МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

А. М. Копп, Д. Л. Орловський

# ОСНОВИ РОБОТИ З БАЗАМИ ДАНИХ ІЗ ВИКОРИСТАННЯМ СУБД MICROSOFT SQL SERVER

Навчально-методичний посібник
для студентів спеціальностей
F2 «Інженерія програмного забезпечення» та F3 «Комп'ютерні науки»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 30.10.2025 р.

Харків
НТУ «ХПІ»
2026

УДК 004.65
К 65

Рецензенти:

*М. А. Гринченко*, канд. техн. наук, доц.,
Національний технічний університет
«Харківський політехнічний інститут»;

*Г. А. Плєхова*, канд. техн. наук, доц.,
Харківський національний автомобільно-дорожній університет

**Копп А. М.**

**К 65** Основи роботи з базами даних із використанням СУБД Microsoft SQL Server : навчально-методичний посібник для студентів спеціальностей F2 «Інженерія програмного забезпечення» та F3 «Комп'ютерні науки» / А. М. Копп, Д. Л. Орловський. Харків : НТУ «ХПІ», 2026. 349 с.

**ISBN 978-617-05-0596-5**

У навчально-методичному посібнику розглядається комплекс питань, пов'язаних із методичними основами та засобами створення і застосування баз даних із використанням системи управління базами даних Microsoft SQL Server.

Розглянуто основи мови запитів, типи даних, створення бази даних та основних її об'єктів, засоби відображення об'єктів баз даних та всіх їх характеристик. Показано методи проектування, створення та зміни структури бази даних; розглянуті питання маніпулювання даними, використання представлень, процедур, що зберігаються та тригерів; використання транзакції для роботи з даними. Розглянуто приклади створення засобів для роботи із базами даних на рівні кінцевих користувачів.

Призначено для студентів, що навчаються за спеціальностями F2 «Інженерія програмного забезпечення» та F3 «Комп'ютерні науки» і слухачів післядипломної системи всіх форм навчання.

Іл. 237. Табл. 11. Бібліогр. 24 назви.

УДК 004.65

ISBN 978-617-05-0596-5

ЗМІСТ

Вступ ..... 8
Тема 1 Загальні відомості про мову SQL та СУБД Microsoft SQL Server... 12
1.1 Мова SQL ..... 12
1.1.1 Загальна стисла характеристика мови SQL ..... 12
1.1.2 Переваги мови SQL ..... 15
1.1.3 Запис SQL-операторів ..... 19
1.1.4 Типи даних мови SQL, визначені стандартом ..... 20
1.2 СУБД Microsoft SQL Server – призначення та основні особливості ..... 25
1.2.1 Загальна стисла характеристика СУБД Microsoft SQL Server .. 25
1.2.2 Типи даних, використовувані в СУБД Microsoft SQL Server .... 32
1.2.3 Основні об'єкти структури бази даних в СУБД Microsoft SQL Server ..... 37
1.2.4 Мова Transact-SQL ..... 39
1.2.5 Застосунок SQL Server Management Studio ..... 41
1.3 Питання для самоперевірки по темі 1 ..... 42
Тема 2 Опис предметної області та структури бази даних ..... 43
2.1 Стислий опис предметної області ..... 43
2.2 Опис структури бази даних ..... 46
2.3 Питання для самоперевірки по темі 2 ..... 48
Тема 3 Створення бази даних та введення даних у базу даних ..... 51
3.1 Теоретичні відомості ..... 51
3.1.1 Оператори DDL мови SQL та Transact-SQL для роботи з базами даних та таблицями бази даних ..... 51
3.1.2 Оператори DML мови SQL та Transact-SQL ..... 59
3.2 Практичне опанування теми 3 ..... 66
3.2.1 Передумови виконання завдань теми 3 ..... 66
3.2.2 Створення бази даних та введення даних у базу даних в інтерактивному режимі ..... 66

3.2.3 Створення бази даних та введення даних у базу даних із використанням засобів мови Transact-SQL ..... 77
3.2.4 Звітність про виконання практичних завдань теми 3 ..... 83
3.3 Питання для самоперевірки по темі 3 ..... 84
Тема 4 Обробка даних засобами мови SQL у середовищі СУБД Microsoft SQL Server ..... 86
4.1 Теоретичні відомості ..... 86
4.1.1 Загальні відомості про засоби обробки даних мови SQL ..... 86
4.1.2 З'єднання і теоретико-множинні операції над відношеннями ..... 91
4.1.3 Обчислення і підведення підсумків в запитах ..... 101
4.1.4 Використання підзапитів та CTE ..... 104
4.2 Практичне опанування теми 4 ..... 107
4.2.1 Передумови виконання завдань теми 4 ..... 107
4.2.2 Побудова та виконання запитів ..... 108
4.2.3 Звітність про виконання практичних завдань теми 4 ..... 126
4.3 Питання для самоперевірки по темі 4 ..... 127
Тема 5 Створення та використання програмних об'єктів бази даних засобами СУБД Microsoft SQL Server ..... 129
5.1 Теоретичні відомості ..... 129
5.1.1 Створювані користувачем функції та вбудовані функції ..... 129
5.1.2 Збережені процедури ..... 137
5.1.3 Тригери ..... 144
5.1.4 Курсори ..... 152
5.2 Практичне опанування теми 5 ..... 162
5.2.1 Передумови виконання завдань теми 5 ..... 162
5.2.2 Побудова та використання створюваних користувачем функцій ..... 162
5.2.3 Побудова та використання збережених процедур ..... 168
5.2.4 Побудова та використання тригерів ..... 174
5.2.5 Побудова та використання курсорів ..... 185

5.2.6 Звітність про виконання практичних завдань теми 5 ..... 188
5.3 Питання для самоперевірки по темі 5 ..... 189
Тема 6 Створення та використання представлень (view) засобами СУБД Microsoft SQL Server ..... 191
6.1 Теоретичні відомості ..... 191
6.1.1 Поняття представлення ..... 191
6.1.2 Оновлення даних в представленнях ..... 193
6.1.3 Переваги і недоліки представлень ..... 194
6.2 Практичне опанування теми 6 ..... 198
6.2.1 Передумови виконання завдань теми 6 ..... 198
6.2.2 Побудова та використання представлення для перегляду даних ..... 199
6.2.3 Оновлення даних за допомогою представлень ..... 202
6.2.4 Звітність про виконання практичних завдань теми 6 ..... 208
6.3 Питання для самоперевірки по темі 6 ..... 209
Тема 7 Вивчення основ роботи із засобами контролю цілісності даних у середовищі СУБД Microsoft SQL Server ..... 210
7.1 Теоретичні відомості ..... 210
7.1.1 Основні поняття цілісності даних ..... 210
7.1.2 Визначення обмежень цілісності ..... 216
7.1.3 Визначення обмежень цілісності в середовищі MS SQL Server ..... 221
7.2 Практичне опанування теми 7 ..... 231
7.2.1 Передумови виконання завдань теми 7 ..... 231
7.2.2 Вивчення особливостей роботи механізму контролю посилальної цілісності No Action ..... 231
7.2.3 Вивчення особливостей роботи механізму контролю посилальної цілісності Cascade ..... 235
7.2.4 Вивчення особливостей роботи механізму контролю посилальної цілісності Set Null ..... 236

7.2.5 Вивчення особливостей роботи механізму контролю посилальної цілісності Set Default ..... 238
7.2.6 Звітність про виконання практичних завдань теми 7 ..... 241
7.3 Питання для самоперевірки по темі 7 ..... 242
Тема 8 Використання транзакцій на прикладі СУБД Microsoft SQL Server ..... 244
8.1 Теоретичні відомості ..... 244
8.1.1 Поняття транзакції ..... 244
8.1.2 Властивості транзакцій ..... 245
8.1.3 Блокування ..... 246
8.1.4 Управління транзакціями ..... 249
8.1.5 Управління транзакціями в середовищі Microsoft SQL Server ..... 250
8.1.6 Блокування в середовищі Microsoft SQL Server ..... 254
8.1.7 Рівні ізоляції в середовищі Microsoft SQL Server ..... 257
8.2 Практичне опанування теми 8 ..... 259
8.2.1 Передумови виконання завдань теми 8 ..... 259
8.2.2 Основи використання транзакцій ..... 259
8.2.3 Рівні ізоляції транзакцій та їх особливості ..... 265
8.2.4 Звітність про виконання практичних завдань теми 8 ..... 269
8.3 Питання для самоперевірки по темі 8 ..... 270
Тема 9 Реалізація простого клієнтського застосунку для роботи з базою даних Microsoft SQL Server ..... 272
9.1 Теоретичні відомості ..... 272
9.1.1 SQL і прикладне програмне забезпечення ..... 272
9.1.2 Технологія ODBC та її особливості ..... 275
9.2 Практичне опанування теми 9 ..... 280
9.2.1 Передумови виконання завдань теми 9 ..... 280
9.2.2 Створення джерела даних ODBC ..... 280
9.2.3 Використання СУБД Microsoft Access як клієнтського застосунку ..... 281

9.2.4 Звітність про виконання практичних завдань теми 9 ..... 285
9.3 Питання для самоперевірки по темі 9 ..... 286
Тема 10 Розробка засобами Microsoft Visual Studio прикладного програмного забезпечення для роботи з базою даних Microsoft SQL Server ..... 288
10.1 Теоретичні відомості ..... 288
10.1.1 Визначення та суть інтерфейсу користувача ..... 290
10.1.2 Графічний інтерфейс користувача ..... 292
10.1.3 Принципи побудови «дружнього» інтерфейсу користувача ..... 294
10.1.4 Правила проектування інтерфейсу користувача ..... 296
10.2 Практичне опанування теми 10 ..... 306
10.2.1 Передумови виконання завдань теми 10 ..... 306
10.2.2 Створення проекту та головної форми ..... 306
10.2.3 Створення найпростіших форм для роботи з даними ..... 309
10.2.4 Створення форми, що забезпечує роботу з кількома таблицями ..... 318
10.2.5 Створення звітної форми з узагальненими даними ..... 340
10.2.6 Звітність про виконання практичних завдань теми 10 ..... 343
10.3 Питання для самоперевірки по темі 10 ..... 344
Рекомендована література ..... 346
Стислі відомості про авторів ..... 348

# ВСТУП

Сучасні інформаційні системи ґрунтуються на використанні баз даних, в яких накопичується різна інформація. Тому зараз розробляються і значно поширюються методи і засоби роботи з базами даних з метою підвищення ефективності роботи людини у різних галузях діяльності. Ці засоби та методи пов'язані з узагальненням і різними додатковими способами обробки даних. Основні ідеї сучасної інформаційної технології базуються на концепції, згідно з якою дані повинні бути організовані в бази даних з метою адекватного відображення реального світу, що змінюється, і задоволення інформаційних потреб користувачів. Ці бази даних створюються і функціонують під управлінням спеціальних програмних комплексів, які називають системами управління базами даних (СУБД).

Зараз на ринку програмного забезпечення пропонується досить багато різних СУБД, які мають різні властивості, функціональні можливості, спрямування тощо. Однією з найбільш відомих та популярних СУБД є Microsoft SQL Server від корпорації Microsoft.

Навчально-методичний посібник «Основи роботи з базами даних із використанням СУБД Microsoft SQL Server» призначений для студентів, що навчаються за спеціальностями F2 «Інженерія програмного забезпечення» та F2 «Комп'ютерні науки».

Використання навчально-методичного посібника передбачає наявність у студентів базових знань щодо реляційних баз даних, особливостей їх проектування та розробки.

Навчально-методичний посібник складається з низки тем, опанування яких має на меті надання студентам базових знань та навичок

щодо використання СУБД Microsoft SQL Server для вирішення задач, пов'язаних із роботою з даними. Далі наведений перелік тем та стисла характеристика матеріалу, що розглядається у цих темах.

**Тема 1 «Загальні відомості про мову SQL та СУБД Microsoft SQL Server».** Тема має загальне та оглядове спрямування стосовно баз даних, мови SQL та СУБД Microsoft SQL Server, практичної частини не містить. Опанування матеріалу цієї теми для студентів є бажаним, але не обов'язковим. Надання студентами звітності про опанування матеріалу цієї теми не вимагається.

**Тема 2 «Опис предметної області та структури бази даних».** Тема має концептуальне спрямування, практичної частини не містить. Але опанування матеріалу цієї теми для студентів є обов'язковим, оскільки тут розглядаються особливості предметної області, для якої буде у подальшому створено базу даних, а також надається опис структури бази даних. Надання студентами звітності про опанування матеріалу цієї теми не вимагається.

**Тема 3 «Створення бази даних та введення даних у базу даних».** Тема має практичне спрямування, пов'язане із питаннями створення бази даних та введення даних у базу даних. Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 4 «Обробка даних засобами мови SQL у середовищі СУБД Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями обробки даних, які зберігаються у базі даних, засобами мови SQL. Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична

частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 5 «Створення та використання програмних об'єктів бази даних засобами СУБД Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями створення та використання програмних об'єктів бази даних (збережені процедури, тригери тощо). Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 6 «Створення та використання представлень (view) засобами СУБД Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями створення та використання спеціальних об'єктів бази даних – представлень (view). Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 7 «Вивчення основ роботи із засобами контролю цілісності даних у середовищі СУБД Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями контролю цілісності даних, які зберігаються у базі даних. Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 8 «Використання транзакцій на прикладі СУБД Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями використання транзакцій при роботі із даними, які зберігаються у базі

даних. Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які можуть допомогти студенту при виконанні практичних завдань.

**Тема 9 «Реалізація простого клієнтського застосунку для роботи з базою даних Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями створення дуже простого клієнтського застосунку для роботи із даними, які зберігаються у базі даних. При цьому як засіб розробки клієнтського застосунку розглядається СУБД Microsoft Access. Виходячи з того, що у деяких студентів можуть бути певні складності стосовно використання Microsoft Access (відсутність досвіду роботи та/або проблеми із доступом до Microsoft Access тощо), **рішення про обов'язковість** опанування студентами цієї теми та надання звітності **приймає викладач**, який відповідає за навчальну дисципліну, для підтримки вивчення якої використовується цей навчально-методичний посібник. Рішення приймається та доводиться до відому студентів перед початком вивчення навчальної дисципліни.

**Тема 10 «Розробка засобами Microsoft Visual Studio прикладного програмного забезпечення для роботи з базою даних Microsoft SQL Server».** Тема має практичне спрямування, пов'язане із питаннями створення клієнтського застосунку для роботи із даними, які зберігаються у базі даних, на рівні кінцевого користувача (End User). Тема містить теоретичну і практичну частини. Практичне опанування теми для студентів **є обов'язковим**. Надання студентами звітності про опанування практичного матеріалу цієї теми **є обов'язковим**. Теоретична частина теми містить стислий огляд питань, які стосуються проектування та розробки інтерфейсу користувача (User Interface).

# ТЕМА 1
# ЗАГАЛЬНІ ВІДОМОСТІ ПРО МОВУ SQL ТА СУБД MICROSOFT SQL SERVER

## 1.1 Мова SQL

### 1.1.1 Загальна характеристика мови SQL

Зростання кількості даних, необхідність їх зберігання і обробки привели до того, що виникла потреба в створенні стандартної мови баз даних, яка могла би функціонувати в численних комп'ютерних системах різних видів. Дійсно, з її допомогою користувачі можуть маніпулювати даними незалежно від того, чи працюють вони на персональному комп'ютері, мережевій робочій станції або універсальній ЕОМ.

Однією з мов, що з'явилися в результаті розробки реляційної моделі даних, є мова SQL (Structured Query Language), яка нині отримала дуже широке поширення і фактично перетворилася на стандартну мову реляційних баз даних. Стандарт на мову SQL був випущений Американським національним інститутом стандартів (ANSI) в 1986 р., а в 1987 р. Міжнародна організація стандартів (ISO) прийняла його як міжнародний. Нинішній стандарт SQL відомий під назвою SQL/92.

З використанням будь-яких стандартів пов'язані не лише численні і цілком очевидні переваги, але і певні недоліки. Передусім, стандарти направляють в певне русло розвиток відповідної індустрії; у разі мови SQL наявність твердих засадничих принципів приводить, кінець кінцем, до сумісності його різних реалізацій і сприяє як підвищенню переносимості програмного забезпечення і баз даних в цілому, так і універсальності роботи адміністраторів баз даних. З іншого боку, стандарти обмежують гнучкість і функціональні можливості конкретної реалізації. Під реалізацією мови SQL розуміється програмний продукт SQL відповідного виробника. Для розширення функціональних можливостей багато розробників, що дотримуються прийнятих стандартів, додають до стандартної мови SQL різні розширення. Слід зазначити, що стандарти

вимагають від будь-якої закінченої реалізації мови SQL наявності певних характеристик і у загальних рисах відбивають основні тенденції, які не лише призводять до сумісності між усіма конкуруючими реалізаціями, але і сприяють підвищенню значущості програмістів SQL і користувачів реляційних баз даних на сучасному ринку програмного забезпечення.

Усі конкретні реалізації мови дещо відрізняються одна від одної. В інтересах самих же виробників гарантувати, щоб їх реалізація відповідала сучасним стандартам ANSI в частині переносимості і зручності роботи користувачів. Проте кожна реалізація SQL містить удосконалення, що відповідають вимогам того або іншого сервера баз даних. Ці удосконалення або розширення мови SQL є додатковими командами і опціями, що є додатками до стандартного пакету і доступні в цій конкретній реалізації.

Нині мова SQL підтримується багатьма десятками СУБД різних типів, розроблених для найрізноманітніших обчислювальних платформ, починаючи від персональних комп'ютерів і закінчуючи мейнфреймами.

Усі мови маніпулювання даними, створені для багатьох СУБД до появи реляційних баз даних, були орієнтовані на операції з даними, представленими у вигляді логічних записів файлів. Зрозуміло, це вимагало від користувача детального знання організації зберігання цих і серйозних зусиль для вказівки тієї, які дані потрібні, де вони розміщуються і як їх отримати.

Мова SQL орієнтована на операції з даними, представленими у вигляді логічно взаємозв'язаних сукупностей таблиць-відношень. Найважливіша особливість його структур – орієнтація на кінцевий результат обробки даних, а не на процедуру цієї обробки. Процесор мови SQL сам визначає, де знаходяться дані, індекси і навіть які найбільш ефективні послідовності операцій слід використовувати для отримання результату, а тому вказувати ці деталі в запиті до бази даних не вимагається.

Реалізація в SQL концепції операцій, орієнтованих на табличне представлення даних, дозволила створити компактну мову з невеликим набором пропозицій. Мова SQL може використовуватися як для виконання запитів до даних, так і для побудови прикладних програм.

Основні категорії команд (або операторів) мови SQL призначені для виконання різних функцій, включаючи побудову об'єктів бази даних і маніпулювання ними, початкове завантаження даних в таблиці, оновлення і видалення існуючої інформації, виконання запитів до бази даних, управління доступом до неї і її загальне адміністрування. Досить часто ці категорії також називають мовами, тобто фактично їх можна вважати підмовами мови SQL.

