентифікатор студента, № залікової книжки, прізвище, ім'я, по батькові, дата народження, рік вступу, назва факультету).

Викладач (ідентифікатор викладача, прізвище, ім'я, по батькові, степінь).

Предмет (ідентифікатор предмету, назва предмету).

Таблиця **Відмітка** містить інформацію про те, які є відмітки за чотирьохбальною шкалою (відмінно, добре, задовільно або зараховано), та нижню і верхню межу для відмітки у відсотках, наприклад, для оцінки «відмінно» ці межі є 90 та 100.

Крім базових сутностей в БД Також повинна бути похідна сутність **Залікова книжка**, яка містить інформацію про те, який студент, по якому предмету, за який семестр, якому викладачу здавав, що здавав (екзамен, залік, курсову роботу), якого числа (дата), оцінку у відсотках. Унікальність записів в таблиці **Залікова книжка** забезпечується сукупністю атрибутів: який студент, по якому предмету, за який семестр, що здавав.

В таблицю **Студент** додати 15 записів, в таблицю **Викладач** – 6, в таблицю **Предмет** – 4, в таблицю **Відмітка** – 4, в таблицю **Залікова книжка** – 30.

Варіанти 2, 3 – БД «Видавництва»

Створити бази даних видавництва, в якій повинні бути виділені наступні базові сутності: **Автор**, **Видавництво**, **Жанр книг, Книга**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Автор (ідентифікатор автора, прізвище, ім'я, по батькові, телефон, електронна адреса).

Видавництво (ідентифікатор видавництва, назва видавництва, телефон, електронна адреса).

Жанр книг (ідентифікатор жанру, назва жанру)

Книга (ідентифікатор книги, назва книги, рік видавництва, кількість сторінок, ідентифікатор жанру книги, ідентифікатор видавництва).

Крім базових сутностей в БД повинна бути похідна сутність **Книги авторів**, яка містить інформацію про те, який автор яку книгу написав. Ця сутність необхідна тому, що між таблицями **Автор** і **Книга** в загальному випадку існує зв'язок багато-до-багатьох: один автор може видати більше одної книги, і у одної книги може бути більше одного автора. Ця таблиця містить два зовнішніх ключа, а їх сукупність складає первинний ключ.

В таблицю **Автор** додати 15 записів, в таблицю **Видавництво** – 4, в таблицю **Жанр книг** – 8, в таблицю **Книга** – 15, в таблицю **Книги авторів** – 18.

Варіанти 4, 5 – БД «Авто-змагання»

Створити базу даних автомобільних змагань, в якій повинні бути виділені наступні базові сутності: **Гонщик**, **Траса**, **Змагання**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Країна (ідентифікатор країни, назва)

Гонщик (ідентифікатор гонщика, прізвище, ім’я, по батькові, назва машини, потужність, країна).

Траса (ідентифікатор траси, назва траси, країна).

Змагання (ідентифікатор змагання, назва змагання, траса, дата проведення, кількість кругів).

Крім базових сутностей в БД повинна бути похідна сутність **Результат змагань**, яка містить інформацію про те, який гонщик в яких змаганнях приймав участь, закінчив гонку чи ні, з яким часом закінчив (якщо закінчив). Ця сутність необхідна тому, що між таблицями **Гонщик** і **Змагання** в загальному випадку існує зв'язок багато-до-багатьох: один гонщик може приймати участь у різних змаганнях, у яких приймають участь багато гонщиків. Ця таблиця містить два зовнішніх ключа, а їх сукупність складає первинний ключ. Атрибут **Результат** (час закінчення гонки) може мати порожні значення.

В таблицю **Країна** додати 10 записів, в таблицю **Гонщик** – 15, в таблицю **Траса** – 4, в таблицю **Змагання** – 6, в таблицю **Результат змагань** – 30.

Варіанти 6, 7 – БД «Поліклініка»

Створити базу даних поліклініки, в якій повинні бути виділені наступні базові сутності: **Пацієнт**, **Спеціалізація лікаря**, **Лікар, Діагноз**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Пацієнт (ідентифікатор пацієнта, прізвище, ім’я, по батькові, дата народження).

Спеціалізація лікаря (ідентифікатор спеціалізації, назва (терапевт, хірург, тощо))

Лікар (ідентифікатор лікаря, прізвище, ім’я, по батькові, спеціалізація).

Діагноз (ідентифікатор діагнозу, назва (перелом, грип, ОРВІ, …)).

Крім базових сутностей в БД повинна бути похідна сутність **Діагноз пацієнта**, яка містить інформацію про те, який пацієнт у якого лікаря якого числа який діагноз отримав. Крім того потрібен атрибут **Термін непрацездатності** (в днях), який може мати порожнє значення. Ця похідна сутність необхідна тому, що між таблицями **Пацієнт**, **Лікар** і **Діагноз** в загальному випадку існує зв'язок багато-до-багатьох: один пацієнт може лікуватися у багатьох лікарів, і у одного лікаря є багато пацієнтів. Ця таблиця містить три зовнішніх ключа, а їх сукупність разом з датою прийому складає первинний ключ.

В таблицю **Пацієнт** додати 15 записів, в таблицю **Спеціалізація лікаря** – 6, в таблицю **Лікар** – 8, в таблицю **Діагноз** – 8, в таблицю **Діагноз пацієнта** – 20.

Варіанти 8, 9 – БД «Лісове господарство»

Створити базу даних лісового господарства, в якій повинні бути виділені наступні базові сутності: **Зелений масив**, **Лісник**, **Порода дерев**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Область (ідентифікатор області, назва)

Зелений масив (ідентифікатор масиву, назва масиву, площа масиву, чи є заповідником, дата останньої перевірки, область).

Лісник (ідентифікатор лісника, прізвище, ім’я, по батькові, дата народження, масив (за який відповідає)).

Порода дерев (ідентифікатор породи дерева, назва породи дерева).

Крім базових сутностей в БД повинна бути похідна сутність **Дерева масивів**, яка містить інформацію про те, яка порода дерев в якому масиві яку площу займає (відсоток). Ця сутність необхідна тому, що між таблицями **Порода дерев** і **Зелений масив** в загальному випадку існує зв'язок багато-до-багатьох: в одному масиві може рости декілька порід дерев, і одна і та сама порода дерев зустрічається в багатьох масивах. Ця таблиця містить два зовнішніх ключа, а їх сукупність складає первинний ключ.