Основні категорії команд мови SQL:

– DDL – мова визначення даних;

– DML – мова маніпулювання даними;

– DQL – мова запитів;

– DCL – мова управління даними;

– TCL – мова управління транзакціями.

Мова визначення даних (Data Definition Language, DDL) дозволяє створювати і змінювати структуру об'єктів бази даних, наприклад, створювати і видаляти таблиці. Основними командами мови DDL є наступні: CREATE, ALTER, DROP.

Мова маніпулювання даними (Data Manipulation Language, DML) використовується для маніпулювання інформацією усередині об'єктів реляційної бази даних за допомогою трьох основних команд: INSERT, UPDATE, DELETE.

Мова запитів (Data Query Language, DQL) найбільш відома користувачам реляційної бази даних, не дивлячись на те, що він включає всього одну команду SELECT. Ця команда разом зі своїми численними опціями і пропозиціями використовується для формування запитів до реляційної бази даних. Досить часто мову запитів DQL розглядають як складову DML.

Команди мови управління даними (Data Control Language, DCL) дозволяють управляти доступом до інформації, що знаходиться усередині бази даних. Як правило, вони використовуються для створення об'єктів, пов'язаних з доступом до даних, а також служать для контролю над розподілом привілеїв між користувачами. Команди управління даними наступні: GRANT (дозвіл на об'єкт), REVOKE (скасування дозволів і заборон на об'єкт), DENY (заборона на об'єкт).

Команди мови управління транзакціями (Transaction Control Language, TCL) дозволяють управляти транзакціями бази даних. Основними командами мови управління транзакціями є: BEGIN TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

### 1.1.2 Переваги мови SQL

Мова SQL є основою багатьох СУБД, оскільки відповідає за фізичну структуризацію і запис даних на диск, а також за читання даних з диска, дозволяє приймати SQL-запити від інших компонентів СУБД і призначених для користувача застосувань. Таким чином, SQL – потужний інструмент, який забезпечує користувачам, програмам і обчислювальним системам доступ до інформації, що міститься в реляційних базах даних.

Основні переваги мови SQL полягають в наступному:

– стандартність – як вже було сказано, використання мови SQL в програмних рішеннях стандартизоване міжнародними організаціями;

– незалежність від конкретних СУБД – усі поширені СУБД використовують SQL, оскільки реляційну базу даних можна перенести з однією СУБД на іншу з мінімальними доопрацюваннями;

– можливість перенесення з однієї обчислювальної системи на іншу – СУБД може бути орієнтована на різні обчислювальні системи, проте додатки, створені за допомогою SQL, допускають використання як для локальних БД, так і для великих розрахованих на багато користувачів систем;

– реляційна основа мови – SQL є мовою реляційних БД, тому він став популярним тоді, коли отримала широке поширення реляційна модель представлення даних. Таблична структура реляційної БД добре зрозуміла, а тому мова SQL проста для вивчення;

– можливість створення інтерактивних запитів – SQL забезпечує користувачам негайний доступ до даних, при цьому в інтерактивному режимі можна отримати результат запиту за дуже короткий час без написання складної програми;

– можливість програмного доступу до БД – мову SQL легко використовувати в додатках, яким необхідно звертатися до баз даних. Одні і ті ж оператори SQL вживаються як для інтерактивного, так і програмного доступу, тому частини програм, що містять звернення до БД, можна спочатку перевірити в інтерактивному режимі, а потім вбудовувати в програму;

– забезпечення різного представлення даних – за допомогою SQL можна представити таку структуру даних, що той або інший користувач бачитиме різні їх представлення. Крім того, дані з різних частин БД можуть бути скомбіновані і представлені у вигляді однієї простої таблиці, а значить, представлення придатні для посилення захисту БД і її налаштування під конкретні вимоги окремих користувачів;

– можливість динамічної зміни і розширення структури БД – мова SQL дозволяє маніпулювати структурою БД, тим самим забезпечуючи гнучкість з точки зору пристосованості БД до вимог предметної області, що змінюються;

– підтримка архітектури клієнт-сервер – мова SQL є одним з кращих засобів для реалізації застосунків на платформі клієнт-сервер. SQL служить сполучною ланкою між клієнтською системою, що взаємодіє з користувачем, і серверною системою, що управляє базою даних, дозволяючи кожному з них зосередитися на виконанні своїх функцій.

Будь-яка мова для роботи з базами даних повинна надавати користувачеві такі можливості:

– створювати бази даних і таблиці з повним описом їх структури;

– виконувати основні операції маніпулювання даними, зокрема, вставку, модифікацію і видалення даних з таблиць;

– виконувати прості і складні запити, що здійснюють перетворення даних.

Крім того, мова роботи з базами даних повинна вирішувати усі вказані вище завдання при мінімальних зусиллях з боку користувача, а структура і синтаксис його команд повинні бути досить прості і доступні для вивчення. І нарешті, віна має бути універсальною, тобто відповідати деякому визнаному стандарту, що дозволить використовувати один і той же синтаксис і структуру команд при переході від однієї СУБД до іншої. Мова SQL задовольняє практично усім цим вимогам.

Мова SQL є прикладом мови з орієнтацією на трансформацію, або ж мови, призначеної для роботи з таблицями з метою перетворення вхідних даних до необхідного вихідного виду. Вона включає тільки команди визначення і маніпулювання даними і не містить яких-небудь команд управління ходом обчислень. Подібні завдання повинні вирішуватися або за допомогою мов програмування або управління завданнями, або інтерактивно, в результаті дій, що виконуються самим користувачем. Унаслідок подібної незавершеності в плані організації обчислювального процесу мова SQL може використовуватися двома способами. Перший передбачає інтерактивну роботу, що полягає у введенні користувачем з терміналу окремих SQL-операторів. Другий полягає у впровадженні SQL-операторів в програми на процедурних мовах. Мова SQL відносно проста у вивченні. Оскільки це не процедурна мова, в нім необхідно вказувати, яка інформація має бути отримана, а не як її можна отримати. Інакше кажучи, SQL не вимагає вказівки методів доступу до даних. Як і більшість сучасних мов, він підтримує вільний формат запису операторів. Це означає, що при введенні окремі елементи операторів не пов'язані з фіксованими позиціями екрану. Мова SQL може використовуватися

широким колом фахівців, включаючи адміністраторів баз даних, прикладних програмістів і інших кінцевих користувачів.

Мова SQL – перша і доки єдина стандартна мова для роботи з базами даних, яка отримала досить широке поширення. Практично усі відомі розробники СУБД нині створюють свої продукти з використанням мови SQL або з SQL-інтерфейсом. В мову SQL зроблені величезні інвестиції як з боку розробників, так і з боку користувачів. Вона стала частиною архітектури застосунків, є стратегічним вибором багатьох великих і впливових організацій.

Але при цьому мова SQL має і певні недоліки. Найбільш відомими з них є такі:

1. SQL не є істинно реляційною мовою – творці реляційної моделі даних Едгар Кодд, Крістофер Дейт та їх прихильники вказують на такі недоліки SQL, що не цілком відповідають вимогам реляційної теорії, як рядки, що повторюються, невизначені значення (nulls), явна вказівка порядку колонок зліва направо, висока надмірність тощо.

2. Складність – хоча SQL і замислювався як засіб роботи кінцевого користувача, врешті-решт мова SQL стала настільки складною, що перетворилася на інструмент програміста.

3. Відступи від стандартів – незважаючи на наявність міжнародного стандарту ANSI SQL-92, багато компаній, що займаються розробкою СУБД, вносять зміни в мову SQL, що застосовується в СУБД, що розробляється, тим самим відступаючи від стандарту. Таким чином, з'являються специфічні для кожної конкретної СУБД діалекти мови SQL.

4. Складність роботи з ієрархічними структурами – ранні діалекти SQL більшості СУБД не пропонували способи маніпуляції деревоподібними структурами. Деякі постачальники СУБД пропонували свої рішення (наприклад, Oracle використовує вираз CONNECT BY). В даний час в ANSI стандартизована рекурсивна конструкція WITH з діалекту SQL DB2.

### 1.1.3 Запис SQL-операторів

Для успішного вивчення мови SQL необхідно привести короткий опис структури SQL-операторів і нотації, які використовуються для визначення формату різноманітних конструкцій мови. Оператор SQL складається із зарезервованих слів, а також із слів, визначуваних користувачем. Зарезервовані слова є постійною частиною мови SQL і мають фіксоване значення. Їх слід записувати в точності так, як це встановлено, не можна розбивати на частини для перенесення з одного рядка на іншу. Слова, визначувані користувачем, задаються ним самим (відповідно до синтаксичних правил) і є ідентифікаторами або іменами різних об'єктів бази даних. Слова в операторові розміщуються також відповідно до встановлених синтаксичних правил.

Ідентифікатори мови SQL призначені для позначення об'єктів у базі даних і є іменами таблиць, представлень, стовпців і інших об'єктів бази даних. Символи, які можуть використовуватися в створюваних користувачем ідентифікаторах мови SQL, мають бути визначені як набір символів. Стандарт SQL задає набір символів, який використовується за умовчанням, - він включає рядкові і прописні букви латинського алфавіту ( A - Z, a - z ), цифри ( 0-9 ) і символ підкреслення ( _ ). На формат ідентифікатора накладаються наступні обмеження:

- ідентифікатор може мати довжину до 128 символів;
- ідентифікатор повинен починатися з букви;
- ідентифікатор не може містити пропуски.

<ідентифікатор>::=<буква>

{<буква>|<цифра>}[,..n]

Більшість компонентів мови не чутлива до регістра. Оскільки у мови SQL вільний формат, окремі SQL-операторі і їх послідовності матимуть більше читаний вигляд при використанні відступів і вирівнювання.

Мова, в термінах якої дається опис мови SQL, називається метамовою. Синтаксичні визначення зазвичай задають за допомогою

спеціальної металінгвістичної символіки, званою формою Бекуса-Науера (БНФ). Прописні букви використовуються для запису зарезервованих слів і повинні вказуватися в операторах точно так, як це буде показано. Рядкові букви вживаються для запису слів, визначуваних користувачем. Вживані в нотації БНФ символи і їх позначення показані в таблиці 1.1.

Таблиця 1.1 – Вживані в нотації БНФ символи і їх позначення

| Символ | Зміст |
|---|---|
| ::= | Рівно за визначенням |
| \| | Необхідність вибору одного з декількох приведених значень |
| <.> | Описана за допомогою метамови структура мови |
| {.} | Обов'язковий вибір деякої конструкції із списку |
| [.] | Необов'язковий вибір деякої конструкції із списку |
| [,.n] | Необов'язкова можливість повторення конструкції від нуля до декількох разів |

### 1.1.4 Типи даних мови SQL, визначені стандартом

Дані – це сукупна інформація, що зберігається у базі даних у вигляді одного з декількох різних типів. За допомогою типів даних встановлюються основні правила для даних, що містяться в конкретному стовпці таблиці, у тому числі розмір пам'яті, що виділяється для них.

У мові SQL є шість скалярних типів даних, визначених стандартом. Їх стислий опис представлений в таблиці 1.2.

Символьні дані складаються з послідовності символів, що входять у визначений розробниками СУБД набір символів. Оскільки набори символів є специфічними для різних діалектів мови SQL, перелік символів, які можуть входити до складу значень даних символьного типу, також залежить від конкретної реалізації. Найчастіше використовуються набори символів ASCII і EBCDIC.

Таблиця 1.2 – Типи даних мови SQL

| Тип даних | Оголошення |
|---|---|
| Символьний | CHAR \| VARCHAR |
| Бітовий | BIT \| BIT VARYING |
| Точні числа | NUMERIC \| DECIMAL \| INTEGER \| SMALLINT |
| Закруглені числа | FLOAT \| REAL \| DOUBLE PRECISION |
| Дата/час | DATE \| TIME \| TIMESTAMP |
| Інтервал | INTERVAL |

Для визначення даних символьного типу використовується наступний формат:

<символьний_тип>::=
{ CHARACTER [ VARYING][довжина] | [CHAR |
VARCHAR][довжина]}

При визначенні стовпця з символьним типом даних параметр довжина застосовується для вказівки максимальної кількості символів, які можуть бути поміщені в цей стовпець (за умовчанням набуває значення 1). Символьний рядок може бути визначений як що має фіксовану або змінну (VARYING) довжину. Якщо рядок визначений з фіксованою довжиною значень, то при введенні в неї меншої кількості символів значення доповнюється до вказаної довжини пропусками, що додаються справа. Якщо рядок визначений зі змінною довжиною значень, то при введенні в неї меншої кількості символів у базі даних будуть збережені тільки введені символи, що дозволить досягти певної економії зовнішньої пам'яті.

Бітовий тип даних використовується для визначення бітових рядків, тобто послідовності двійкових цифр (бітів), кожна з яких може мати значення або 0, або 1. Дані бітового типу визначаються за допомогою наступного формату:

<бітовий_тип>::=
BIT [VARYING][довжина]

Тип точних числових даних застосовується для визначення чисел, які мають точне представлення, тобто числа складаються з цифр, необов'язкової десяткової точки і необов'язкового символу знаку. Дані точного числового типу визначаються точністю і завдовжки дробовій частині. Точність задає загальну кількість значущих десяткових цифр числа, в яке входить довжина як цілої частини, так і дробовою, але без урахування самої десяткової точки. Масштаб вказує кількість дробових десяткових розрядів числа.

<фіксований_тип>::=
{NUMERIC[точність[,масштаб]]|{DECIMAL|DEC}
[точність[, масштаб]]
| {INTEGER |INT}| SMALLINT}

Типи NUMERIC і DECIMAL призначені для зберігання чисел в десятковому форматі. За умовчанням довжина дробової частини дорівнює нулю, а точність, що приймається за умовчанням, залежить від реалізації. Тип INTEGER (INT) використовується для зберігання великих позитивних або негативних цілих чисел. Тип SMALLINT – для зберігання невеликих позитивних або негативних цілих чисел; в цьому випадку витрата зовнішньої пам'яті істотно скорочується.

Тип закруглених чисел застосовується для опису даних, які не можна точно представити в комп'ютері, зокрема дійсних чисел. Закруглені числа або числа з плаваючою точкою представляються в науковій нотації, при якій число записується за допомогою мантиси, помноженої на певну міру десяти (порядок), наприклад:10E3, +5.2E6, -0.2E-4 . Для визначення даних речового типу використовується формат:

<речовий_тип>::=
{ FLOAT [точність]| REAL |
DOUBLE PRECISION}

Параметр точність задає кількість значущих цифр мантиси. Точність типів REAL і DOUBLE PRECISION залежить від конкретної реалізації.

Тип даних "дата/час" використовується для визначення моментів часу з деякою встановленою точністю. Стандарт SQL підтримує наступний формат:

<тип_дати/часу>::=
{DATE | TIME[точність][WITH TIME ZONE]|
TIMESTAMP[точність][WITH TIME ZONE]}

Тип даних DATE використовується для зберігання календарних дат, що включають поля YEAR (рік), MONTH (місяць) і DAY (день). Тип даних TIME – для зберігання відміток часу, що включають поля HOUR (години), MINUTE (хвилини) і SECOND (секунди). Тип даних TIMESTAMP – для спільного зберігання дати і часу. Параметр точність задає кількість дробових десяткових знаків, що визначають точність збереження значення в полі SECOND. Якщо цей параметр опускається, за умовчанням його значення для стовпців типу TIME набуває рівним нулю (тобто зберігаються цілі секунди), тоді як для полів типу TIMESTAMP він приймається рівним 6. Наявність ключового слова WITH TIME ZONE визначає використання полів TIMEZONE HOUR і TIMEZONE MINUTE, тим самим задаються година і хвилини зміщення зонального часу по відношенню до універсального координатного часу (Грінвічського часу).

Дані типу INTERVAL використовуються для представлення періодів часу.

Поняття домену. Домен – це набір допустимих значень для одного або декількох атрибутів. Якщо в таблиці бази даних або в декількох таблицях є присутніми стовпці, що мають одні і ті ж характеристики, можна описати тип такого стовпця і його поведінку через домен, а потім поставити у відповідність кожному з однакових стовпців ім'я домена. Домен визначає усі потенційні значення, які можуть бути присвоєні атрибуту.

Стандарт SQL дозволяє визначити домен за допомогою наступного оператора:

<визначення_домена>::=
CREATE DOMAIN ім'я_домена [AS]
тип_даних
[ DEFAULT значення]
[ CHECK (допустимі_значення)]

Кожному створюваному домену привласнюється ім'я, тип даних, значення за умовчанням і набір допустимих значень. Слід зазначити, що приведений формат оператора являється неповним. Тепер при створенні таблиці можна вказати замість типу даних ім'я домена.

Видалення доменів з бази даних виконується за допомогою оператора:

DROP DOMAIN ім'я_домена [ RESTRICT |
CASCADE]

У разі вказівки ключового слова CASCADE будь-які стовпці таблиць, створені з використанням домена, що видаляється, будуть автоматично змінені і описані як вміщуючі дані того типу, який був вказаний у визначенні домена, що видалявся.

Альтернативою доменам в середовищі СУБД Microsoft SQL Server є типи даних, визначені користувачем.

## 1.2 СУБД Microsoft SQL Server – призначення та основні особливості

### 1.2.1 Загальна стисла характеристика СУБД Microsoft SQL Server

СУБД Microsoft (MS) SQL Server є програмним комплексом, що складається з трьох основних компонентів:

– ядро СУБД – Database Engine;

– служби Analysis Services;

– служби Reporting Services.

Компанія Microsoft пропонує візуалізацію архітектури СУБД Microsoft SQL Server, представлену на рисунку 1.1.


Рисунок 1.1 – Архітектура СУБД Microsoft SQL Server

Компонент Database Engine є основною службою для зберігання, обробки та забезпечення безпеки даних. Цей компонент забезпечує керований доступ до ресурсів та швидку обробку транзакцій, що дозволяє

використовувати його навіть у найвибагливіших корпоративних застосунках обробки даних.

Компонент Database Engine використовується для створення реляційних баз даних для оперативної обробки транзакцій (On-Line Transaction Processing – OLTP) або оперативної аналітичної обробки даних (On-Line Analytical Processing – OLAP).

OLAP – технологія оперативної аналітичної обробки даних, що використовує методи та засоби для збирання, зберігання та аналізу багатовимірних даних з метою підтримки прийняття рішень.

Можливості компонента Database Engine включають створення таблиць для зберігання даних і об'єктів баз даних, таких як індекси, представлення та процедури, що зберігаються, для швидкого перегляду, ефективного зберігання, управління даних їх захистом. Середовище SQL Server Management Studio надає інтерфейс користувача для управління об'єктами баз даних, а для фіксації подій сервера можна використовувати програму SQL Server Profiler.