В таблицю **Область** додати 6 записів, в таблицю **Зелений масив** – 8, в таблицю **Лісник** – 8, в таблицю **Порода дерев** – 12, в таблицю **Дерева масивів** – 30.

Варіант 10 – БД «Екскурсії містом»

Створити базу даних екскурсій, в якій повинні бути виділені наступні базові сутності: **Вулиця, Тип визначної пам'ятки, Визначні пам'ятки, Екскурсія**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Вулиця (ідентифікатор вулиці, назва вулиці).

Тип визначної пам'ятки (ідентифікатор типу визначної пам'ятки, назва типу визначної пам'ятки (монумент, музей, театр, пам'ятник архітектури)).

Визначні пам'ятки (ідентифікатор визначної пам'ятки, тип, назва визначної пам'ятки, вулиця, номер будинку).

Екскурсія (ідентифікатор екскурсії, назва екскурсії, дата проходження)

Крім базових сутностей в БД повинна бути похідна сутність **Маршрут**, яка містить інформацію про те, в якій екскурсії в якій черговості яка визначна пам'ятка відвідується. Ця сутність необхідна тому, що між таблицями **Екскурсія** і **Визначні пам'ятки** в загальному випадку існує зв'язок багато-до-багатьох: одна екскурсія може містити відвідини декількох визначних пам'яток, і одна й та сама пам'ятка може включатися в декілька екскурсій. Ця таблиця містить два зовнішніх ключа, а їх сукупність є унікальним сполученням, первинний ключ таблиці складає сполучення ідентифікатора екскурсії та черги пам'ятки (номеру в маршруті).

В таблицю **Тип визначної пам'ятки** додати 4 записи, в таблицю **Вулиця** – 10, в таблицю **Визначні пам'ятки** – 10, в таблицю **Екскурсія** – 8, в таблицю **Маршрут** – 20.

Варіант 11, 12 – БД «Каталог фільмів»

Створити базу даних фільмофонду, в якій повинні бути виділені наступні базові сутності: **Актор, Жанр фільму, Фільм**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Країна (ідентифікатор країни, назва)

Жанр фільму (ідентифікатор жанру, назва жанру, дозволений вік).

Актор (ідентифікатор актора, прізвище, ім'я, по батькові, країна).

Фільм (ідентифікатор фільму, жанр, назва фільму, рік виходу, тривалість (у хвилинах), країна).

Крім базових сутностей в БД повинна бути похідна сутність **Роль**, яка містить інформацію про те, який актор у якому фільмі знімався, та яку роль грав. Ця сутність необхідна тому, що між таблицями **Актор** і **Фільм** в загальному випадку існує зв'язок багато-до-багатьох: один актор може грати в декількох фільмах, і у одному фільмі знімаються багато акторів. Ця таблиця містить два зовнішніх ключа, а їх сукупність складає первинний ключ – не будемо розглядати випадок, коли один актор в одному фільмі грає декілька ролей.

В таблицю **Жанр фільму** додати 5 записів, в таблицю **Актор** – 15, в таблицю **Фільм** – 15, в таблицю **Роль** – 25.

Варіант 13 – БД «Облік рейсів літаків»

Створити базу даних фільмофонду, в якій повинні бути виділені наступні базові сутності: **Місто, Літак**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Країна (ідентифікатор країни, назва)

Місто (ідентифікатор міста, назва міста, країна, кількість жителів).

Літак (ідентифікатор літака, назва літака, крейсерська швидкість кількість пасажирських місць)).

Крім базових сутностей в БД повинні бути дві похідні сутності: **Рейс**, яка містить інформацію про те, з якого міста на якому літаку в яке місто здійснюється рейс, та о котрій годині відліт та номер рейсу, – та сутність **Розклад**, яка містить інформацію, в які дати які рейси здійснюються. Таблиця **Розклад** має один зовнішній ключ, а первинним ключем її буде сукупність двох атрибутів.

В таблицю **Місто** додати 8 записів, в таблицю **Літак** – 8, в таблицю **Рейс** – 12, в таблицю **Розклад** – 25.

Варіант 14 – БД «Міжміські маршрути»

Створити базу даних пасажирських автомобільних перевезень, в якій повинні бути виділені наступні базові сутності: **Автобус, Місто**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Автобус (ідентифікатор автобусу, назва, кількість місць).

Місто (ідентифікатор міста, назва, область).

Крім базових сутностей в БД повинні бути похідні сутністі **Рейс, Маршрут** та **Розклад**.

Рейс містить інформацію про те, на якому автобусі з якого міста до якого міста виконується перевезення, та номер рейсу. Ця таблиця містить три зовнішніх ключа, при цьому один з них використовується в таблиці двічі (ідентифікатор міста – як місто відправлення, і як місто призначення).

Маршрут містить інформації про зупинки автобусів: номер рейсу, номер зупинки, місто, в якому відбувається зупинка, та час в дорозі до цієї зупинки.

Розклад містить інформацію про те, в які дати з якого часу виконуються рейси.

В таблицю **Автобус** додати 5 записів, в таблицю **Місто** – 12, в таблицю **Рейс** – 10, в таблицю **Маршрут** – 30, в таблицю **Розклад** – 10.

Варіант 15 – БД «Тенісні турніри»

Створити базу даних тенісних змагань, в якій повинні бути виділені наступні базові сутності: **Турнір**, **Тенісист**. Ці сутності мають наступні суттєві в контексті цієї БД атрибути:

Турнір (ідентифікатор турніру, країна проведення, назва турніру).

Тенісист (ідентифікатор тенісиста, прізвище, ім'я, по батькові, країна, рік народження, стать).

Крім базових сутностей в БД повинні бути три похідні сутності: **Проведення турніру**, яка містить інформацію про те, якій турнір в якому році проводився та з якої дати (початок турніру); сутність **Розклад**, яка містить інформацію, в якому турнірі в яку дату якій тенісист з яким тенісистом грає, та номер цього матчу (в середині турніру); **Результат матчу**, яка містить інформацію, в якому матчу якого турніру скільки геймів виграв першій тенісист, і скільки другий. В таблиці **Проведення турніру** повинна бути унікальною сукупність ідентифікатору турніру та року проведення, але в якості первинного ключа зручніше вибрати додатковий атрибут з властивістю IDENTITY, який буде зовнішнім ключем для таблиць **Розклад** і **Результат матчу**. В таблиці **Розклад** первинним ключем буде сукупність атрибутів турніру проведення і номеру матчу в середині турніру; але унікальною повинна бути сукупність атрибутів: турнір проведення, перший тенісист, другий тенісист. В таблиці **Результат матчу** сукупність атрибутів: турнір проведення і номер матчу в середині турніру, – є і зовнішнім ключем і унікальним.

В таблицю **Турнір** додати 5 записів, в таблицю **Тенісист** – 10, в таблицю **Проведення турніру** – 8, в таблицю **Розклад** – 16, в таблицю **Результат матчу** – 16.

Варіанти завдань створення представлень

Варіант 1 – БД «Деканат»

Створити представлення для таблиці **Залікова книжка**, яке крім атрибутів таблиці **Залікова книжка** повинно містити прізвище з ініціалами студента, назву дисципліни, прізвище з ініціалами викладача, назву оцінки.

Варіанти 2 – БД «Видавництва»

Створити представлення для таблиць **Книга** та **Книги авторів.**

В представленні для таблиці **Книга** крім атрибутів самої таблиці додати атрибути назва видавництва, назва жанру.

В представленні для таблиці **Книги авторів** крім атрибутів самої таблиці додати атрибути Письменник (в одному стовпчику прізвище, ім'я та по-батькові і у дужках електронна адреса), та Книжка – в одному стовпчику назва книги та через кому рік видавництва та назва видавництва.

Варіанти 3 – БД «Видавництва»

Створити представлення для таблиць **Книга** та **Книги авторів.**

В представленні для таблиці **Книга** крім атрибутів самої таблиці додати атрибути назва видавництва, назва жанру.

В представленні для таблиці **Книги авторів** крім атрибутів самої таблиці додати атрибути Письменник (в одному стовпчику прізвище, ім'я та по-батькові і у дужках телефон), та Книжка – в одному стовпчику назва книги та у дужках рік видавництва та назва видавництва.

Варіанти 4, 5 – БД «Авто-змагання»

Створити представлення для таблиць **Гонщик**, **Траса**, **Змагання** та **Результат змагань.**

В представленнях для таблиць **Гонщик** та **Траса** крім атрибутів самих таблиць додати атрибут назва країни.

В представленні для таблиці **Змагання** крім атрибутів самої таблиці додати колонку Trasa, в якій виводиться назва траси з вказівкою у дужках назви країни, до якої відноситься траса.

В представленні для таблиці **Результат змагань** крім атрибутів самої таблиці додати колонки Men та Comp. В колонку Men виводиться прізвище та імя гонщика з вказівкою у дужках назви країни, до якої відноситься гонщик. В колонку Men виводиться назва змагання з вказівкою у дужках назви траси, на якій проводиться змагання. Також потрібно додати колонку Time_res, в яку вивести час закінчення гонки, конвертований у символьний рядок у форматі часу з мілісекундами.

Варіанти 6 – БД «Поліклініка»

Створити представлення для таблиць **Лікар** та **Діагноз пацієнта.**

В представленні для таблиці **Лікар** крім атрибутів самої таблиці додати атрибут назва спеціалізації.

В представленні для таблиці **Діагноз пацієнта** крім атрибутів самої таблиці додати колонки FIO_p, FIO_d та назва діагнозу. В колонку FIO_p виводиться прізвище з ініціалами пацієнта з вказівкою у дужках повних років віку пацієнта. В колонку FIO_d виводиться прізвище з ініціалами лікаря з вказівкою у дужках назви спеціалізації.

Варіанти 7 – БД «Поліклініка»

Створити представлення для таблиць **Лікар** та **Діагноз пацієнта.**

В представленні для таблиці **Лікар** крім атрибутів самої таблиці додати атрибут назва спеціалізації.

В представленні для таблиці **Діагноз пацієнта** крім атрибутів самої таблиці додати колонки FIO_p, FIO_d та назва діагнозу. В колонку FIO_p

виводиться прізвище з ініціалами пацієнта з вказівкою у дужках дати народження пацієнта у форматі «dd/mm/yyyy». В колонку FIO_d виводиться назва спеціалізації лікаря та його прізвище з ініціалами, наприклад: терапевт Ковальчук Т.С.

Варіанти 8 – БД «Лісове господарство»

В представленні для таблиці **Зелений масив** крім атрибутів самої таблиці додати атрибут назва області.

В представленні для таблиці **Лісник** крім атрибутів самої таблиці додати атрибут назва зеленого масиву з вказанням у дужках назви області.

В представленні для таблиці **Дерева масивів** крім атрибутів самої таблиці додати атрибути назва зеленого масиву, назва породи дерева.

Варіанти 9 – БД «Лісове господарство»

В представленні для таблиці **Зелений масив** крім атрибутів самої таблиці додати атрибут назва області.

В представленні для таблиці **Лісник** крім атрибутів самої таблиці додати атрибут назва зеленого масиву з вказанням у дужках дати останньої перевірки.

В представленні для таблиці **Дерева масивів** крім атрибутів самої таблиці додати атрибути назва зеленого масиву, назва породи дерева.

Варіант 10 – БД «Екскурсії містом»

Створити представлення для таблиць **Визначні пам'ятки** та **Маршрут.**

В представленні для таблиці **Лікар** крім атрибутів самої таблиці додати атрибути назва вулиці та назва типу визначної пам'ятки.

В представленні для таблиці **Маршрут** крім атрибутів самої таблиці додати колонки Excur та Place. В колонку Excur виводиться назва екскурсії з вказівкою у дужках дати проходження у форматі «dd/mm/yyyy». В колонку Place виводиться назва типу визначної пам'ятки та назва самої пам'ятки.