Під час установлення Microsoft SQL Server створюються чотири системні бази даних: master, model, msdb, tempdb. У базі даних master зберігається вся інформація відносно конфігурації і функціонування MS SQL Server. Вона містить відомості про всі облікові записи користувачів, про інші бази даних, а також про параметри сервера. База даних model є шаблоном для баз даних, що створюються, і завжди має бути в системі. Під час створення нової користувацької бази даних MS SQL Server створює копію бази даних model. У базі даних msdb зберігається інформація про планування задач (jobs) і подій (alerts), а також про організацію роботи операторів, які отримують повідомлення. База даних tempdb, яку формує MS SQL Server, призначена для збереження тимчасових таблиць. База даних tempdb є глобальним ресурсом, який автоматично доступний усім користувачам і створюється кожного разу під час запуску MS SQL Server. Усі тимчасові таблиці tempdb автоматично вилучаються, коли користувач відключається від сервера.

Кожна база даних MS SQL Server складається з кількох об'єктів, які використовуються для збереження, організації та обробки даних. Об'єктами бази даних MS SQL Server є таблиці, індекси, представлення, обмеження, правила, значення по замовчуванню, тригери, процедури і типи даних.

Таблиця є основним об'єктом, який зберігає всі дані, що зберігаються у базі даних. У MS SQL Server є таблиці двох типів – системні і користувацькі. У системних таблицях зберігається інформація про MS SQL Server і його об'єкти, а в користувацьких – інформація з первинних документів та інших джерел. Імена всіх системних таблиць починаються з префікса sys. До складу таблиць входить файл транзакцій, який утворюється автоматично під час створення бази даних і призначається для забезпечення цілісності і відновлення бази даних у разі виникнення помилок (якщо в базу даних вносять зміни, то в журналі транзакцій зберігаються нові і старі значення рядків таблиці).

MS SQL Server використовує діалект мови структурованих запитів SQL – мову Transact-SQL, за допомогою якої описують структуру баз даних та виконують операції над даними.

Служби Microsoft SQL Server Analysis Services підтримують роботу з багатовимірними даними, здійснюють процес OLAP, надають можливість проектувати та створювати багатовимірні структури даних, отримані з різних джерел даних, та керувати цими багатовимірними об'єктами.

Служби Analysis Services дозволяють аналітикам працювати з багатовимірними структурами, що містять детальні статистичні дані в єдиній уніфікованій логічній моделі, реалізуючи модель сховищ даних.

Сховище даних – це предметно-орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію, та організований з метою підтримки прийняття рішень та виконання завдань бізнес-аналітики.

Служби Analysis Services забезпечують можливість швидкого, доступного для розуміння користувача низхідного аналізу великої кількості даних, виконуючи операції «Зріз», «Обертання», «Консолідація»

та «Деталізація» багатовимірних даних. Результати аналізу можуть надаватися користувачам у текстовому чи графічному форматі, враховуючи національні стандарти (позначення десяткової частини речового числа через точку чи кому; формат дати; відображення грошових величин тощо).

Завдяки службам Analysis Services можна проектувати, створювати та візуалізувати моделі інтелектуального аналізу даних. Розмаїття стандартних алгоритмів інтелектуального аналізу даних дозволяє створювати такі моделі з урахуванням інших джерел даних.

Служби Analysis Services містять такі засоби для надання можливостей інтелектуального аналізу даних:

– конструктор інтелектуального аналізу даних, призначений для створення та перегляду моделей інтелектуального аналізу даних, управління моделями та складання прогнозів за допомогою побудованих моделей;

– мова розширень інтелектуального аналізу даних, яку можна використовувати для управління моделями інтелектуального аналізу даних та створення складних прогнозуючих запитів.

Для пошуку закономірностей у даних використовуються методи Data Mining, які забезпечують дослідження та виявлення машинними алгоритмами в сирих даних прихованих знань, які раніше не були відомі, нетривіальні, практично корисні та доступні для інтерпретації людиною.

Служби SQL Server Reporting Services представляють серверну платформу, що надає засоби створення корпоративних звітів з підтримкою веб-інтерфейсу, які дозволяють включати до звітів дані з різних джерел, публікувати звіти у різноманітних форматах, централізовано керувати безпекою та підписками.

Служби Reporting Services пропонують широкий вибір готових до використання коштів та служб для створення, розгортання та управління звітами організації, а також функції програмування, що дозволяють розширити та налаштувати функціональність звітів. Завдяки API-

інтерфейсам, розробники можуть проводити інтеграцію і розширювати можливості обробки даних і роботу зі звітами в програмах користувача. Інструменти служб Reporting Services працюють в оточенні Microsoft Visual Studio та інтегровані з компонентами SQL Server.

За допомогою служб Reporting Services можна створити інтерактивні, табличні, графічні звіти та звіти вільної форми з реляційних, багатовимірних та XML-джерел даних. Є можливість публікувати звіти, планувати їхню обробку або здійснювати доступ до звітів на вимогу.

При проектуванні звітів можна вибрати мобільний тип звітів або звіт з розбиттям на сторінки, заздалегідь підготовлені для відображення на пристроях, що переносяться, включаючи розширену візуалізацію даних, графіків, діаграм, карт і так далі.

Служби Integration Services – це платформа, на якій створюються високопродуктивні рішення щодо інтеграції даних, а також пакети для зберігання даних, які дають змогу вилучати, перетворювати та завантажувати дані.

Служби Integration Services створені для того, щоб вирішувати складні бізнес-завдання, виконуючи такі функції, як копіювання та завантаження файлів, надсилання повідомлень електронною поштою у відповідь на події, оновлення сховищ даних, очищення та інтелектуальний аналіз даних, а також за допомогою цієї служби здійснюється управління об'єктами та даними SQL Server. Передбачається робота пакетів як окремо, так і для вирішення складних бізнес-задач. Служби Integration Services отримують і перетворюють дані з декількох джерел, таких як файли даних XML, неструктуровані файли, джерела реляційних даних, а потім завантажують отриману інформацію в реляційні об'єкти.

Служби Integration Services містять велику кількість вбудованих завдань та перетворень, в них надано безліч засобів для побудови пакетів, також ця служба займається керуванням пакетами та їх виконанням. За допомогою графічних інструментів служб Integration Services можна створювати готові рішення без жодного рядка коду, існує спосіб

програмування об'єктної моделі Integration Services для створення пакетів і створення в програмному коді завдань користувача та інших об'єктів пакета.

Реплікація (Replication) – це процес, що є копіюванням та перенесенням інформації між екземплярами (репліками) бази даних, а також синхронізацію цих баз даних для забезпечення узгодженості даних. Технології реплікації дають можливість розміщення даних у різних місцях, а також забезпечують доступ до них користувачів, які віддалено підключені. Реплікація дозволяє підключитися через комутовані та бездротові з'єднання, а також Інтернет, використовуючи локальні або глобальні мережі.

Реплікація транзакцій зазвичай використовується у сценаріях «сервер-сервер», для яких необхідна висока пропускна здатність, у тому числі покращення масштабованості та доступності, зберігання та протоколювання даних, інтеграція даних з декількох сайтів, об'єднання різноманітних даних, автономна обробка пакетів. Реплікація злиттям розроблена переважно для мобільних додатків чи розподілених серверних додатків, у яких можливе виникнення конфліктів даних. Звичайні сценарії включають обмін даними з мобільними користувачами, клієнтські програми точки продажу (Point-of-Sale – POS) та інтеграцію даних з декількох сайтів. Реплікація моментальних знімків використовується для забезпечення початкового набору даних для реплікації транзакцій і реплікації злиттям; вона також може застосовуватися за необхідності виконання повного оновлення даних. Маючи в своєму розпорядженні ці три типи реплікації, MS SQL Server являє собою потужну і гнучку систему для синхронізації даних рівня підприємства.

Крім реплікації, можна синхронізувати бази даних за допомогою служб Microsoft Sync Framework та Sync Services for ADO.NET. Служби Sync Services for ADO.NET надають інтуїтивно зрозумілий, гнучкий API, який можна використовувати для побудови програм, призначених для сценаріїв поза мережею та спільних сценаріїв.

Компонент Service Broker створений для допомоги розробникам, які створюють безпечні масштабовані програми баз даних. Це нова технологія компонента Database Engine, який є платформою взаємодії користувачів один з одним на основі обміну повідомленнями, в той час як незалежні компоненти додатків виступають єдиним цілим. У компоненті Service Broker включена інфраструктура асинхронного програмування, яка може використовуватися як додатками в межах однієї бази даних або екземпляра, так і розподіленими застосунками.

Компонент SQL Server Service Broker забезпечує власну підтримку компонента SQL Server Database Engine для програм обміну повідомленнями та програм з чергами повідомлень. Це полегшує розробникам створення складних застосунків, що використовують компоненти Database Engine для зв'язку між різнорідними базами даних. Компонент Service Broker полегшує розробникам процес створення розподілених та надійних застосунків.

Розробники програм, які використовують компонент Service Broker, можуть розподіляти робоче навантаження між кількома базами даних без програмування складної взаємодії та створення внутрішнього обміну повідомленнями. Це скорочує розробку та перевірку, тому що компонент Service Broker забезпечує взаємодію в контексті діалогу. Крім того, це підвищує продуктивність. За допомогою компонента Service Broker виконуються всі завдання відповідно до транзакцій, які гарантують надійність та технічну узгодженість.

СУБД MS SQL Server є дуже відомою та популярною серед інших СУБД. На рисунку 1.2 наведено її місце у загальному рейтингу СУБД, а на рисунку 1.3 – місце у рейтингу СУБД, які є орієнтованими на роботу із базами даних реляційного типу. Ці відомості запозичені з веб-ресурсів https://db-engines.com/en/ranking та https://db-engines.com/en/ranking/relational+dbms відповідно.

426 systems in ranking, November 2025

| Rank Nov 2025 | Rank Oct 2025 | Rank Nov 2024 | DBMS | Database Model | Score Nov 2025 | Score Oct 2025 | Score Nov 2024 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1239.78 | +27.01 | -77.23 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 865.82 | -13.84 | -151.98 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 718.87 | +3.82 | -80.94 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 651.36 | +8.16 | -2.97 |
| 5. | 5. | 5. | MongoDB | Multi-model | 371.68 | +3.66 | -29.25 |
| 6. | 6. | ↑ 7. | Snowflake | Relational | 197.84 | -0.81 | +55.35 |
| 7. | 7. | ↓ 6. | Redis | Key-value, Multi-model | 145.09 | +2.76 | -3.55 |
| 8. | 8. | ↑ 14. | Databricks | Multi-model | 131.50 | +2.70 | +45.04 |
| 9. | 9. | 9. | IBM Db2 | Relational, Multi-model | 119.28 | -3.10 | -2.47 |
| 10. | 10. | ↓ 8. | Elasticsearch | Multi-model | 113.97 | -2.69 | -17.67 |

Рисунок 1.2 – Загальний рейтинг СУБД

include secondary database models

165 systems in ranking, November 2025

| Rank Nov 2025 | Rank Oct 2025 | Rank Nov 2024 | DBMS | Database Model | Score Nov 2025 | Score Oct 2025 | Score Nov 2024 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1239.78 | +27.01 | -77.23 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 865.82 | -13.84 | -151.98 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 718.87 | +3.82 | -80.94 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 651.36 | +8.16 | -2.97 |
| 5. | 5. | 5. | Snowflake | Relational | 197.84 | -0.81 | +55.35 |
| 6. | 6. | ↑ 9. | Databricks | Multi-model | 131.50 | +2.70 | +45.04 |
| 7. | 7. | ↓ 6. | IBM Db2 | Relational, Multi-model | 119.28 | -3.10 | -2.47 |
| 8. | 8. | ↓ 7. | SQLite | Relational | 104.19 | -0.36 | +4.71 |
| 9. | 9. | ↑ 10. | MariaDB | Relational, Multi-model | 87.36 | -0.41 | +4.66 |
| 10. | 10. | ↓ 8. | Microsoft Access | Relational | 78.29 | -2.50 | -13.03 |
| 11. | 11. | 11. | Microsoft Azure SQL Database | Relational, Multi-model | 76.38 | +0.90 | -0.15 |

Рисунок 1.3 – Рейтинг СУБД, які є орієнтованими на роботу із базами даних реляційного типу

### 1.2.2 Типи даних, використовувані в СУБД Microsoft SQL Server

#### 1.2.2.1 Системні типи даних

Один з основних моментів процесу створення таблиці - визначення типів даних для її полів. Тип даних поля таблиці визначає тип інформації,

яка розміщуватиметься в цьому полі. Поняття типу даних в MS SQL Server повністю адекватно поняттю типу даних в сучасних мовах програмування. MS SQL Server підтримує велике число різних типів даних: текстові, числові, двійкові та інші (рисунок 1.4).

| | | | |
|---|---|---|---|
| image | smalldatetime | bit | binary |
| text | real | decimal | char |
| uniqueidentifier | money | numeric | timestamp |
| tinyint | datetime | smallmoney | nvarchar |
| smallint | float | varbinary | nchar |
| int | ntext | varchar | sysname |

Рисунок 1.4 – Основні типи даних СУБД MS SQL Server

Зробимо стислий огляд типів даних.

Для зберігання символьної інформації використовуються символьні типи даних, до яких відносяться CHAR (довжина), VARCHAR (довжина), NCHAR (довжина), NVARCHAR (довжина). Останні два призначені для зберігання символів Unicode. Максимальне значення довжини обмежене 8000 знаками ( 4000 – для символів Unicode).

Зберігання символьних даних великого об'єму (до 2 Гб) здійснюється за допомогою текстових типів даних TEXT і NTEXT.

До цілочисельних типів даних відносяться INT (INTEGER), SMALLINT, TINYINT, BIGINT. Для зберігання даних цілочисельного типу використовується, відповідно, 4 байти (діапазон від –231 до 231-1), 2 байти (діапазон від –215 до 215-1), 1 байт (діапазон від 0 до 255) або 8 байт (діапазон від –263 до 263-1). Об'єкти і вирази цілочисельного типу можуть застосовуватися у будь-яких математичних операціях.

Числа, у складі яких є десяткова точка, називаються нецілочисельними. Нецілочисельні дані розділяються на два типи – десяткові і приблизні.

До десяткових типів даних відносяться типи DECIMAL [(точність[,масштаб])] чи DEC і NUMERIC [(точність[,масштаб])]. Типи

даних DECIMAL і NUMERIC дозволяють самостійно визначити формат точності числа з плаваючою комою. Параметр точність вказує максимальна кількість цифр даних цього типу (до і після десяткової точки в сумі), що вводяться, а параметр масштаб - максимальна кількість цифр, розташованих після десяткової точки. У звичайному режимі сервер дозволяє вводити не більше 28 цифр, використовуваних в типах DECIMAL і NUMERIC (від 2 до 17 байт).

До приблизних типів даних відносяться FLOAT (точність до 15 цифр, 8 байт) і REAL (точність до 7 цифр, 4 байти). Ці типи представляють дані у форматі з плаваючою комою, тобто для представлення чисел використовується мантиса і порядок, що забезпечує однакову точність обчислень незалежно від того, наскільки мало або велике значення.

Для зберігання інформації про дату і час призначені такі типи даних, як DATETIME і SMALLDATETIME, що використовують для представлення дати і часу 8 і 4 байти відповідно.

Типи даних MONEY і SMALLMONEY роблять можливим зберігання інформації грошового типу ; вони забезпечують точність значень до 4 знаків після коми і використовують 8 і 4 байти відповідно.

Тип даних BIT дозволяє зберігати один біт, який набуває значень 0 або 1.

У середовищі MS SQL Server реалізований ряд спеціальних типів даних.

Тип даних TIMESTAMP застосовується в якості індикатора зміни версії рядка в межах бази даних.

Тип даних UNIQUEIDENTIFIER використовується для зберігання глобальних унікальних ідентифікаційних номерів.

Тип даних SYSNAME призначений для ідентифікаторів об'єктів.

Тип даних SQL_VARIANT дозволяє зберігати значення будь-якого з підтримуваних SQL Server типів даних за винятком TEXT, NTEXT, IMAGE і TIMESTAMP.

Тип даних TABLE, подібно до тимчасових таблиць, забезпечує зберігання набору рядків, призначених для наступної обробки. Тип даних TABLE може застосовуватися тільки для визначення локальних змінних і значень, що повертають призначені користувачами функції.

Тип даних CURSOR потрібний для роботи з такими об'єктами, як курсори, і може бути затребуваний тільки для змінних і параметрів процедур, що зберігаються. Курсори SQL Server є механізмом обміну даними між сервером і клієнтом. Курсор дозволяє клієнтським застосуванням працювати не з повним набором даних, а лише з однією або декількома рядками.

**1.2.2.2 Створення користувацького типу даних**

У системі SQL -сервера є підтримка призначених для користувача типів даних. Вони можуть використовуватися при визначенні якого-небудь специфічного або часто вживаного формату.

Створення користувацького типу даних здійснюється виконанням системної процедури:

```
sp_addtype [@typename=]type,[@phystype=]
                system_data_type
                  [,[@nulltype=]'null_type']
```

Тип даних system_data_type вибирається з таблиці, яку наведено на рисунку 1.5.

Видалення користувацького типу даних відбувається в результаті виконання процедури sp_droptype type.

Отримати список усіх типів даних, включаючи призначені для користувача, можна з системної таблиці systypes:

```
SELECT * FROM  systypes
```

| | | | |
|---|---|---|---|
| image | smalldatetime | decimal | bit |
| text | real | 'decimal[(p[,s])]' | 'binary(n)' |
| uniqueidentifier | datetime | numeric | 'char(n)' |
| smallint | float | 'numeric[(p[,s])]' | 'nvarchar(n)' |
| int | 'float(n)' | 'varbinary(n)' | |
| | ntext | 'varchar(n)' | 'nchar(n)' |

Рисунок 1.5 – Типи даних system_data_type

Разом з типами даних базовими поняттями при роботі з мовою SQL в середовищі MS SQL Server є вирази, оператори, змінні та конструкції, що управляють.

Вирази є комбінацією ідентифікаторів, функцій, знаків логічних і арифметичних операцій, констант і інших об'єктів. Вираження може бути використане в якості аргументу в командах, процедурах, що зберігаються, або запитах.

Вираз складається з операндів (власне даних) і операторів (знаків операцій, які виконують над операндами). Операндами можуть виступати константи, змінні, імена стовпців, функції, підзапити.

Оператори – це знаки операцій над одним або декількома виразами для створення нового вираження. Серед операторів можна виділити унарні оператори, оператори присвоєння, арифметичні оператори, строкові оператори, оператори порівняння, логічні оператори, бітові оператори.

У середовищі SQL Server існує декілька способів передання даних між командами. Один з них – передача даних через локальні змінні. Перш ніж використовувати яку-небудь змінну, її слід оголосити. Оголошення змінної виконується командою DECLARE, що має наступний формат:

```
DECLARE {@ім'я_змінної тип_даних }
        [,..n]
```

Значення змінної можна призначити за допомогою команд SET і SELECT. За допомогою команди SELECT змінній можна призначити не лише конкретне значення, але і результат обчислення виразу.

### 1.2.3 Основні об'єкти структури бази даних в СУБД Microsoft SQL Server

Логічна структура бази даних визначає структуру таблиць, взаємини між ними, список користувачів, збережені процедури, правила, умовчання і інші об'єкти бази даних. До основних об'єктів бази даних MS SQL Server відносяться об'єкти, представлені в таблиці 1.3.

Таблиця 1.3 – Основні об'єкти бази даних MS SQL Server

| Об'єкт | Опис |
|---|---|
| Tables | Таблиці бази даних, в яких зберігаються власне дані |
| Views | Перегляди (віртуальні таблиці) для відображення даних з таблиць |
| Stored Procedures | Збережені процедури |
| Triggers | Тригери - спеціальні Збережені процедури, викликаються при зміні даних в таблиці |
| User Defined function | Створювані користувачем функції |
| Indexes | Індекси – додаткові структури, покликані підвищити продуктивність роботи з даними |
| User Defined Data Types | Визначувані користувачем типи даних |
| Keys | Ключі - один з видів обмежень цілісності даних |
| Constraints | Обмеження цілісності - об'єкти для забезпечення логічної цілісності даних |
| Users | Користувачі, що мають доступ до бази даних |
| Roles | Ролі, що дозволяють об'єднувати користувачів в групи |
| Rules | Правила бази даних, що дозволяють контролювати логічну цілісність даних |
| Defaults | Умовчання або стандартні установки бази даних |

Стисло розглянемо особливості основних об'єктів баз даних.

Усі дані в базі даних містяться в об'єктах, званих таблицями. Таблиці є сукупністю відомостей про об'єкти, явища, процеси реального світу. Ніякі інші об'єкти не зберігають дані, але вони можуть звертатися до даних в таблиці. Таблиці в базі даних MS SQL Server мають таку ж структуру, що і таблиці усіх інших реляційних СУБД і містять:

– рядки, де кожен рядок (чи запис) є сукупністю атрибутів (властивостей) конкретного екземпляра об'єкту;

– стовпці, де кожен стовпець (поле) є атрибутом або сукупністю атрибутів. Поле рядка є мінімальним елементом таблиці. Кожен стовпець в таблиці має певне ім'я, тип даних і розмір.

Представленнями (переглядами) називають віртуальні таблиці, вміст яких визначається запитом. Подібно до реальних таблиць, представлення містять іменовані стовпці і рядки з даними. Для кінцевих користувачів представлення виглядає як таблиця, але насправді воно не містить даних, а лише представляє дані, розташовані в одній або декількох таблицях. Інформація, яку бачить користувач через представлення, не зберігається у базі даних як самостійний об'єкт.

Збережені процедури, є групою команд (SQL та/або інших), об'єднаних в один модуль. Така група команд компілюється і виконується як єдине ціле.

Тригерами називається спеціальний клас процедур, що зберігаються. Тригери автоматично запускаються при додаванні, зміні або видаленні даних з таблиці.

Функції в мовах програмування – це конструкції, що містять часто виконуваний код. Функція виконує які-небудь дії над даними і повертає деяке значення.

Індекс – структура, пов'язана з таблицею або представленням і призначена для прискорення пошуку інформації в них. Індекс визначається для одного або декількох стовпців, званих індексованими стовпцями. Він містить відсортовані значення індексованого стовпця або стовпців з посиланнями на відповідний рядок початкової таблиці або представлення.

Підвищення продуктивності досягається за рахунок сортування даних. Використання індексів може істотно підвищити продуктивність пошуку, проте для зберігання індексів потрібний додатковий простір у базі даних.

Призначені для користувача типи даних – це типи даних, які створює користувач на основі системних типів даних, коли в декількох таблицях необхідно зберігати однотипні значення; причому треба гарантувати, що стовпці в таблиці матимуть однаковий розмір, тип даних і чутливість до значень NULL.

Обмеження цілісності – механізм, що забезпечує автоматичний контроль відповідності даних встановленим умовам (чи обмеженням). Обмеження цілісності мають пріоритет над тригерами, правилами і значеннями за умовчанням. До обмежень цілісності відносяться: обмеження на значення NULL, перевірочні обмеження, обмеження унікальності (унікальний ключ), обмеження первинного ключа і обмеження зовнішнього ключа. Останні три обмеження тісно пов'язано з поняттям ключів.

Правила використовуються для обмеження значень, що зберігаються в стовпці таблиці або в призначеному для користувача типі даних. Вони існують як самостійні об'єкти бази даних, які зв'язуються із стовпцями таблиць і призначеними для користувача типами даних. Контроль значень даних може бути реалізований і за допомогою обмежень цілісності.

Умовчання – самостійний об'єкт бази даних, що представляє значення, яке буде присвоєно елементу таблиці при вставці рядка, якщо в команді вставки явно не вказано значення для цього стовпця.

### 1.2.4 Мова Transact-SQL

Transact-SQL (або T-SQL) – процедурне розширення мови SQL, створене компанією Microsoft (для Microsoft SQL Server) і Sybase (для Sybase ASE) (https://uk.wikipedia.org/wiki/Transact-SQL).

Transact-SQL був розширений наступними додатковими можливостями, такими як:

– керуючі оператори;

– локальні і глобальні змінні;

– додаткові функції для обробки рядків, дат, математичні функції тощо;

– підтримка аутентифікації Microsoft Windows та інші.

Мова Transact-SQL є ключем до використання MS SQL Server. Всі застосунки, які взаємодіють з екземпляром MS SQL Server, незалежно від їхньої реалізації і інтерфейсу користувача, відправляють з сервера інструкції Transact-SQL.

Ключові особливості T-SQL:

1. Процедурна логіка: додає оператори керування обчислювальним процесом (такі, як IF...ELSE, WHILE), що не є частиною стандартної мови SQL.

2. Змінні: дозволяє використовувати локальні та глобальні змінні для зберігання даних під час виконання запитів.

3. Розширені функції: велика бібліотека функцій для маніпуляції рядками, датами, математичними обчисленнями.

4. Інтеграція з Windows: підтримка автентифікації Microsoft Windows.

5. Робота з даними: використовується для створення, зміни, видалення та управління даними в базах даних SQL Server.

Для чого використовується T-SQL:

1. Розробка баз даних: створення та керування схемами баз даних.

2. Аналіз даних: складні запити та трансформація даних.

3. Автоматизація: написання скриптів для виконання регулярних завдань.

4. Оптимізація: створення процедур та функцій для підвищення продуктивності.

Отже, Transact-SQL є потужним інструментом для розробників та адміністраторів баз даних, що дозволяє виходити за межі базового SQL і

створювати комплексні рішення для управління даними в середовищі Microsoft SQL Server.

### 1.2.5 Застосунок SQL Server Management Studio

SQL Server Management Studio (SSMS) – це інтегроване графічне середовище (Integrated Development Environment – IDE) від компанії Microsoft для керування, конфігурації та адміністрування всіх компонентів Microsoft SQL Server, що надає розробникам та адміністраторам потужний інструментарій для роботи з базами даних, написання запитів (мовою T-SQL), керування об'єктами та моніторингу серверів. Це безкоштовний інструмент, що поєднує редактор SQL-коду та інструменти адміністрування в одному застосунку для середовища Windows (https://learn.microsoft.com/en-us/ssms/).

Основні функції та можливості SSMS:

1. Управління об'єктами: перегляд, створення, зміна та видалення об'єктів бази даних (таблиці, представлення, процедури) через візуальний навігатор Object Explorer.

2. Редактор запитів: потужний редактор для написання та виконання T-SQL-запитів.

3. Адміністрування: налаштування безпеки, керування користувачами та дозволами, створення резервних копій, моніторинг продуктивності.

4. Інтеграція: тісно інтегрується з екосистемою MS SQL Server, дозволяючи керувати різними службами, такими як Analysis Services, Reporting Services тощо.

При цьому SQL Server Management Studio– це самостійний продукт. Він не відповідає будь-якій конкретній версії MS SQL Server.

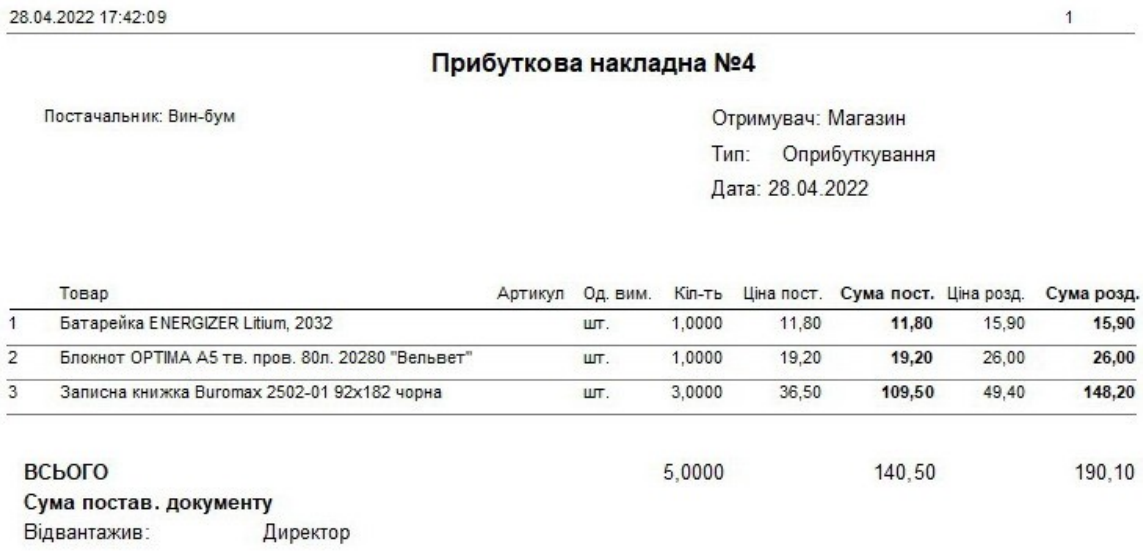
### 1.3 Питання для самоперевірки по темі 1

1. Що таке мова SQL?
2. Назвіть основні категорії команд мови SQL.
3. Які оператори входять до підмножини DDL мови SQL?
4. Які оператори входять до підмножини DML мови SQL?
5. Що таке метамова?
6. Що таке форма Бекуса-Науера?
7. Які символи вживані в нотації Бекуса-Науера?
8. Типи даних мови SQL.
9. Як визначаються дані символьного типу в мові SQL?
10. Як визначаються дані бітового типу в мові SQL?
11. Як визначаються точні числові дані в мові SQL?
12. Як визначаються числові дані в десятковому форматі в мові SQL?
13. Як визначаються закруглені числові дані в мові SQL?
14. Як визначаються дані типу "дата/час" в мові SQL?
15. Що таке домен?
16. Де можна подивитися рейтинг сучасних СУБД?
17. Назвіть основні складові програмного комплексу Microsoft SQL Server.
18. Назвіть основні об'єкти бази даних Microsoft SQL Server.
19. Перерахуйте основні типи даних СУБД Microsoft SQL Server і дайте стислу характеристику кожному типу.
20. Що таке користувацький тип даних?
21. Для чого призначена мова Transact-SQL?
22. Які відмінності мови Transact-SQL від мови SQL?
23. Для чого призначений застосунок SQL Server Management Studio?

# ТЕМА 2
# ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА СТРУКТУРИ БАЗИ ДАНИХ

## 2.1 Стислий опис предметної області

Деяке підприємство закуповує продукцію у різних постачальників. Закупка продукції здійснюється партіями і оформлюється у вигляді договорів на постачання. Кожен договір на постачання продукції має унікальний номер і може бути укладений тільки з одним постачальником. Підставою для постачання є будь-який документ (попереднє замовлення, рахунок-фактура тощо). У документах за кожним договором для кожного виду продукції зазначаються: найменування, розмір поставленої партії (тобто кількість закупленої продукції) та ціна (у гривнях). Закуплена продукція надходить на склад. При надходженні продукції на склад обов'язково виконується формування прибуткової накладної. Один з можливих способів оформлення такої накладної наведений на рисунку 2.1. Цей приклад було запозичено з веб-ресурсу https://minisoft.ua/ru/node/328.



28.04.2022 17:42:09 1

**Прибуткова накладна №4**

Постачальник: Вин-бум

Отримувач: Магазин
Тип: Оприбуткування
Дата: 28.04.2022

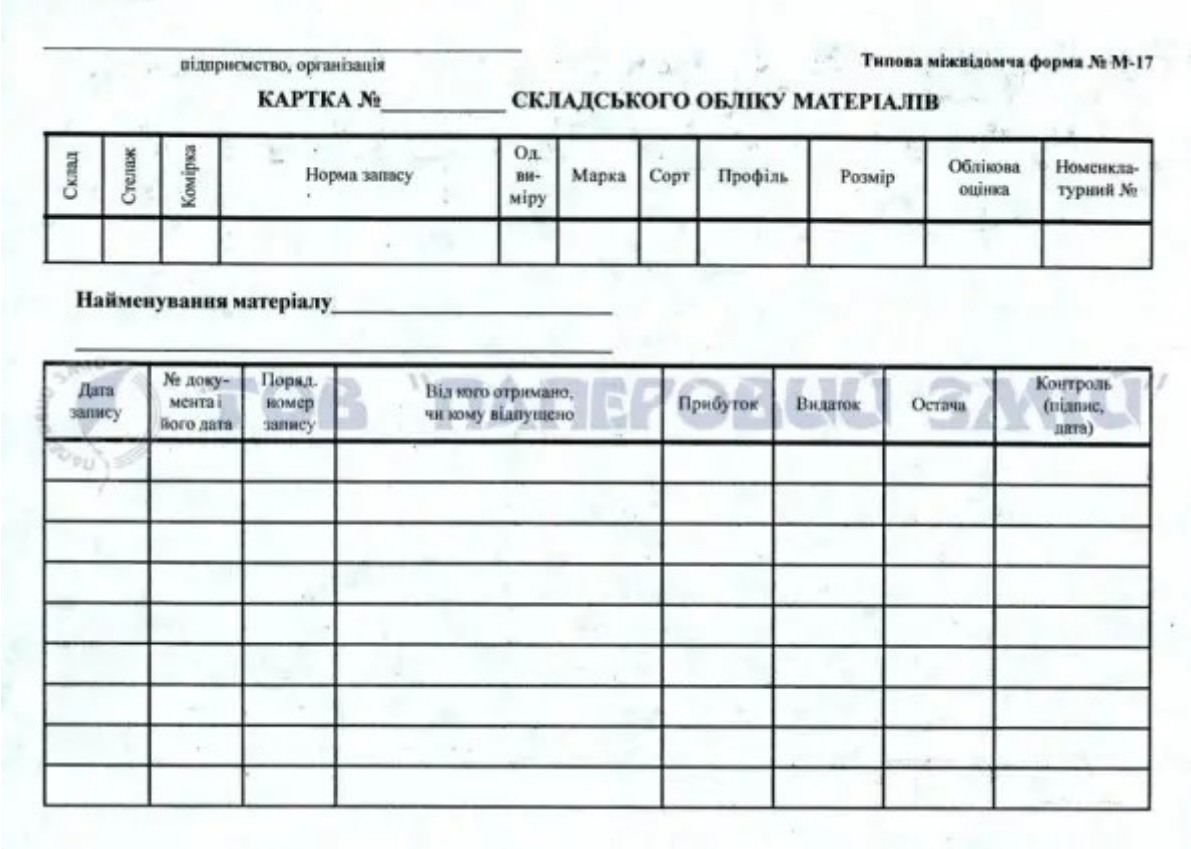
| | Товар | Артикул | Од. вим. | Кіл-ть | Ціна пост. | Сума пост. | Ціна розд. | Сума розд. |
|---|---|---|---|---|---|---|---|---|
| 1 | Батарейка ENERGIZER Litium, 2032 | | шт. | 1,0000 | 11,80 | **11,80** | 15,90 | **15,90** |
| 2 | Блокнот OPTIMA A5 тв. пров. 80л. 20280 "Вельвет" | | шт. | 1,0000 | 19,20 | **19,20** | 26,00 | **26,00** |
| 3 | Записна книжка Buromax 2502-01 92x182 чорна | | шт. | 3,0000 | 36,50 | **109,50** | 49,40 | **148,20** |
| ВСЬОГО | | | | 5,0000 | | 140,50 | | 190,10 |

**Сума постав. документу**

Відвантажив: Директор

Рисунок 2.1 – Приклад прибуткової накладної

Також при надходженні продукції на склад виконується заповнення карток складського обліку. Один з можливих способів оформлення такої картки наведений на рисунку 2.2. Цей приклад було запозичено з веб-ресурсу https://zmey.ua/ua/p525120511-kartochka-skladskogo-ucheta.html.



підприємство, організація

Типова міжвідомча форма № М-17

**КАРТКА №_____ СКЛАДСЬКОГО ОБЛІКУ МАТЕРІАЛІВ**

| Склад | Стелаж | Комірка | Норма запасу | Од. виміру | Марка | Сорт | Профіль | Розмір | Облікова оцінка | Номенклатурний № |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |

**Найменування матеріалу**_____

| Дата запису | № документа і його дата | Поряд. номер запису | Від кого отримано, чи кому відпущено | Прибуток | Видаток | Остача | Контроль (підпис, дата) |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Рисунок 2.2 – Картка складського обліку

Аналіз предметної області дозволив виділити і деталізувати основні бізнес-процеси, пов'язані з закупками продукції. Передбачається, що в процесі роботи підприємства доведеться зберігати досить великі обсяги даних, пов'язані з закупками продукції. Крім того, інформація про закупки продукції повинна бути організована таким чином, щоб персонал і керівництво підприємства мали можливість здійснювати її аналітичну обробку. У зв'язку з цим для зберігання і обробки інформації, пов'язаної з закупками продукції, необхідно створити базу даних. Аналіз бізнес-процесів дозволив виділити наступні інформаційні масиви, які можуть входити до складу такої бази даних.

1. Відомості про постачальників продукції.

Включають в себе інформацію про суб'єктів підприємницької діяльності, які працюють на ринку і пропонують продукцію, в придбанні якої зацікавлене підприємство, що розглядається. Постачальники як суб'єкти господарювання можуть бути юридичними особами або фізичними особи (тобто, фізична особа – підприємець або приватний підприємець).

До відомостей про постачальників відносяться такі дані, як:

– назва суб'єкта підприємницької діяльності, індивідуальний податковий номер, номер свідоцтва платника податку на додану вартість (ПДВ) (для юридичної особи);

– прізвище, ім'я, по батькові, номер свідоцтва про реєстрацію (для фізичної особи);

– адреса місцезнаходження, контактні дані (для фізичної та юридичної особи) тощо.

2. Відомості про постачання (тобто, закупки) продукції.

Кожна закупка здійснюється на підставі договору (або контракту) на поставку, який укладається між постачальником і підприємством.

Для кожної закупки відома наступна інформація:

– постачальник;

– дата поставки;

– загальна сума поставки;

– дані про поставлену продукцію.

Дані про закуплену продукцію включають в себе (по кожному виду продукції):

– найменування продукції;

– кількість одиниць;

– ціну за одиницю.