Варіант 11 – БД «Каталог фільмів»

Створити представлення для таблиці **Актор**, в якому крім атрибутів таблиці **Актор** повинна бути назва країни актора.

Створити представлення для таблиці **Фільм**, в якому крім атрибутів таблиці **Фільм** повинна бути назва країни створення фільму та назва жанру.

В представленні для таблиці **Роль** крім атрибутів самої таблиці додати атрибути **Man** (в одному стовпчику прізвище, ім'я та у дужках країну), та

Kino – в одному стовпчику назва фільму та у дужках рік виходу та назва країни.

Варіант 12 – БД «Каталог фільмів»

Створити представлення для таблиці **Актор**, в якому крім атрибутів таблиці **Актор** повинна бути назва країни актора.

Створити представлення для таблиці **Фільм**, в якому крім атрибутів таблиці **Фільм** повинна бути назва країни створення фільму та назва жанру.

В представленні для таблиці **Роль** крім атрибутів самої таблиці додати атрибути **Actor_is** (в одному стовпчику прізвище, ім'я та у дужках країну), та **Kino** – в одному стовпчику назва фільму та через кому рік виходу та назва країни.

Варіант 13 – БД «Облік рейсів літаків»

Створити представлення для таблиць **Місто, Рейс** та **Розклад.**

В представленні для таблиці **Місто** крім атрибутів самої таблиці додати атрибут назва країни.

В представленні для таблиці **Рейс** крім атрибутів самої таблиці додати атрибути з назвами літака, міста відбуття та міста прибуття.

В представленні для таблиці **Розклад** крім атрибутів самої таблиці додати колонку From_To. В цю колонку виводиться назва міст відбуття та прибуття з вказівкою у дужках номеру рейсу, наприклад:

Київ – Одеса (214).

Варіант 14 – БД «Міжміські маршрути»

Створити представлення для таблиць **Рейс, Маршрут** та **Розклад.**

В представленні для таблиці **Рейс** крім атрибутів самої таблиці додати атрибути з назвами автобусу, міста відбуття та міста прибуття.

В представленні для таблиці **Розклад** крім атрибутів самої таблиці додати колонку From_To. В цю колонку виводиться назва міст відбуття та прибуття з вказівкою у дужках номеру рейсу, наприклад: Київ – Одеса (214).

В представленні для таблиці **Маршрут** крім атрибутів самої таблиці додати колонку з назвою міста зупинки автобусу.

Варіант 15 – БД «Тенісні турніри»

Турнір (ідентифікатор турніру, країна проведення, назва турніру).

Тенісист (ідентифікатор тенісиста, прізвище, ім’я, по батькові, країна, рік народження, стать).

Створити представлення для таблиць **Проведення турніру**, **Розклад** та **Результат матчу.**

В представленні для таблиці **Проведення турніру** крім атрибутів самої таблиці додати назва турніру.

В представленні для таблиці **Розклад** крім атрибутів самої таблиці додати колонки Tour, gamer1, gamer2. В колонку Tour виводиться назва турніру з вказівкою у дужках року проведення. В колонці gamer1 виводиться Ім’я та прізвище першого тенісиста з вказівкою у дужках назви його країни. Аналогічно для другого тенісиста.

В представленні для таблиці **Результат матчу** крім атрибутів самої таблиці додати колонки Match_Tour, gamers, game_res. В колонку Match_Tour виводиться номер матчу та через крапку назва турніру з вказівкою у дужках року проведення, наприклад:

5. Уімблдон (2012).

В колонку gamers виводиться ім’я та прізвище першого тенісиста та через дефіс ім’я та прізвище першого.

В колонку game_res виводиться кількість геймів, які виграв 1-й тенісист, та через двократку – кількість геймів, які виграв 2-й тенісист, наприклад:

6:3

Лабораторна робота №6. Збережені процедури в базі даних

Перелік тем лекційного курсу

1. Серверне програмне забезпечення.
2. Команда створення збереженої процедури для бази даних в СКБД MS SQL Server.

Основні теоретичні відомості

Команда створення збереженої процедури для бази даних в СКБД MS SQL Server

Команда створення збережених процедур має декілька опцій, не всі з яких використовуються однаково часто. Найчастіше ця команда використовується у наступному форматі:

```
CREATE PROC[EDURE ]  procedure_name
   [ { @parameter data_type } [ OUTPUT ]
   ] [ ,...n ]

AS
BEGIN
sql_statement [ ...n ]
END

GO
```

Аргументи

procedure_name

Ім'я збереженої процедури. Воно грає роль ідентифікатора і має бути унікальним у межах бази даних.

@parameter

Параметр процедури. Один або більше параметрів можуть бути об'явлені в команді CREATE PROCEDURE. Значення кожного обявленого параметра повинно задаватись користувачем при виклику збереженої процедури (якщо параметр не визначений як параметр по умовчанню). Збережена процедура може мати maximum 2100 параметрів.

Ім'я параметра процедури повинно починатися з символа (@). Параметр є локальною змінною процедури; параметри з однаковими іменами можуть використовуватись у різних процедурах.

data_type

Тип даних параметра. Усі типи даних, у тому числі **text, ntext** та **image**, можуть використовуватись у якості параметрів збережених процедур. Але тип даних **cursor** може використовуватись лише у якості вихідного (OUTPUT) параметра. Якщо використовується параметр типу **cursor**, для нього повинно бути вказане ключове слово OUTPUT.

OUTPUT

Вказує, що процедура повертає значення цього параметра

[**,**...*n*]

Вказує, що декілька параметрів процедури відокремлюються комою.

AS

Вказує, що далі йде програмний код процедури.

BEGIN

END

Операторні дужки.

sql_statement

Довілька команда мови Transact-SQL може входити у програмний код процедури. Є деякі обмеження, наприклад, в коді процедури не може бути присутня команда GO

n

Код збереженої процедури може складатись з довільної кількісті команд мови Transact-SQL.

Збережені процедури можуть створюватись для будь-якої обробки інформації в базі даних. Одним з прикладів використання збережених процедур є створення процедур для маніпулювання даними в таблицях: для додавання запису в таблицю, змінення значення атрибутів записів в таблиці, видалення записів з таблиці. Крім власне команд маніпулювання даними: INSERT, UPDATE або DELETE, – такі процедури повинні містити додаткові команди перевірки даних для забезпечення збереження цілісності бази даних після додавання, змінення або видалення записів в таблиці.

Для збережених процедур, які використовуються для видалення записів з таблиць, бажано перевіряти, чи повинне бути каскадне видалення записів з підлеглих таблиць.

При додавання та оновленні записів в таблицях потрібно перевіряти, чи не вводяться вже існуючі в таблиці значення для унікальний ключів.

Розглянемо ці вимоги на прикладі баз даних GreenMassiv, Publishing, Filmoteka та Racing, діаграми яких неведені на рис.2-5.

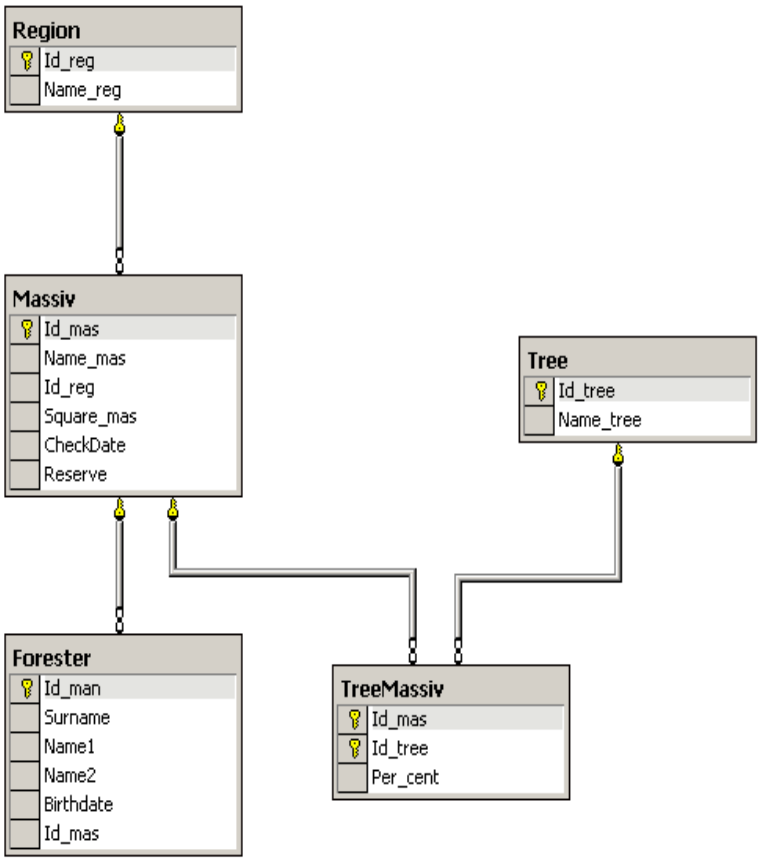

Рисунок 10 – Схема бази даних GreenMassiv

В базі даних GreenMassiv таблиці Forester та TreeMassiv не мають підлеглих (дочірніх) таблиць, тому для цих таблиць не потрібно перевіряти, чи буде каскадне видалення записів у підлеглих таблицях. Тому збережені процедури видалення записів з цих таблиць будуть зовсім прості. У якості параметрів цих збережених процедур повинні виступати лише первинні ключі таблиць: один параметр @Id_man для таблиці Forester, та два параметри @Id_mas та @Id_tree для таблиці TreeMassiv. Наприклад, процедура для таблиці Forester буде мати вигляд:

```
CREATE PROCEDURE ForesterDel
@Id_man smallint
AS
    DELETE Forester where Id_man=@Id_man
GO
```