Ціна, за якою постачається продукція, може відрізнятися від стандартного прайс-листа постачальника (для конкретного покупця можуть діяти спеціальні знижки, ціна на окремі види продукції може призначатися індивідуально тощо).

### 2.2 Опис структури бази даних

Для зберігання і обробки подібної інформації засобами СУБД MS SQL Server може бути використана база даних (БД). Структуру цієї бази даних відображає модель даних у нотації IDEF1X. Цю модель даних наведено на рисунку 2.3. До складу бази даних, яка створюється за такою структурою, будуть входити таблиці, опис яких наведено далі.

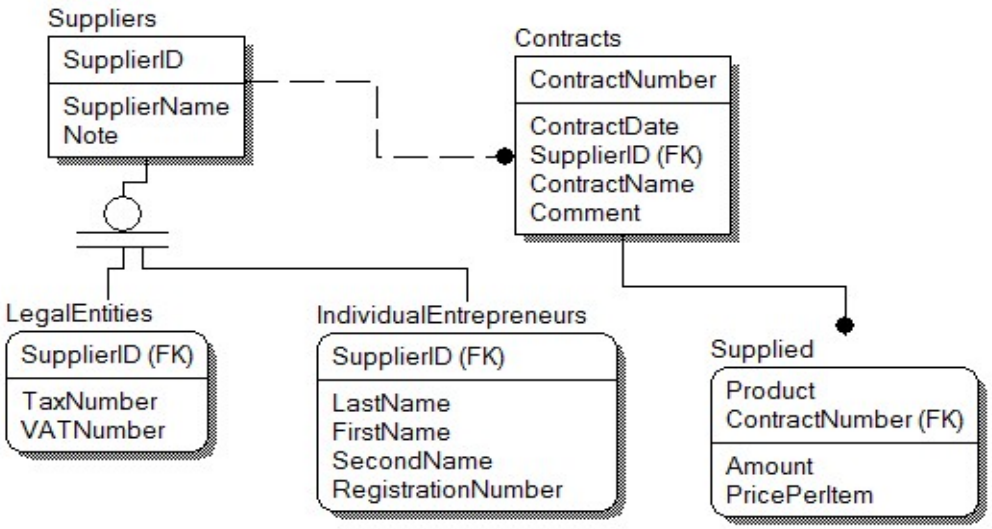



Рисунок 2.3 – Модель даних у нотації IDEF1X

1. Таблиця «Suppliers». Призначена для зберігання загальних даних про постачальників. Опис структури таблиці «Suppliers» наведений у таблиці 2.1.

Таблиця 2.1 – Опис структури таблиці «Suppliers»

| Field Name | Key | Data Type | Field Size | Description |
|---|---|---|---|---|
| SupplierName | | Text | | найменування постачальника |
| SupplierID | PK | Integer | | код постачальника |
| Note | | Text | | примітка |

2. Таблиця «LegalEntities». Призначена для зберігання загальних даних про постачальників, які є юридичними особами. Опис структури таблиці «LegalEntities» наведений у таблиці 2.2.

Таблиця 2.2 – Опис структури таблиці «LegalEntities»

| Field Name | Key | Data Type | Field Size | Description |
|---|---|---|---|---|
| SupplierID | PK, FK | Integer | | код постачальника |
| TaxNumber | | Char | 20 | податковий номер |
| VATNumber | | Char | 20 | номер свідоцтва платника ПДВ |

3. Таблиця «IndividualEntrepreneurs». Призначена для зберігання загальних даних про постачальників, які є фізичними особами (тобто приватними підприємцями). Опис структури таблиці «IndividualEntrepreneurs» наведений у таблиці 2.3.

Таблиця 2.3 – Опис структури таблиці «IndividualEntrepreneurs»

| Field Name | Key | Data Type | Field Size | Description |
|---|---|---|---|---|
| SupplierID | PK, FK | Number | Integer | код постачальника |
| LastName | | Char | 20 | прізвище |
| FirstName | | Char | 20 | ім'я |
| SecondName | | Char | 20 | по батькові |
| RegistrationNumber | | Text | 20 | номер свідоцтва про реєстрацію суб'єкта господарювання |

4. Таблиця «Contracts». Призначена для зберігання загальних даних про укладені договори. Опис структури таблиці «Contracts» наведений у таблиці 2.4.

Таблиця 2.4 – Опис структури таблиці «Contracts»

| Field Name | Key | Data Type | Field Size | Description |
|---|---|---|---|---|
| ContractNumber | PK | Integer | | номер договору |
| ContractDate | | Datetime | | дата підписання договору |
| SupplierID | FK | Integer | | код постачальника |
| ContractName | | Text | | найменування договору |
| Comment | | Text | | примітка |

5. Таблиця «Supplied». Призначена для зберігання загальних даних про закуплену продукцію. Опис структури таблиці «Supplied» наведений у таблиці 2.5.

Таблиця 2.5 – Опис структури таблиці «Supplied»

| Field Name | Key | Data Type | Field Size | Description |
|---|---|---|---|---|
| ContractNumber | PK, FK | Integer | | номер договору |
| Product | PK | Char | 20 | найменування товару |
| Amount | | Decimal | 4,0 | розмір партії (кількість штук) |
| PricePerItem | | Decimal | 8,2 | ціна за штуку (в гривнях) |

### 2.3 Питання для самоперевірки по темі 2

1. Які основні господарчі операції виконуються при закупках продукції?
2. Наявність яких документів є обов'язковою при закупках продукції?
3. Що таке витратна накладна? Хто, коли та за яких умов створює витратну накладну?
4. Що таке прибуткова накладна? Хто, коли та за яких умов створює прибуткову накладну?

5. Яким чином обліковується продукція на складі? Хто (тобто який працівник або працівники) відповідає за обліковування продукції на складі?

6. Які документи продавець надає покупцю при продажі продукції?

7. Що таке аномалії оновлення?

8. Які різновиди аномалій оновлення існують?

9. Які негативні наслідки для роботи з базою даних мають аномалії оновлення?

10. Що таке атомарність даних у базі даних?

11. Що таке нормальна форма? Назвіть основні нормальні форми.

12. Які вимоги до структури бази даних висувають нормальні форми?

13. Що таке відношення? Як це зв'язано із структурою реляційної бази даних?

14. Що таке нормалізація відношень?

15. Яким чином виконується нормалізація відношень при проектуванні реляційної бази даних?

16. Що таке декомпозиція відношень?

17. Що таке вимога «loss-less join»? Як ця вимога зв'язана з нормалізацією відношень?

18. Що таке вимога «dependency saving»? Як ця вимога зв'язана з нормалізацією відношень?

19. Поясніть, чому для зберігання інформації було обрано саме таку структуру бази даних. Які недоліки характерні для використовуваної структури бази даних?

20. Чи можна було б доповнити або розширити запропоновану структуру бази даних? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

21. Чи є порушення вимоги атомарності даних у запропонованій структурі бази даних? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

22. Чи є у запропонованій структурі бази даних аномалії оновлення даних? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

23. Чи відповідає запропонована структура бази даних вимогам нормальних форм? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

24. Яким чином контролюється неповторюваність даних у базі даних?

25. Неповторюваність яких даних контролюється у запропонованій базі даних? Яким чином?

26. Чи є у запропонованій базі даних дані, неповторюваність яких не контролюється, виходячи із наведеного опису, але її потрібно контролювати? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

27. Проаналізуйте структуру таблиці «Supplied». Поясніть, чи доцільно зберігати у такому вигляді дані про закуплену продукцію (при цьому як позитивну, так і негативну відповідь треба обґрунтувати).

28. Чи можна створити прибуткову накладну використовуючи дані, що будуть зберігатися у запропонованій базі даних? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

29. Чи можна створити дані для картки складського обліку використовуючи дані, що будуть зберігатися у запропонованій базі даних? Як позитивну, так і негативну відповідь треба обґрунтувати та навести відповідні приклади.

# ТЕМА 3
# СТВОРЕННЯ БАЗИ ДАНИХ ТА ВВЕДЕННЯ ДАНИХ У БАЗУ ДАНИХ

## 3.1 Теоретичні відомості

### 3.1.1 Оператори DDL мови SQL та Transact-SQL для роботи з базами даних та таблицями бази даних

Для роботи з базами даних застосовують такі оператори DDL SQL:

CREATE DATABASE – створює базу даних;

ALTER DATABASE – змінює вже існуючу базу даних;

DROP DATABASE – видаляє вже існуючу базу даних.

Стисло розглянемо структуру оператора CREATE DATABASE, яку наведено нижче.

```
CREATE DATABASE db_name
[ON [PRIMARY] {file_spec1}, ….]
[LOG ON {file_spec2}, ….]
[COLLATE collaction_name]
[FOR {ATTACH I ATTACH_REBUILD_LOG} ]
```

Розглянемо призначення основних аргументів у цій структурі.

db_name – задає ім'я бази даних, що створюється.

Опція ON задає всі файли бази даних явно.

file_spec – задає додаткові опції такі як логічне ім'я файла, фізичне ім'я та розмір.

PRIMARY – задає перший найважливіший файл, який містить системні таблиці та іншу внутрішню інформацію про БД. Якщо ця опція відсутня, то перший файл у списку специфікацій використовується як первинний.

LOG ON – визначає файл для розміщення протоколу трансакцій. Якщо опція LOG ON відсутня то протокол трансакцій створюється за замовчуванням.

COLLATE – задає порядок сортування БД.

FOR {ATTACH I ATTACH_REBUILD_LOG} – вказує, що БД створена шляхом приєднання набору файлів операційної системи.

Наведена вище структура оператору CREATE DATABASE є спрощеною. Загальна структура оператора CREATE DATABASE у мові Transact-SQL наведена на рисунку 3.1. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-database-transact-sql?view=sql-server-ver16&tabs=sqlpool

Для довідки. В базі даних MS SQL Server для зберігання використовуються три типи файлів. Це первинні і вторинні файли та журнали транзакцій. База даних повинна обов'язково містити первинний файл і принаймні один журнал транзакцій. При необхідності можна створити один чи декілька вторинних файлів даних і додаткові файли журналів транзакцій.

Первинні файли містять інформацію запуску бази даних. Також в первинних файлах зберігаються дані. В кожній базі даних є один первинний файл даних.

Вторинні файли містять всі дані, які не помістилися в первинному файлі. Якщо первинний файл має великий розмір достатній для зберігання всієї бази даних, то в такій базі даних вторинні файли не використовуються.

Журнали транзакцій містять дані, які необхідні для відновлення бази даних. В кожній базі даних повинен бути як мінімум один журнал трансакцій. Мінімальний розмір файлу журналу дорівнює 512 КБ.

В разі потреби у зміні властивостей вже існуючої бази даних може бути застосований оператор ALTER DATABASE.

```
CREATE DATABASE database_name
[ CONTAINMENT = { NONE | PARTIAL } ]
[ ON
    [ PRIMARY ] <filespec> [ ,...n ]
    [ , <filegroup> [ ,...n ] ]
    [ LOG ON <filespec> [ ,...n ] ]
]
[ COLLATE collation_name ]
[ WITH <option> [,...n ] ]
[;]

<option> ::=
{
    FILESTREAM ( <filestream_option> [,...n ] )
  | DEFAULT_FULLTEXT_LANGUAGE = { lcid | language_name | language_alias }
  | DEFAULT_LANGUAGE = { lcid | language_name | language_alias }
  | NESTED_TRIGGERS = { OFF | ON }
  | TRANSFORM_NOISE_WORDS = { OFF | ON}
  | TWO_DIGIT_YEAR_CUTOFF = <two_digit_year_cutoff>
  | DB_CHAINING { OFF | ON }
  | TRUSTWORTHY { OFF | ON }
  | PERSISTENT_LOG_BUFFER=ON ( DIRECTORY_NAME='<Filepath to folder on DAX
formatted volume>' )
  | LEDGER = {ON | OFF}
}
<filestream_option> ::=
{
    NON_TRANSACTED_ACCESS = { OFF | READ_ONLY | FULL }
  | DIRECTORY_NAME = 'directory_name'
}
<filespec> ::=
{
(
  NAME = logical_file_name ,
  FILENAME = { 'os_file_name' | 'filestream_path' }
  [ , SIZE = size [ KB | MB | GB | TB ] ]
  [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
  [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
)
}
<filegroup> ::=
{
FILEGROUP filegroup name [ [ CONTAINS FILESTREAM ] [ DEFAULT ] | CONTAINS
MEMORY_OPTIMIZED_DATA ]
  <filespec> [ ,...n ]
}
```

Рисунок 3.1 – Структура оператора CREATE DATABASE у мові Transact-SQL

Загальна структура оператора ALTER DATABASE у мові Transact-SQL наведена на рисунку 3.2. За допомогою цього оператора можна змінити основні характеристики бази даних, включаючи ім'я бази даних. Докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql?view=sql-server-ver16&tabs=sqlpool

```
ALTER DATABASE { database_name | CURRENT }
{
   MODIFY NAME = new_database_name
  | COLLATE collation_name   | <file_and_filegroup_options>
  | SET <option_spec> [ ,...n ] [ WITH <termination> ]
}
[;]
<file_and_filegroup_options>::=
  <add_or_modify_files>::=
  <filespec>::=
  <add_or_modify_filegroups>::=
  <filegroup_updatability_option>::=
<option_spec>::=
{
  | <auto_option>   | <change_tracking_option>   | <cursor_option>
  | <database_mirroring_option>   | <date_correlation_optimization_option>
  | <db_encryption_option>   | <db_state_option>   | <db_update_option>
  | <db_user_access_option><delayed_durability_option>   | <external_access_option>
  | <FILESTREAM_options>   | <HADR_options>   | <parameterization_option>
  | <query_store_options>   | <recovery_option>   | <service_broker_option>
  | <snapshot_option>   | <sql_option>   | <termination>   | <temporal_history_retention>
  | <data_retention_policy>   | <compatibility_level>
     { 150 | 140 | 130 | 120 | 110 | 100 | 90 }
}
```

Рисунок 3.2 – Структура оператора ALTER DATABASE у мові Transact-SQL

В разі потреби у видаленні вже існуючої бази даних може бути застосований оператор DROP DATABASE.

Загальна структура оператора DROP DATABASE у мові Transact-SQL наведена на рисунку 3.3. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/statements/drop-database-transact-sql?view=sql-server-ver16

```
DROP DATABASE [ IF EXISTS ] { database_name | database_snapshot_name } [ ,...n ] [;]
```

Рисунок 3.3 – Структура оператора DROP DATABASE у мові Transact-SQL

Для роботи з таблицями бази даних застосовують такі оператори DDL SQL:

CREATE TABLE – створює нову таблицю у базі даних;

ALTER TABLE – змінює структуру вже існуючої таблиці у базі даних;

DROP TABLE – видаляє існуючу існуючої таблицю з бази даних.

Оператор створення таблиці у загальному випадку має формат:

CREATE TABLE <ім'я таблиці>

(<ім'я стовпця><тип даних>[NOT NULL]

[,<ім'я стовпця><тип даних>[NOT NULL]]…);

Обов'язковими операндами оператора є ім'я створюваної таблиці та ім'я хоча б одного стовпця (поля) з вказанням типу даних, що зберігатимуться у цьому стовпці. При створенні таблиці для окремих полів можуть вказуватись деякі додаткові правила контролю введення в них даних, наприклад конструкція NOT NULL (не пусте) означає, що в цьому стовпці повинно бути визначене значення.

Наведена вище структура оператору CREATE TABLE є дуже спрощеною. Загальна структура оператора CREATE TABLE у мові

Transact-SQL наведена на рисунку 3.4. Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16

```
CREATE TABLE [ database_name.[schema_name] . | schema_name . ] table_name     ( { <column_definition> | <computed_column_definition> }
                [ <table_constraint> ] [ ,...n ] )
                [ ON { partition_scheme_name ( partition_column_name ) | filegroup | "default" } ]
                [ { TEXTIMAGE_ON { filegroup | "default" } ]
[ ; ]
<column_definition> ::=
column_name <data_type>
   [ COLLATE collation_name ]
   [ NULL | NOT NULL ]
   [
      [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ]
   ]
   [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]
<data type> ::=
[ type_schema_name . ] type_name
   [ ( precision [ , scale ] | max |
      [ { CONTENT | DOCUMENT } ] xml_schema_collection ) ]
<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [
         WITH FILLFACTOR = fillfactor
       | WITH ( < index_option > [ , ...n ] )
]
      [ ON { partition_scheme_name ( partition_column_name )
         | filegroup | "default" } ]
  | [ FOREIGN KEY ]
      REFERENCES [ schema_name . ] referenced_table_name [ ( ref_column ) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
  | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
< table_constraint > ::=
[ CONSTRAINT constraint_name ]
{    { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ] (column [ ASC | DESC ] [ ,...n ] )
      [
         WITH FILLFACTOR = fillfactor
        |WITH ( <index_option> [ , ...n ] )
      ]
   [ ON { partition_scheme_name (partition_column_name) | filegroup | "default" } ]
   | FOREIGN KEY ( column [ ,...n ] )
      REFERENCES referenced_table_name [ ( ref_column [ ,...n ] ) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
```

Рисунок 3.4 – Структура оператора CREATE TABLE у мові Transact-SQL

Формат оператора зміни структури таблиці у загальному випадку має вигляд:

ALTER TABLE < ім'я таблиці >

({ADD,MODIFY,DROP} <і'мя стовпця> [<тип даних>][NOT NULL]

[,{ADD,MODIFY,DROP} < і'мя стовпця > [<тип даних >][NOT NULL]]…);

Зміна структури таблиці може полягати в додаванні (ADD), зміні (MODIFY) або видаленні (DROP) одного чи декількох стовпців таблиці. Правила запису оператора ALTER TABLE такі ж, як і оператора CREATE TABLE. При видаленні стовпця тип даних вказувати не потрібно.

Наведена вище структура оператору ALTER TABLE є дуже спрощеною. Загальна структура оператора ALTER TABLE у мові Transact-SQL наведена на рисунку 3.5.

Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі:

https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver16

Оператор видалення таблиці у загальному випадку має вигляд:

DROP TABLE <ім'я таблиці>;

Оператор дозволяє видалити наявну таблицю.