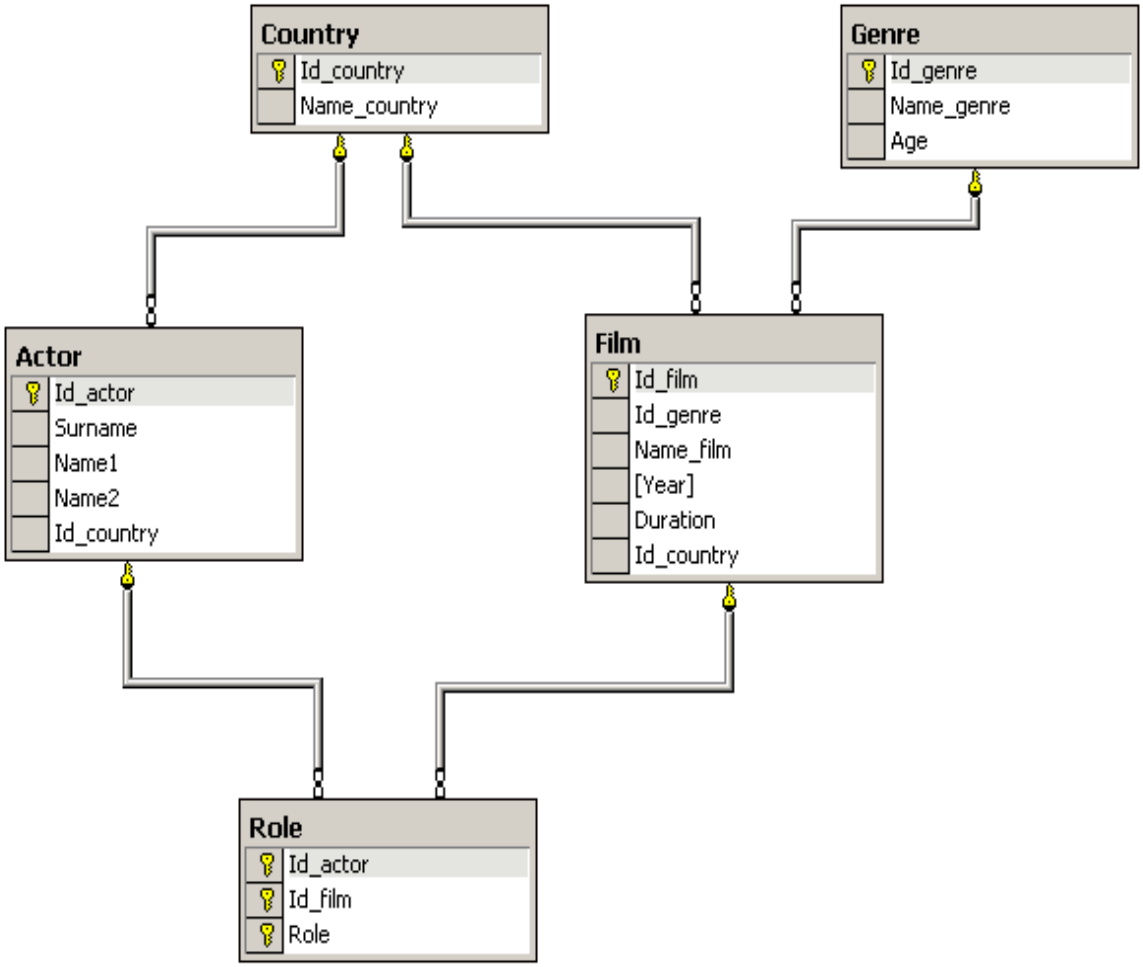

Рисунок 11 – Схема бази даних Filmoteka

В базах даних Publishing, Filmoteka та Racing є лише одна таблиця, що не має підлеглих таблиць, це таблиці ABook, Role та Result відповідно. Всі ці таблиці мають складові первинні ключі. У якості параметрів збереженої процедури для видалення записів з цих таблиць повинні виступати всі складові первинного ключа. Наприклад, в таблиці Role бази даних Filmoteka всі три атрибута разом складають первинний ключ. Тому збережена процедура для видалення записів з цієї таблиці буде мати вигляд:

```
CREATE PROCEDURE RoleDel
@Id_actor smallint,
@Id_film smallint,
@Role varchar(60)
AS
    DELETE Role where Id_actor =@Id_actor AND Id_film=@Id_film
    AND Role=@Role
GO
```

Зауваження: типи даних параметрів процедури повинні відповідати типам даних тих атрибутів таблиць, значенні яких вони представляють

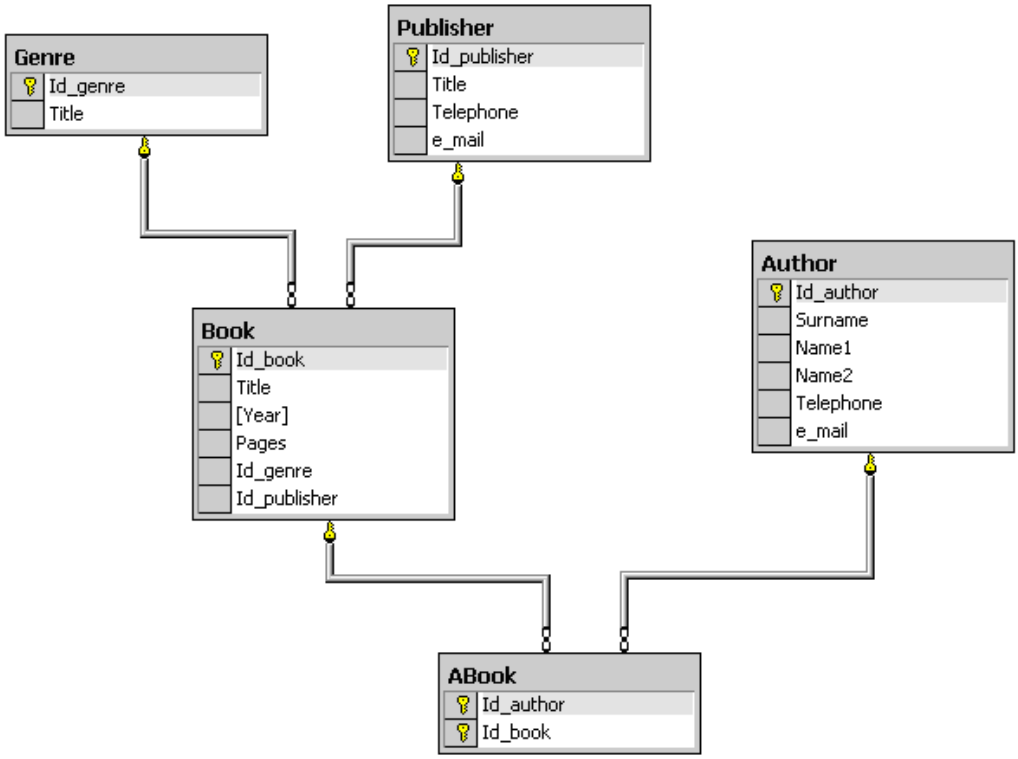

Рисунок 12 – Схема бази даних Publishing

Якщо таблиця має підлеглі таблиці, то під час видалення запису з такої таблиці можливі два випадки:

1. видалити запис разом з усіма записами в підлеглих таблицях, які зв’язані з цим записом;
2. видалити запис тільки у випадку відсутності зв’язаних з ним записів у підлеглих таблицях; при наявності зв’язаних записів повідомити про неможливість видалення запису з таблиці.

Таким чином у процедурі видалення записів з таблиць, що мають підлеглі таблиці, повинні бути додаткові параметри, крім ключових атрибутів таблиці. Один з цих параметрів повинен вказувати, по якому з двох вказаних вище випадків повинні видалятись записи в таблиці. Інші параметри повинні бути вихідними параметриами (параметрами, що повертаються) для повідомлення, чи відбулось видалення з таблиці.

Наприклад, в базі даних Publishing є дві таблиці: Book та Author, – у яких є тільки одна підлегла таблиця, яка сама не має підлеглих таблиць. Тому при видаленні записів з цих таблиць потрібно перевіряти тільки наявність зв’язаних записів в таблиці ABook. Процедура видалення записів з таблиці Book може мати вигляд:

```
CREATE PROCEDURE BookDel
@Id_book smallint,
@m bit,
@k tinyint output,
@msg varchar(100) output
AS
BEGIN
        if @m=0
        BEGIN
                DELETE ABook where Id_book=@Id_book
                DELETE Book where Id_book=@Id_book
                SET @k=0
                SET @msg='OK'
        END
        ELSE
        BEGIN
                SET @k=(SELECT k=COUNT(*) from ABook
                where Id_book=@Id_book)
                if @k=0
                begin
                        SET @msg='OK'
                        DELETE Book where Id_book=@Id_book
                End
                ELSE
                        SET @msg='В таблиці  ABook є '+STR(@k,4)+
                                ' записів для цієї книги'
        END
END
GO
```

В процедурі BookDel параметр @m використовується для визначення потрібного випадка видалення запису з таблиці Book: для @m=0 використовується каскадне видалення записів з підлеглої таблиці ABook; для @m=1 каскадного видалення не відбувається, і запис з таблиці Book видаляється тільки у випадку, коли немає зв'язаних з ним записів в таблиці ABook.

Вихідні параметри @k та @msg мають сенс кода помилки та повідомлення про помилку: коли @k=0, запис з таблиці Book видаляється.

Такий формат збереженої процедури для видалення записів буде у тих випадках, коли видаляється запис з таблиці, яка має лише одну підлеглу таблицю. Тобто ций формат підходить для видалення записів з таблиці Author бази даних Publishing (підлегла таблиця ABook); з таблиці Tree бази

даних GreenMassiv (підлегла таблиця TreeMassiv); з таблиці Tree бази даних GreenMassiv (підлегла таблиця TreeMassiv); з таблиці Film бази даних Filmoteka (підлегла таблиця Role); з таблиці Actor бази даних Filmoteka (підлегла таблиця Role); з таблиці Racer бази даних Racing (підлегла таблиця Result); з таблиці Competition бази даних Racing (підлегла таблиця Result).

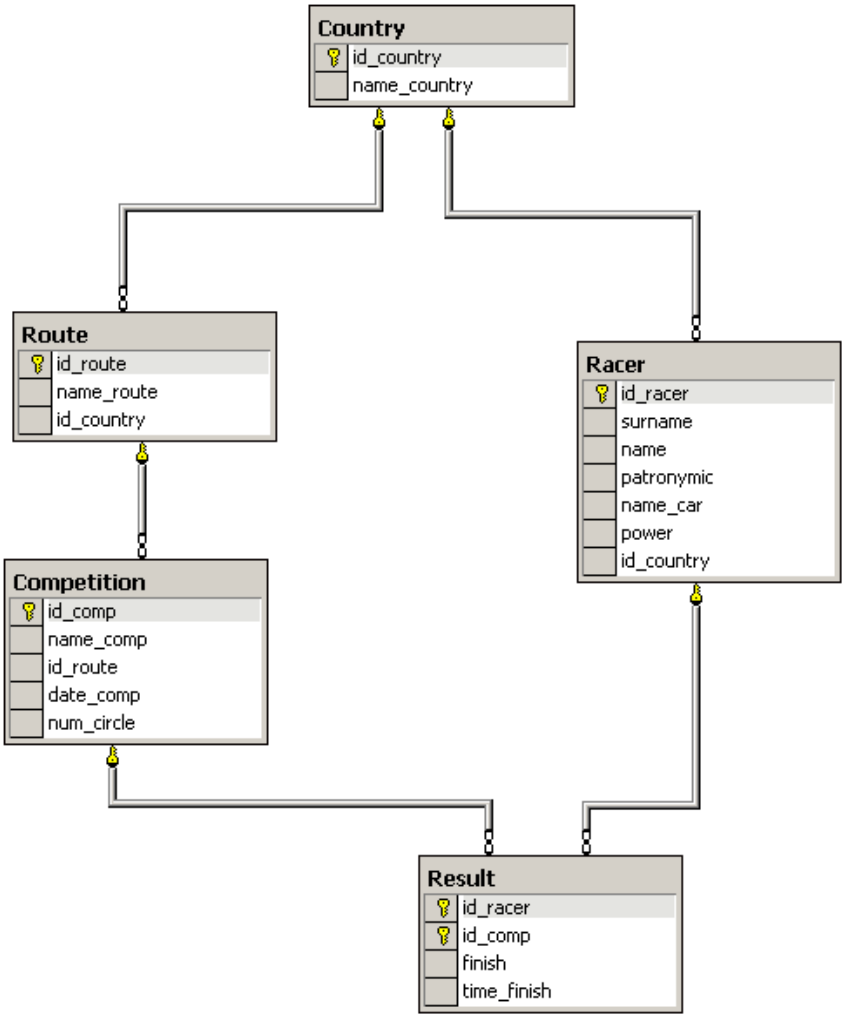

Рисунок 13 – Схема бази даних Racing

Якщо у таблиці є підлеглі таблиці уц двох рівнях: Genre – Film – Role (Filmoteka), Publisher – Book – ABook та Publisher – Book – ABook (Publishing), Route – Competition – Result (Racing), потрібно проводити перевірку у двох рівнях, як видно з прикладу процедури RouteDel (процедури видалення запису з таблиці Route бази даних Racing):

```
CREATE PROCEDURE RouteDel
@id_route smallint,
@m bit,
@k tinyint output,
@msg varchar(100) output
AS
```

```
BEGIN
	if @m=0
	BEGIN
		DELETE Result where id_comp IN
		(SELECT id_comp FROM Competition
		where id_route=@id_route)
		DELETE Competition where id_route=@id_route
		DELETE Route where id_route=@id_route
		SET @k=0
		SET @msg='OK'
	END
	ELSE
	BEGIN
		SET @k=(SELECT k=COUNT(*) from Competition
		where id_route=@id_route)
		if @k=0
		begin
			SET @msg='OK'
			DELETE Route where id_route=@id_route
		End
		ELSE
		BEGIN
			DECLARE @@k tinyint
			SET @@k=(SELECT k=COUNT(*) from Result
			where id_comp IN (SELECT id_comp FROM
			Competition where id_route=@id_route))
			if @@k=0
			BEGIN
				SET @msg='В таблиці  Competition є '
				+STR(@k,4)+ ' записів для цієї траси без
				результатів змагань'
				SET @k=1
			END
			ELSE
			BEGIN
				SET @msg='В таблиці Competition є '
				+STR(@k,4)+ ' записів для цієї траси та є'
				+STR(@@k,4)+' записів результатів змагань'
				SET @k=2
			END
		END
	END
END
GO
```

При створенні збережених процедур для додавання та оновлення записів в таблицях необхідно також стежити за виконанням обмежень цілісності бази даних. У цих випадках можуть порушуватись як цілісність сутностей: не унікальні значення первинного (PRIMARY KEY) та потенційного (UNIQUE) ключів, – так і посилальна цілісність: невідповідність значення зовнішнього ключа наявним значенням відповідного первинного ключа.