Загальна структура оператора DROP TABLE у мові Transact-SQL наведена на рисунку 3.6. Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі:

https://learn.microsoft.com/en-us/sql/t-sql/statements/drop-table-transact-sql?view=sql-server-ver16

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
{ ALTER COLUMN column_name
    {
      [ type_schema_name. ] type_name [ ( { precision [ , scale ]
          | max | xml_schema_collection } ) ]
      [ NULL | NOT NULL ]
      [ COLLATE collation_name ]
    | {ADD | DROP } { ROWGUIDCOL | PERSISTED }
    }
    | [ WITH { CHECK | NOCHECK } ] ADD
    {
      <column_definition>
     | <computed_column_definition>
     | <table_constraint>
    } [ ,...n ]
    | DROP
    {
      [ CONSTRAINT ] constraint_name
      [ WITH ( <drop_clustered_constraint_option> [ ,...n ] ) ]
      | COLUMN column_name
    } [ ,...n ]
    | [ WITH { CHECK | NOCHECK } ] { CHECK | NOCHECK } CONSTRAINT
      { ALL | constraint_name [ ,...n ] }
    | { ENABLE | DISABLE } TRIGGER
      { ALL | trigger_name [ ,...n ] }
    | SWITCH [ PARTITION source_partition_number_expression ]
      TO [ schema_name. ] target_table
      [ PARTITION target_partition_number_expression ]
} [ ; ]
```

Рисунок 3.5 – Структура оператора ALTER TABLE у мові Transact-SQL

```
DROP TABLE [ IF EXISTS ] { database_name.schema_name.table_name |
schema_name.table_name | table_name } [ ,...n ]
[ ; ]
```

Рисунок 3.6 – Структура оператора DROP TABLE у мові Transact-SQL

### 3.1.2 Оператори DML мови SQL та Transact-SQL

Оператори DML мови SQL орієнтовані на виконання операцій над групами записів, хоча в деяких випадках їх можна проводити і над окремим записом.

Запити DML є досить потужним засобом, оскільки дозволяють оперувати не лише окремими рядками, але і набором рядків. За допомогою запитів DML користувач може додати, видалити або оновити блоки даних. Існує три види запитів DML:

• INSERT INTO – запит додавання;

• DELETE – запит видалення;

• UPDATE – запит оновлення.

Оператор INSERT застосовується для додавання записів в таблицю. Формат оператора у загальному випадку має вигляд:

<оператор_вставки>::=INSERT INTO <ім'я_таблиці>
[(ім'я_стовпця [,..n])] {VALUES (значення[,..n])|
<SELECT_оператор>}

Тут параметром ім'я_таблиці є або ім'я таблиці бази даних, або ім'я оновлюваного представлення.

Перша форма оператора INSERT з параметром VALUES призначена для вставки єдиного рядка у вказану таблицю. Список стовпців вказує стовпці, яким будуть присвоєні значення в записах, що додаються. Список може бути опущений, тоді маються на увазі усі стовпці таблиці (окрім оголошених як лічильник), причому в певному порядку, встановленому при створенні таблиці. Якщо в операторі INSERT вказується конкретний список імен полів, то будь-які пропущені в нім стовпці мають бути оголошені при створенні таблиці як що допускають значення NULL, за винятком тих випадків, коли при описі стовпця використовувався параметр DEFAULT. Список значень повинен таким чином відповідати списку стовпців:

– кількість елементів в обох списках має бути однаковою;

– повинна існувати пряма відповідність між позицією одного і того ж елементу в обох списках, тому перший елемент списку значень повинен відноситися до першого стовпця в списку стовпців, другий - до другого стовпця і так далі

– типи цих елементів в списку значень мають бути сумісні з типами цих відповідних стовпців таблиці.

Друга форма оператора INSERT з параметром SELECT дозволяє скопіювати множину рядків з однієї таблиці в іншу. Такі оператори також називають багаторядковими. Пропозицією SELECT багаторядкового оператора INSERT може бути будь-який допустимий оператор SELECT-SQL. Рядки, що вставляються у вказану таблицю, в точності повинні відповідати рядкам результуючої таблиці, створеної при виконанні вкладеного запиту. Усі обмеження, вказані вище для першої форми оператора SELECT, застосовні і в цьому випадку.

Оскільки оператор SELECT в загальному випадку повертає множину записів, то оператор INSERT в такій формі призводить до додавання в таблицю аналогічного числа нових записів.

В запис, що міститься в середині багаторядкового оператора INSERT, стандарт SQL1 вимагає деяких логічних обмежень:

– в запит не можна включати ORDER BY;

– таблиця результату запиту повинна містити таку ж кількість колонок, що і оператор INSERT;

– запит не може бути запитом на об'єднання кількох різних операторів SELECT;

– ім'я цільової таблиці оператора INSERT не може бути присутнім в виразі FROM запиту на читання чи любого запиту, вкладеного в нього. Таким чином забороняється додавання таблиці саму в себе.

В стандарті SQL2 останні два обмеження були послаблені і в запиті дозволяється об'єднання операторів, об'єднання таблиць і виразів, дозволяється «самододавання».

```
 [ WITH <common_table_expression> [ ,...n ] ]
INSERT
{
      [ TOP ( expression ) [ PERCENT ] ]
      [ INTO ]
      { <object> | rowset_function_limited
        [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
      }
   {
      [ ( column_list ) ]
      [ <OUTPUT Clause> ]
      { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] ) [ ,...n     ]
      | derived_table
      | execute_statement
      | <dml_table_source>
      | DEFAULT VALUES
      }
   }
}
[;]

<object> ::=
{
   [ server_name . database_name . schema_name .
     | database_name .[ schema_name ] .
     | schema_name .
   ]
  table_or_view_name
}

<dml_table_source> ::=
   SELECT <select_list>
   FROM ( <dml_statement_with_output_clause> )
     [AS] table_alias [ ( column_alias [ ,...n ] ) ]
   [ WHERE <search_condition> ]
      [ OPTION ( <query_hint> [ ,...n ] ) ]
```

Рисунок 3.7 – Структура оператора INSERT у мові Transact-SQL

Наведена вище структура оператору INSERT є дуже спрощеною. Загальна структура оператора INSERT у мові Transact-SQL наведена на рисунку 3.7. Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі:
https://learn.microsoft.com/en-us/sql/t-sql/statements/insert-transact-sql?view=sql-server-ver16

Оператор UPDATE застосовується для зміни значень в групі записів або в одному записі вказаної таблиці.

Формат оператора у загальному випадку має вигляд:

<оператор_зміни> ::=
UPDATE ім'я_таблиці SET ім'я_стовпця = <вираження>[,..n]
[WHERE <умова_відбору>]

Параметр ім'я_таблиці – це або ім'я таблиці бази даних, або ім'я оновлюваного представлення. У пропозиції SET вказуються імена одного і більше за стовпці, дані в яких необхідно змінити. Пропозиція WHERE є необов'язковою. Якщо вона відсутня, значення вказаних стовпців будуть змінені в усіх рядках таблиці. Якщо пропозиція WHERE є присутньою, то оновлені будуть тільки ті рядки, які задовольняють умові відбору. Вираження є новим значенням відповідного стовпця і має бути сумісне з ним за типом даних.

Наведена вище структура оператору UPDATE є дуже спрощеною. Загальна структура оператора UPDATE у мові Transact-SQL наведена на рисунку 3.8. Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним.

```
[ WITH <common_table_expression> [...n] ]
UPDATE
   [ TOP ( expression ) [ PERCENT ] ]
   { { table_alias | <object> | rowset_function_limited
       [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
     }
     | @table_variable
   }
   SET
      { column_name = { expression | DEFAULT | NULL }
        | { udt_column_name.{ { property_name = expression
                              | field_name = expression }  | method_name ( argument [ ,...n ] )
                            }
        }
        | column_name { .WRITE ( expression , @Offset , @Length ) }
        | @variable = expression
        | @variable = column = expression
        | column_name { += | -= | *= | /= | %= | &= | ^= | |= } expression
        | @variable { += | -= | *= | /= | %= | &= | ^= | |= } expression
        | @variable = column { += | -= | *= | /= | %= | &= | ^= | |= } expression
      } [ ,...n ]
   [ <OUTPUT Clause> ]
   [ FROM{ <table_source> } [ ,...n ] ]
   [ WHERE { <search_condition>
         | { [ CURRENT OF
               { { [ GLOBAL ] cursor_name }
                  | cursor_variable_name
               }
             ]
          }
         }
   ]
   [ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]
<object> ::=
{
   [ server_name . database_name . schema_name .  | database_name .[ schema_name ] .
   | schema_name .     ]
   table_or_view_name}
```

Рисунок 3.8 – Структура оператора UPDATE у мові Transact-SQL

Опис структури, що наведено на рисунку 3.8 та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/queries/update-transact-sql?view=sql-server-ver16

Оператор DELETE призначений для видалення групи записів з таблиці.

Формат оператора у загальному випадку має вигляд:

<оператор_видалення> ::=DELETE
FROM <ім'я_таблиці>[WHERE <умова_відбору>]

Тут параметром ім'я_таблиці є або ім'я таблиці бази даних, або ім'я оновлюваного представлення.

Якщо пропозиція WHERE є присутньою, видаляються записи з таблиці, що задовольняють умові відбору. Якщо опустити пропозицію WHERE, з таблиці будуть видалені усі записи, проте сама таблиця збережеться.

Наведена вище структура оператору DELETE є дуже спрощеною. Загальна структура оператора DELETE у мові Transact-SQL наведена на рисунку 3.9. Далі призначення та використання цього оператору та його складових буде розглядатися на практичних прикладах, тому докладно розглядати тут аргументи оператора не вважається доцільним. Опис цієї структури та призначення аргументів у разі потреби можна переглянути на веб-ресурсі: https://learn.microsoft.com/en-us/sql/t-sql/statements/delete-transact-sql?view=sql-server-ver16

```
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
   [ TOP ( expression ) [ PERCENT ] ]
   [ FROM ]
   { { table_alias
     | <object>
     | rowset_function_limited
     [ WITH ( table_hint_limited [ ...n ] ) ] }
     | @table_variable
   }
   [ <OUTPUT Clause> ]
   [ FROM table_source [ ,...n ] ]
   [ WHERE { <search_condition>
          | { [ CURRENT OF
                { { [ GLOBAL ] cursor_name }
                   | cursor_variable_name
                }
            ]
           }
          }
   ]
   [ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]

<object> ::=
{
   [ server_name.database_name.schema_name.
     | database_name. [ schema_name ] .
     | schema_name.
   ]
   table_or_view_name
}
```

Рисунок 3.9 – Структура оператора DELETE у мові Transact-SQL

## 3.2 Практичне опанування теми 3

### 3.2.1 Передумови виконання завдань теми 3

Увага! Виконання практичних завдань, які наведено далі, передбачає використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи MS SQL Server. При виконанні цієї та наступних практичних тем передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

### 3.2.2 Створення бази даних та введення даних у базу даних в інтерактивному режимі

При створенні бази даних та введенні даних у базу даних в інтерактивному режимі користувач виконує таку послідовність дій.

1. У разі потреби зберігання файлів, які з'являться при виконанні лабораторної роботи, окремо, створити на певному логічному диску комп'ютера каталог із довільним ім'ям (наприклад, E:\LABMSSQL).

2. Виконати запуск застосунку SQL Server Management Studio.

3. У вікні підключення, яке наведено на рисунку 3.10, натиснути кнопку Connect. Слід звернути увагу, що це спрощений варіант підключення, тому що для виконання процедур ідентифікації та аутентифікації при підключенні до Microsoft SQL Server буде використано облікові дані користувача операційної системи Windows.

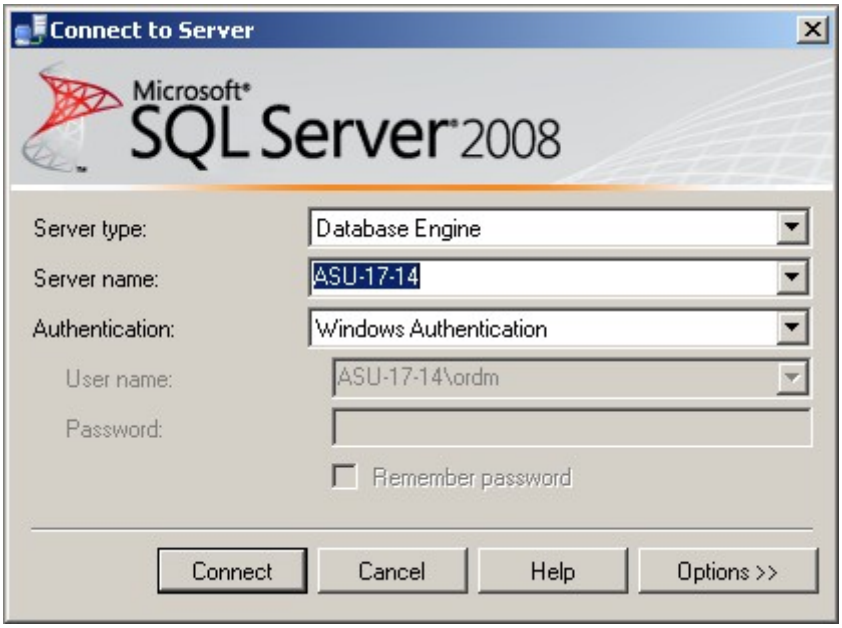



Рисунок 3.10

4. Після появи на екрані середовища SQL Server Management Studio у вікні Object Explorer вибрати пункт Databases (рисунок 3.11), натиснути праву кнопку миші і в меню вибрати пункт New Database… . В результаті на екрані з'явиться вікно, яке дозволить ввести основні параметри нової бази даних. Необхідно ввести ім'я нової бази даних – delivery та визначити місце розміщення файлів - E:\LABMSSQL (рисунок 3.12). Після введення даних натиснути кнопку ОК. Нова база даних з'явиться у списку баз даних у вікні Object Explorer (рисунок 3.11).

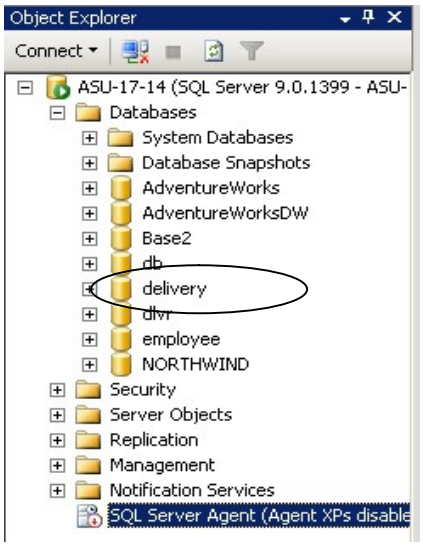



Рисунок 3.11

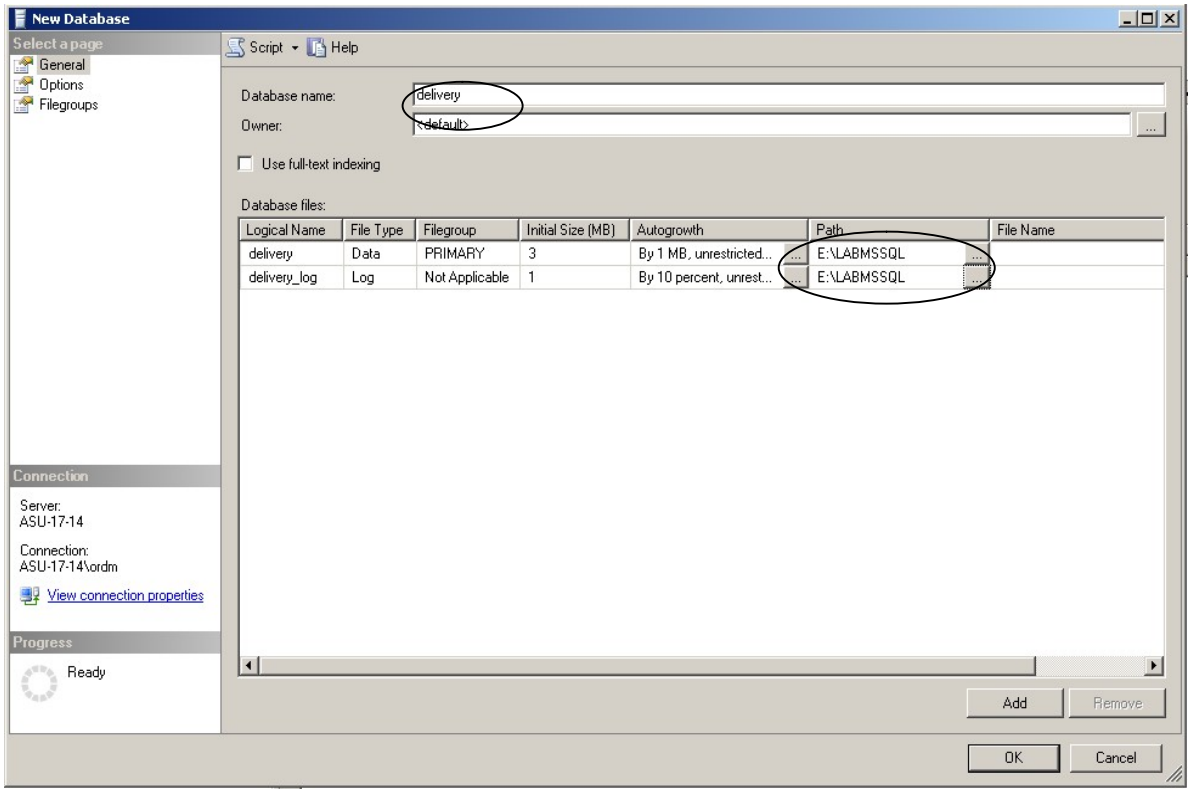



Рисунок 3.12

4. Вибрати створену базу даних та розкрити список її об'єктів (рисунок 3.13).

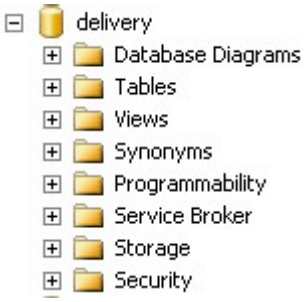



Рисунок 3.13

5. У списку об'єктів бази даних клацнути правою кнопкою миші по пункту Tables і в меню вибрати пункт New Table… . Ввести поля нової таблиці (рисунок 3.14), визначивши при цьому типи даних і ключове поле (для цього потрібно клацнути правою кнопкою миші по полю і вибрати в меню відповідний пункт (рисунок 3.15)).

| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | SupplierID | int | ☐ |
| | SupplierName | text | ☐ |
| | Note | text | ☑ |
| | | | ☐ |

Рисунок 3.14

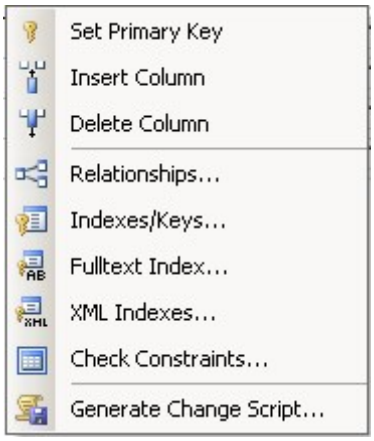



Рисунок 3.15

6. Закрити вкладку із структурою нової таблиці. Зберегти нову таблицю під назвою «Suppliers» (без лапок).

7. Аналогічно створити таблиці «IndividualEntrepreneurs» та «LegalEntities». Їхні структури наведені на рисунках 3.16 та 3.17 відповідно.

| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | SupplierID | int | ☐ |
| | LastName | char(20) | ☐ |
| | FirstName | char(20) | ☐ |
| | SecondName | char(20) | ☐ |
| | RegistrationNumber | char(10) | ☑ |
| | | | ☐ |

Рисунок 3.16

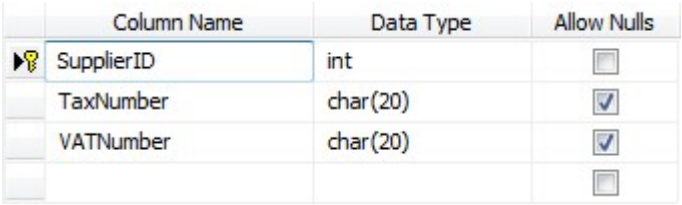



Рисунок 3.17

8. Створити таблицю «Contracts». Структура таблиці наведено на рисунку 3.18. Особливістю цієї таблиці є те, що для поля «ContractNumber» має бути встановлена властивість автоприрощення (autoincrement) з початковим значенням 1 і кроком зміни 1. Для цього Microsoft SQL Server використовується властивість Identity. Необхідно змінити значення властивості, як показано рисунку 3.19.

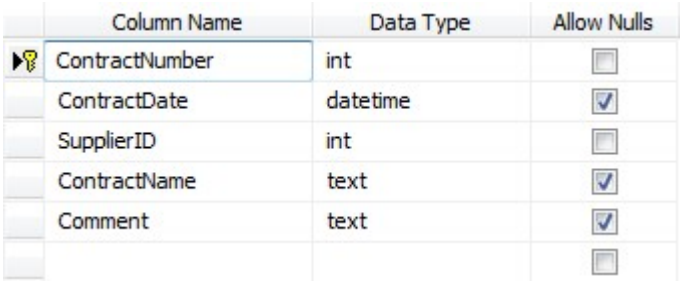



Рисунок 3.18


Рисунок 3.19

9. Створити таблицю «Supplied». Структура таблиці наведена на рисунку 3.20. Особливістю таблиці є складовий первинний ключ. Для його створення потрібно виділити ключові поля (це можна зробити мишею, при натиснутій клавіші Shift) і потім визначити їх як ключові поля.

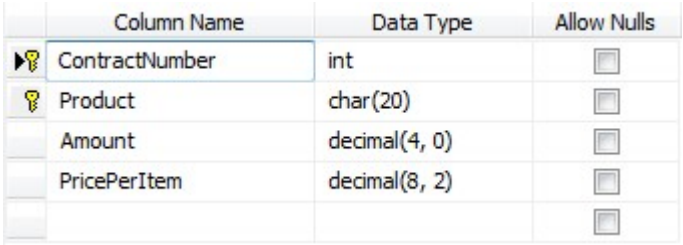



Рисунок 3.20

10. У результаті створення таблиць структура створеної бази даних матиме вигляд, наведений на рисунку 3.21. У тому випадку, якщо список таблиць не відображається, можна клацнути правою кнопкою миші на ім'я бази даних і в меню вибрати пункт Refresh.

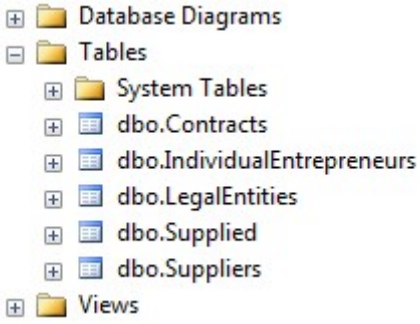



Рисунок 3.21

11. Тепер між створеними таблицями необхідно встановити зв'язки. Це, зокрема, можна зробити за допомогою візуальних засобів. Для цього потрібно створити діаграму бази даних. Для створення діаграми потрібно клацнути правою кнопкою миші по пункту Database Diagrams (рисунок 3.21) і в меню вибрати пункт New Database Diagram. Потім потрібно послідовно додати до складу діаграми таблиці, вибираючи їх зі списку та натискаючи кнопку Add (рисунок 3.22).

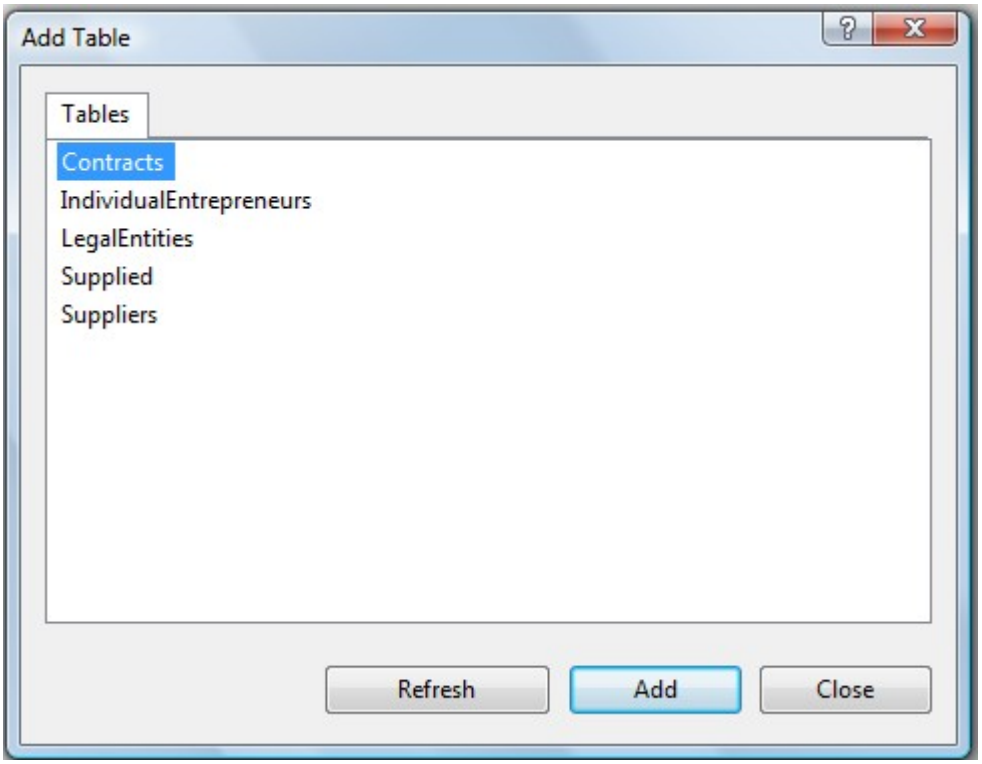



Рисунок 3.22

12. Після включення таблиць до складу діаграми необхідно зв'язати їх ключові поля. Для цього потрібно вибрати за допомогою миші ключове поле батьківської таблиці і, не відпускаючи кнопку миші, тягнути покажчик миші до дочірньої таблиці. В результаті встановлення зв'язку на екран буде виведено вікно, що відображає ім'я зв'язку та поля, що зв'язуються (рисунок 3.23). Цей приклад відображає встановлення зв'язку між таблицями «Suppliers» та «LegalEntities». Підтвердивши параметри зв'язку, користувач може підтвердити або змінити параметри зовнішнього ключа і тип відносин цілісності посилання (рисунок 3.24).

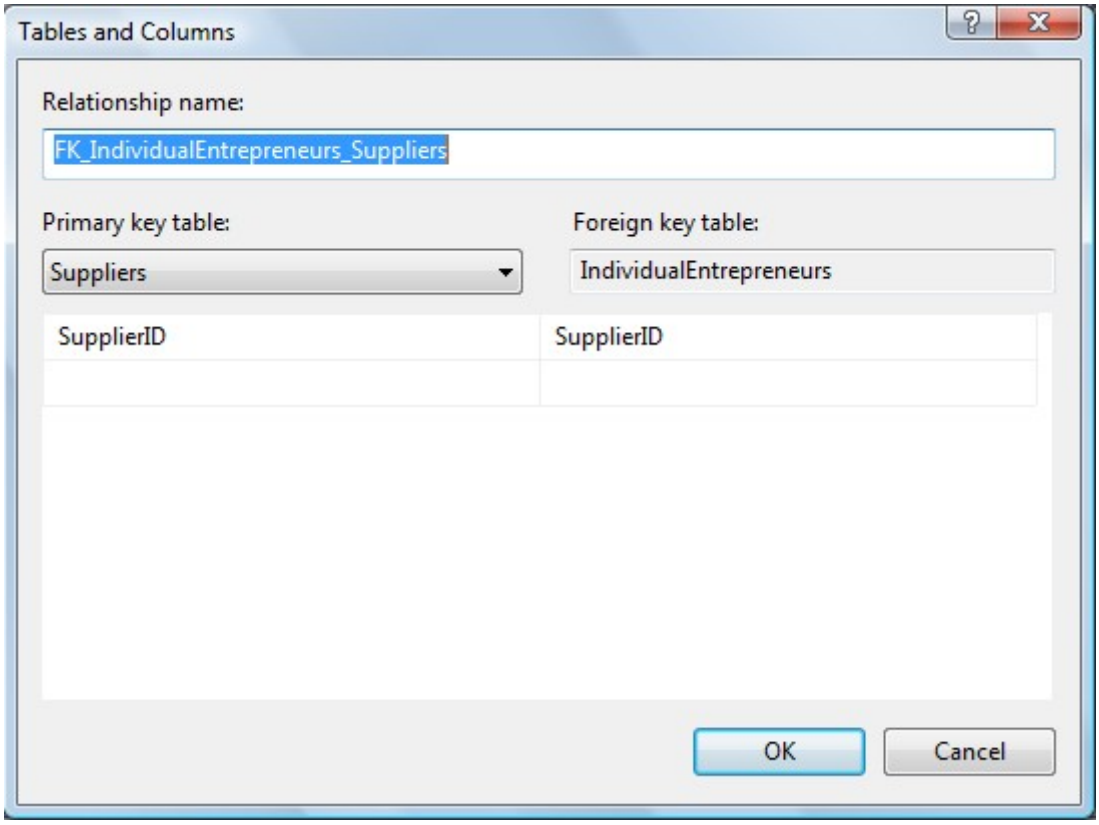



Рисунок 3.14

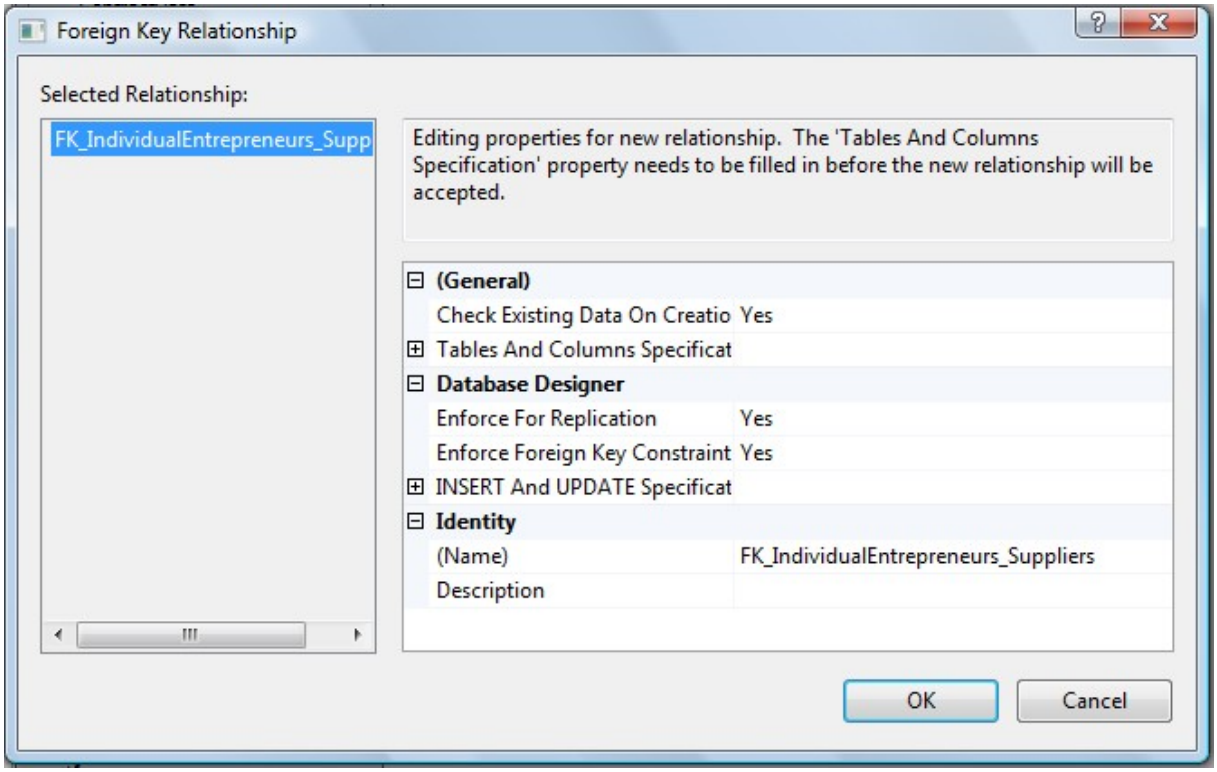



Рисунок 3.24

13. Внаслідок встановлення зв'язків між таблицями діаграма може мати вигляд (рисунок 3.25). Сформовану діаграму можна закрити і зберегти при цьому довільним ім'ям, наприклад Diagram_0. Ця діаграма з'явиться у загальному списку діаграм бази даних.

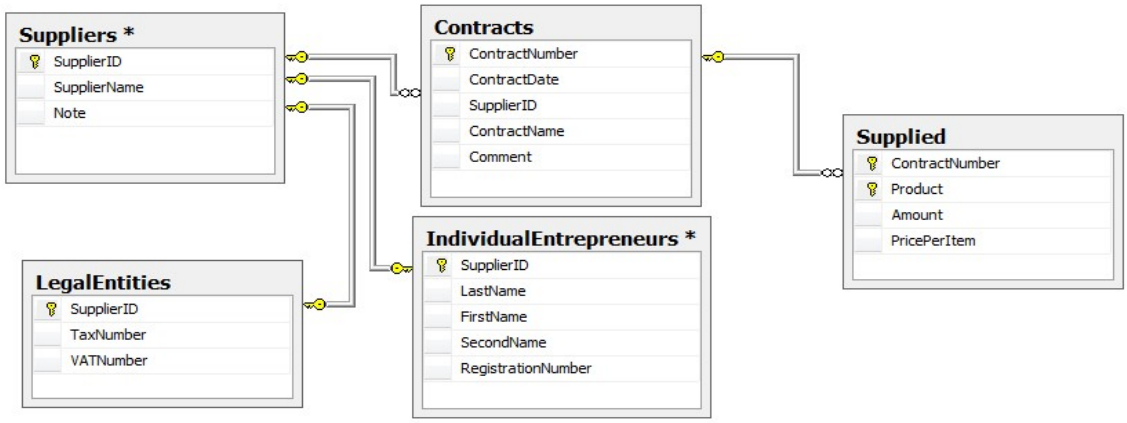



Рисунок 3.25

14. За допомогою діаграми баз даних можна змінювати структуру таблиць, встановлювати зв'язки, додаткові властивості полів тощо. Припустимо, що у полях «Amount» і «PricePerItem» таблиці «Supplied» необхідно реалізувати вимоги, які полягають у тому, що дані, які зберігаються у цих полях, мають бути позитивними. Для цього треба знову відкрити діаграму, клацнути правою кнопкою миші по таблиці «Supplied» і в меню вибрати пункт Check Constraints... . У вікні потрібно натиснути кнопку Add і ввести вираз для контролю і назву (рисунок 3.26).

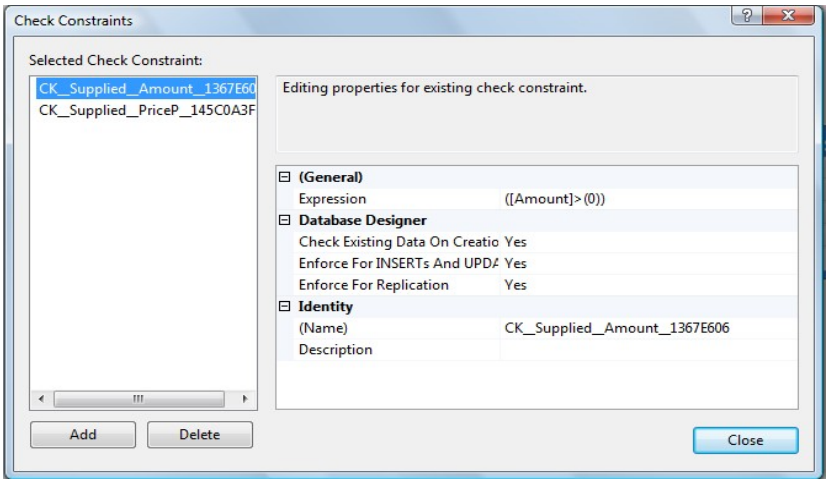



Рисунок 3.26

15. Аналогічно можна сформувати контрольний вираз для поля «PricePerItem». У цьому випадку вираз (Expression) матиме вигляд: ([PricePerItem]>0), а ім'я (Name): CK_Supplied_PricePerItem. Після внесення цих змін діаграму можна закрити та зберегти.

16. Після закриття діаграми необхідно проаналізувати структурні зміни, зроблені у таблицях (поява нових ключів тощо). Для цього слід проаналізувати об'єкти кожної таблиці, послідовно відкриваючи таблиці у списку таблиць.

17. Тепер треба ввести дані у таблиці. Для введення інформації в таблицю потрібно вибрати таблицю в списку таблиць, клацнувши по ній правою кнопкою миші, і в меню вибрати пункт Open Table (або Change first 200 records). В результаті таблицю буде виведено на екран у вигляді, що дозволяє вводити нові дані або коригувати введені раніше. Використовуючи інтерактивні засоби SQL Server Management Studio, необхідно ввести в таблиці бази даних інформацію, наведену на рисунках 3.27 – 3.31.

| | SupplierID | SupplierName | Note |
|---|---|---|---|
| ▶ | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12-34-56, факс 22-12-33) |
| | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 |
| | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 |
| | 4 | ЗАТ «Транссервіс» | м. Одеса, вул. Дерибасівська, 75 |
| | 5 | ПП Сидорчук М.С. | м. Полтава, вул. Свободи, 15, кв. 43 |
| * | *NULL* | *NULL* | *NULL* |

Рисунок 3.27

| | SupplierID | LastName | FirstName | SecondName | ReqistrationNumber |
|---|---|---|---|---|---|
| ▶ | 1 | Іваненко | Ілля | Іванович | 00143987 |
| | 3 | Петренко | Павло | Петрович | 12345678 |
| | 5 | Сидорчук | Микита | Степанович | 09876541 |
| * | *NULL* | *NULL* | *NULL* | *NULL* | *NULL* |

Рисунок 3.28

| | SupplierID | TaxNumber | VATNumber |
|---|---|---|---|
| ▶ | 2 | 00123987 | 19848521 |
| | 4 | 29345678 | 25912578 |
| * | *NULL* | *NULL* | *NULL* |

Рисунок 3.29

| | ContractNumber | ContractDate | SupplierID | ContractName | Comment |
|---|---|---|---|---|---|
| ▶ | 1 | 1999-09-01 00:00:00.000 | 1 | Договір № 1 | Підстава - накладна №34 від 30/08/99 |
| | 2 | 1999-09-10 00:00:00.000 | 1 | Договір № 2 | Підстава - рахунок-фактура № 08-78 від 28/08/99 |
| | 3 | 1999-09-10 00:00:00.000 | 3 | Договір № 3 | Підстава - рахунок-фактура № 08-78 від 28/08/99 |
| | 4 | 1999-09-23 00:00:00.000 | 3 | Договір № 4 | Підстава - замовлення № 56 від 28/08/99 |
| | 5 | 1999-09-24 00:00:00.000 | 2 | Договір № 5 | Підстава - накладна № 74 від 11/09/99 |
| | 6 | 1999-10-01 00:00:00.000 | 1 | Договір № 6 | Підстава - рахунок-фактура № 09-12 від 28/09/99 |
| | 7 | 1999-10-02 00:00:00.000 | 2 | Договір № 7 | Підстава - накладна № 85 від 21/09/99 |
| * | NULL | NULL | NULL | NULL | NULL |

Рисунок 3.30

| | ContractNumber | Product | Amount | PricePerItem |
|---|---|---|---|---|
| ▶ | 1 | відеомагнітофон | 12 | 722,33 |
| | 1 | комп'ютер | 24 | 1554,22 |
| | 1 | магнітофон | 25 | 655,12 |
| | 1 | стереосистема | 12 | 220,45 |
| | 1 | телевізор | 10 | 1253,45 |
| | 2 | відеомагнітофон | 8 | 450,67 |
| | 2 | комп'ютер | 43 | 1453,18 |
| | 2 | магнітофон | 5 | 455,14 |
| | 2 | стереосистема | 11 | 511,43 |
| | 3 | магнітофон | 11 | 544,00 |
| | 3 | монітор | 85 | 545,32 |
| | 3 | телевізор | 52 | 899,99 |
| | 4 | магнітофон | 22 | 323,19 |
| | 4 | принтер | 41 | 350,77 |
| | 4 | стереосистема | 27 | 330,55 |
| | 4 | телевізор | 56 | 990,56 |
| | 5 | відеомагнітофон | 17 | 850,12 |
| | 5 | магнітофон | 33 | 585,67 |
| | 5 | монітор | 44 | 590,23 |
| | 5 | телевізор | 14 | 860,33 |
| | 6 | комп'ютер | 32 | 1850,24 |
| | 6 | монітор | 51 | 520,95 |
| | 6 | телевізор | 34 | 810,15 |
| | 7 | комп'ютер | 15 | 1234,56 |
| | 7 | монітор | 22 | 389,75 |
| | 7 | телевізор | 62 | 900,58 |
| * | NULL | NULL | NULL | NULL |

Рисунок 3.31

### 3.2 Створення бази даних та введення даних у базу даних із використанням засобів мови SQL

У середовищі SQL Server Management Studio працювати з базою даних можна, використовуючи безпосередньо оператори мови SQL. Для цього потрібно створити один або кілька запитів. Кожен запит може містити будь-яку кількість операторів мови SQL. Розглянемо послідовність дій для створення запитів, за допомогою яких буде створено базу даних, її структуру та введено дані у базу даних.

1. Натиснути кнопку New Query на панелі інструментів.

2. Ввести текст запиту, наведений рисунку 3.32.