При створенні збережених процедур для додавання та оновлення записів в таблицях не будемо перевіряти збереження посилальної цілісності, тому що ці процедури будуть використовуватись у клієнтській програмі таким чином, який унеможливлює порушення посилальної цілісності – значення зовнішнього ключа не присвоюється якомусь довільному числу, а вибирається з наявних значень відповідного первинного ключа.

Для збереження цілісності сутностей перед додаванням записа в таблицю, або перед зміною значень атрибутів в записах таблиці, необхідно перевіряти, чи є вже в таблиці інший запис з таким самим значенням унікального ключа. Для первинного ключа така перевірка проводиться тільки у тому випадку, коли він не має властивості автонумерації (тобто для мови Transact-SQL для СКБД MS SQL Server не має властивості IDENTITY).

Наприклад, команда створення таблиці Tree у базі даних GreenMassiv (рис.10) має вигляд:

```
CREATE TABLE Tree(
        Id_tree smallint IDENTITY PRIMARY KEY,
        Name_tree varchar(50) NOT NULL UNIQUE
)
```

При додаванні записів в цю таблицю потрібно перевіряти, чи немає вже в таблиці БД дерева з такою ж назвою:

```
CREATE PROCEDURE TreeAdd
        @Name_tree  varchar(50),
        @Id_tree smallint OUTPUT,
        @k tinyint OUTPUT,
        @msg varchar(100) OUTPUT
AS
BEGIN
        SET @k = (SELECT k=COUNT(*) FROM Tree
                        WHERE Name_tree = @Name_tree )
        IF @k = 0
        BEGIN
                        SET @msg = 'OK'
                        INSERT Tree(Name_tree) VALUES (@Name_tree)
                        SET @Id_tree = (SELECT Id_tree FROM Tree
```

```
                WHERE Name_tree = @Name_tree )
    END
    ELSE
        SET @msg = ‘В базі даних вже є запис про дерево з такою назвою’
END
GO
```

При створення збереженої процедури для оновлення записів потрібно перевіряти, чи є інші записи з новим значення унікального атрибуту, тобто записи з іншим значенням первинного ключа. Наприклад, в таблиці Genre БД Filmoteka (рис.11) для запису можно змінювати значення двох атрибутів: Name_genre (назва жанру фильма) та Age (дозволений вік):

```
CREATE TABLE Genre (
    Id_genre smallint IDENTITY PRIMARY KEY,
    Name_genre varchar(40) NOT NULL UNIQUE,
    Age tinyint
)
```

Атрибут Name_genre є унікальним ключем. Якщо для жанру буде змінюватись лише значення атрибуту Age, а значення Name_genre буде залишатись старим, то це є дозволена операція, і вона не повинна викликати помилку. Тому команда створення процедури для оновлення записів в таблиці Genre має вигляд:

```
CREATE PROCEDURE GenreEdit
    @Id_genre smallint,
    @Name_genre  varchar(40),
    @Age tinyint,
    @k tinyint OUTPUT,
    @msg varchar(100) OUTPUT
AS
BEGIN
    SET @k = (SELECT k=COUNT(*) FROM Genre
                WHERE Name_genre = @Name_genre  AND
                Id_genre != @Id_genre)
    IF @k = 0
    BEGIN
                SET @msg = ‘OK’
        UPDATE Genre SET Name_genre = @Name_genre
                Age = @Age WHERE Id_genre = @Id_genre
    END
```

```
    ELSE
        SET @msg = 'В базі даних вже є запис про жанр з такою назвою'
END
GO
```

Варіанти завдань створення збережених процедур

Варіант 1 – БД «Деканат»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Студент**, **Предмет, Залікова книжка**.

Варіанти 2 – БД «Видавництва»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Автор**, **Жанр книг**, **Книга**.

Варіанти 3 – БД «Видавництва»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Автор**, **Видавництво**, **Книги авторів**.

Варіанти 4 – БД «Авто-змагання»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Країна, Гонщик**, **Змагання**.

Варіанти 5 – БД «Авто-змагання»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Країна, Траса**, **Результат змагань**.

Варіанти 6 – БД «Поліклініка»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Пацієнт**, **Спеціалізація лікаря**, **Лікар**.

Варіанти 7 – БД «Поліклініка»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Пацієнт**, **Діагноз, Діагноз пацієнта**.

Варіанти 8 – БД «Лісове господарство»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Порода дерев**, **Зелений масив, Дерева масивів**.

Варіанти 9 – БД «Лісове господарство»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Область**, **Зелений масив, Лісник**.

Варіант 10 – БД «Екскурсії містом»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Вулиця**, **Визначні пам'ятки, Маршрут**.

Варіант 11– БД «Каталог фільмів»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Країна**, **Актор, Фільм**.

Варіант 12 – БД «Каталог фільмів»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Жанр фільму**, **Фільм, Роль**.

Варіант 13 – БД «Облік рейсів літаків»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Літак**, **Місто**, **Рейс**.

Варіант 14 – БД «Міжміські маршрути»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Місто**, **Рейс, Маршрут**.

Варіант 15 – БД «Тенісні турніри»

Створити збережені процедури для додавання, оновлення та видалення записів в таблицях: **Тенісист**, **Проведення турніру, Розклад**.

Література

1. Дейт К. Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.: ил.
2. Мейер С.М. Проектирование баз данных – М.: Мир, 1987.
3. Мамаев Е.В. Micrjsoft® SQL Sever 2000. — СПб.: БХВ-Петербург, 2004. — 1280 с.: ил.
4. Дэвидсон Л. Проектирование баз даннях на SQL Server 2000.: Пер. с англ. – М..: БИНОМ. Лаборатория знаний, 2003. – 680 с., ил.
5. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1440 с.: ил.