```
USE [master]
GO

IF  EXISTS (SELECT name FROM sys.databases WHERE name = N'dlvr')
DROP DATABASE [dlvr]
GO

USE [master]
GO

CREATE DATABASE [dlvr] ON  PRIMARY
( NAME = N'dlvr', FILENAME = N'D:\dlvr.mdf' , SIZE = 3072KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'dlvr_log', FILENAME = N'D:\dlvr_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO

ALTER DATABASE [dlvr] SET COMPATIBILITY_LEVEL = 100
GO


USE dlvr

CREATE TABLE Suppliers (SupplierID int PRIMARY KEY, SupplierName text NOT NULL, Note text)

CREATE TABLE IndividualEntrepreneurs (SupplierID int PRIMARY KEY,
                                      LastName char(20) NOT NULL,
                                      FirstName char(20) NOT NULL,
                                      SecondName char(20) NOT NULL,
                                      RegistrationNumber char(10)
                                      FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID))

CREATE TABLE LegalEntities (SupplierID int PRIMARY KEY,
                            TaxNumber char(20),
                            VATNumber char(20)
                            FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID))

CREATE TABLE Contracts (ContractNumber int IDENTITY (1,1) PRIMARY KEY,
                        ContractDate datetime,
                        SupplierID int NOT NULL,
                        ContractName text,
                        Comment text
                        FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID))

CREATE TABLE Supplied (ContractNumber int,
                       Product char(20),
                       Amount decimal(4,0) NOT NULL CHECK (Amount>0),
                       PricePerItem decimal(8,2) NOT NULL CHECK (PricePerItem>0)
                       FOREIGN KEY (ContractNumber) REFERENCES Contracts(ContractNumber)
                       PRIMARY KEY (ContractNumber,Product))
```

Рисунок 3.32

3. Виконати запит. Для цього потрібно натиснути кнопку Execute на панелі інструментів. Якщо текст запиту не містить помилок, на екрані з'явиться вікно Messages з повідомленням Command(s) completed successfully. В іншому випадку буде виведена інформація про помилки, що є в тексті запиту.

4. У разі успішного виконання запиту далі слід перевірити наявність об'єктів бази даних. У тому випадку, якщо список таблиць відразу не відображається, можна клацнути правою кнопкою миші на ім'я бази даних і в меню вибрати пункт Refresh.

5. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_create_tables.sql)

6. За допомогою операторів DDL мови SQL можна як створювати об'єкти бази даних, а й змінювати структуру раніше створених об'єктів. Припустимо, що в таблиці «Supplied» розмір поля «Amount» може не відповідати реальним значенням даних, що зберігаються. У зв'язку із цим розмір поля необхідно збільшити. Це можна зробити за допомогою наступного запиту (рисунок 3.33).

```
USE dlvr
ALTER TABLE Supplied ALTER COLUMN Amount decimal(5,0) NOT NULL
```

Рисунок 3.33

7. Послідовність дій під час створення та виконання запиту аналогічна послідовності дій, розглянутих вище. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_alter_tables.sql).

8. Розглянемо ще один приклад використання оператора ALTER TABLE. Припустимо, що після того, як базу даних було створено, з'ясувалося, що кожен договір постачання також характеризується формою оплати. Форма оплати визначає порядок взаєморозрахунків із постачальником за поставлену продукцію. Можуть, наприклад, використовуватись такі типи оплати як готівкова чи безготівкова. Крім

того, будь-якої миті може з'явитися якась нова форма оплати. Для того щоб відобразити в базі даних таку структурну зміну, необхідно створити таблицю «PaymentType» з полями «PaymentTypeID» та «PaymentTypeName» та зв'язати її з таблицею Договори. Це можна зробити за допомогою запиту, який наведено на рисунку 3.34. Послідовність дій під час створення та виконання запиту аналогічна послідовності дій, розглянутих вище. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_alter_tables1.sql). Після виконання запиту обов'язково необхідно перевірити наявність змін у структурі бази даних.

```
USE dlvr

CREATE TABLE PaymentType (PaymentTypeID int PRIMARY KEY, PaymentTypeName char(20) NOT NULL)

ALTER TABLE Contracts ADD PaymentTypeID int NULL
    FOREIGN KEY (PaymentTypeID) REFERENCES PaymentType(PaymentTypeID)
```

Рисунок 3.34

9. Розглянутий запит дозволяє виконати необхідну структурну зміну, проте ім'я створеного зовнішнього ключа не вказано явно і буде визначено довільним чином. Це може спричинити певні незручності при роботі з цим зовнішнім ключем (наприклад, при його видаленні). Тому розглянемо змінений варіант попереднього запиту, що дозволяє вказати ім'я зовнішнього ключа. Текст запиту наведено на рисунку 3.35. Перед виконанням цього запиту необхідно видалити з бази даних зроблені за допомогою попереднього запиту структурні зміни – поле «PaymentTypeID» з таблиці «Contracts» та таблицю «PaymentType». Послідовність дій під час створення та виконання запиту аналогічна послідовності дій, розглянутих вище. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_alter_tables2.sql). Після виконання запиту обов'язково потрібно перевірити наявність змін у структурі бази даних, зокрема ім'я створеного у таблиці «Contracts» зовнішнього ключа.

```
USE dlvr

CREATE TABLE PaymentType (PaymentTypeID int PRIMARY KEY, PaymentTypeName char(20) NOT NULL)

ALTER TABLE Contracts ADD PaymentTypeID int NULL
    CONSTRAINT FK_PaymentType FOREIGN KEY (PaymentTypeID) REFERENCES PaymentType(PaymentTypeID)
```

Рисунок 3.35

10. Тепер припустимо, що ці структурні зміни виявилися непотрібними і, отже, створену таблицю і зв'язок потрібно видалити. Це також можна зробити за допомогою оператора ALTER TABLE. Текст запиту, за допомогою якого провадиться таке видалення, наведено на рисунку 3.36. Послідовність дій під час створення та виконання запиту аналогічна послідовності дій, розглянутих вище. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_alter_drop.sql). Після виконання запиту обов'язково необхідно перевірити наявність змін у структурі бази даних.

```
USE dlvr

ALTER TABLE Contracts DROP CONSTRAINT FK_PaymentType
ALTER TABLE Contracts DROP COLUMN PaymentTypeID

DROP TABLE PaymentType
```

Рисунок 3.36

11. Запити можуть містити як оператори DDL, так і оператори DML. Це дозволяє реалізувати основні операції маніпулювання даними також за допомогою запитів. Далі створимо запит, за допомогою якої у таблиці створеної бази даних буде введено дані. Текст запиту наведено на рисунку 3.37.

12. Виконати запит. Для цього потрібно натиснути кнопку Execute на панелі інструментів. Якщо текст запиту не містить помилок, на екрані з'явиться вікно Messages з повідомленнями типу (1 row(s) affected). В іншому випадку буде виведена інформація про помилки, що є в тексті запиту. У разі успішного виконання запиту, далі слід перевірити наявність

інформації в таблицях бази даних. Для цього потрібно вибрати таблицю, клацнувши по ній правою кнопкою миші, і в меню вибрати пункт Open Table. Створений запит можна закрити та зберегти з довільним ім'ям (наприклад, SQLQuery_insert.sql)

```
USE dlvr

INSERT INTO Suppliers (SupplierID,SupplierName,Note) VALUES (1,'ПП Іваненко І.І','м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12-34-56, факс 22-12-33)')
INSERT INTO Suppliers (SupplierID,SupplierName,Note) VALUES (2,'ТОВ «Інтерфрут»','м. Київ, пр. Перемоги, 154, к. 3')
INSERT INTO Suppliers (SupplierID,SupplierName,Note) VALUES (3,'ПП Петренко П.П','м. Харків, пр. Науки, 55, к. 108, тел.32-18-44')
INSERT INTO Suppliers (SupplierID,SupplierName,Note) VALUES (4,'ЗАТ «Транссервіс»','м. Одеса, вул. Дерибасівська, 75')
INSERT INTO Suppliers (SupplierID,SupplierName,Note) VALUES (5,'ПП Сидорчук М.С.','м. Полтава, вул. Свободи, 15, кв. 43')

INSERT INTO LegalEntities VALUES (2, '00123987','19848521')
INSERT INTO LegalEntities VALUES (4, '29345678','25912578')

INSERT INTO IndividualEntrepreneurs VALUES (1, 'Іваненко','Ілля','Іванович','00143987')
INSERT INTO IndividualEntrepreneurs VALUES (3, 'Петренко','Павло','Петрович','12345678')
INSERT INTO IndividualEntrepreneurs VALUES (5, 'Сидорчук','Микита','Степанович','09876541')

INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19990901',1,'Договір № 1','Підстава - накладна №34 від 30/08/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19990910',1,'Договір № 2','Підстава - рахунок-фактура № 08-78 від 28/08/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19990910',3,'Договір № 3','Підстава - рахунок-фактура № 08-78 від 28/08/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19990923',3,'Договір № 4','Підстава - замовлення № 56 від 28/08/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19990924',2,'Договір № 5','Підстава - накладна № 74 від 11/09/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19991001',1,'Договір № 6','Підстава - рахунок-фактура № 09-12 від 28/09/99')
INSERT INTO Contracts (ContractDate,SupplierID,ContractName,Comment) VALUES ('19991002',2,'Договір № 7','Підстава - накладна № 85 від 21/09/99')

INSERT INTO Supplied VALUES (1,'телевізор',10,1253.45)
INSERT INTO Supplied VALUES (1,'стереосистема',12,220.45)
INSERT INTO Supplied VALUES (1,'відеомагнітофон',12,722.33)
INSERT INTO Supplied VALUES (1,'комп'ютер',24,1554.22)
INSERT INTO Supplied VALUES (1,'магнітофон',25,655.12)
INSERT INTO Supplied VALUES (2,'магнітофон',5,455.14)
INSERT INTO Supplied VALUES (2,'відеомагнітофон',8,450.67)
INSERT INTO Supplied VALUES (2,'стереосистема',11,511.43)
INSERT INTO Supplied VALUES (2,'комп'ютер',43,1453.18)
INSERT INTO Supplied VALUES (3,'магнітофон',11,544.00)
INSERT INTO Supplied VALUES (3,'телевізор',52,899.99)
INSERT INTO Supplied VALUES (3,'монітор',85,545.32)
INSERT INTO Supplied VALUES (4,'магнітофон',22,323.19)
INSERT INTO Supplied VALUES (4,'стереосистема',27,330.55)
INSERT INTO Supplied VALUES (4,'принтер',41,350.77)
INSERT INTO Supplied VALUES (4,'телевізор',56,990.56)
INSERT INTO Supplied VALUES (5,'телевізор',14,860.33)
INSERT INTO Supplied VALUES (5,'відеомагнітофон',17,850.12)
INSERT INTO Supplied VALUES (5,'магнітофон',33,585.67)
INSERT INTO Supplied VALUES (5,'монітор',44,590.23)
INSERT INTO Supplied VALUES (6,'комп'ютер',32,1850.24)
INSERT INTO Supplied VALUES (6,'телевізор',34,810.15)
INSERT INTO Supplied VALUES (6,'монітор',51,520.95)
INSERT INTO Supplied VALUES (7,'комп'ютер',15,1234.56)
INSERT INTO Supplied VALUES (7,'монітор',22,389.75)
INSERT INTO Supplied VALUES (7,'телевізор',62,900.58)
```

Рисунок 3.37

В результаті виконання практичних завдань було створено дві практично однакові бази даних. Тим не менш, у цих базах даних можуть бути певні відмінності. Необхідно проаналізувати об'єкти баз даних, виявити відмінності (якщо такі є) і встановити причину їх появи. Також необхідно створити у новій базі даних dlvr діаграму. При створенні діаграми слід звернути увагу на те, що зв'язки між таблицями в діаграмі з'являються автоматично при додаванні до діаграми. Треба також порівняти діаграму dlvr з діаграмою delivery.

У процесі роботи з базою даних може виникнути потреба копіювати файли бази даних з метою створення, наприклад, резервної копії. У СУБД Microsoft SQL Server є кілька способів створення копій бази даних. Одним із найпростіших способів є відключення та підключення бази даних. Для відключення та підключення бази даних потрібно виконати таку послідовність дій.

1. У вікні Object Explorer вибрати базу даних, що відключається (в даному випадку – раніше створену базу даних delivery (dlvr).

2. Клацнути по базі даних правою кнопкою миші і в меню вибрати пункт Tasks. Цей пункт відповідає підменю, в якому потрібно вибрати пункт Detach…. Потім у вікні Detach Database потрібно натиснути кнопку ОК. В результаті вимкнена база даних зникне зі списку баз даних, а файли бази даних стануть доступними для виконання файлових маніпуляцій.

3. Вимкнену базу даних можна знову підключити. Для цього у вікні Object Explorer потрібно клацнути правою кнопкою миші по пункту Databases і в меню вибрати пункт Attach .... Потім у вікні Attach Databases потрібно натиснути кнопку Add і вибрати базу даних, що підключається, вказавши місцезнаходження її файлів. Після цього необхідно натиснути кнопку ОК. В результаті база даних з'явиться у списку баз даних

4. Перевірити можливість роботи з базою даних (тобто наявність об'єктів знову підключеної бази даних, наявність даних у таблицях тощо).

Для збереження результатів виконання практичних завдань треба відключити створені бази даних та зберегти файли delivery.mdf, delivery_log.ldf, dlvr.mdf, dlvr_log.ldf.

Також треба зберегти файли, що містять тексти запитів:

SQLQuery_create_tables.sql, SQLQuery_alter_tables.sql,

SQLQuery_alter_tables1.sql, SQLQuery_alter_tables2.sql,

SQLQuery_alter_drop.sql, SQLQuery_insert.sql

### 3.2.4 Звітність про виконання практичних завдань теми 3

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) навести скріншоти створених таблиць бази даних у режимі таблиці та режимі конструктора;

3) відобразити структуру створеної бази даних та відношень між таблицями;

4) навести тексти запитів, за допомогою яких було створено базу даних та введено дані до бази даних;

5) зробити висновки за результатами виконання практичних завдань.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлений за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 3.3 Питання для самоперевірки по темі 3

1. Поясніть, чому для зберігання інформації було обрано саме таку структуру бази даних. Які недоліки характерні для використовуваної структури бази даних?
2. Проаналізуйте дані, що зберігаються у таблиці «Supplied». Поясніть, чи доцільно зберігати у такому вигляді дані саме про такі товари (при цьому як позитивну, так і негативну відповідь треба обґрунтувати).
3. Чи можна змінити структуру бази даних Microsoft SQL Server, і якщо так, то як?
4. Як створити базу даних засобами СУБД Microsoft SQL Server?
5. Перерахуйте основні типи даних СУБД Microsoft SQL Server і дайте стислу характеристику кожному типу.
6. Які оператори входять до підмножини DDL мови SQL?
7. Для чого призначений оператор CREATE DATABASE? Яка структура цього оператора?
8. Для чого призначений оператор ALTER DATABASE? Яка структура цього оператора?
9. Для чого призначений оператор DROP DATABASE? Яка структура цього оператора?
10. Для чого призначений оператор CREATE TABLE? Яка структура цього оператора?
11. Для чого призначений оператор ALTER TABLE? Яка структура цього оператора?

12. Для чого призначений оператор DROP TABLE? Яка структура цього оператора?

13. Властивість «IDENTITY». Призначення, переваги і недоліки. Де та як ця властивість використовувалися при проектуванні?

14. Як модифікувати структуру бази даних, яку було створено засобами СУБД Microsoft SQL Server (додати нову таблицю або змінити структуру існуючої таблиці)?

15. Які оператори входять до підмножини DML мови SQL?

16. Для чого призначений оператор INSERT? Яка структура цього оператора?

17. Для чого призначений оператор DELETE? Яка структура цього оператора?

18. Для чого призначений оператор UPDATE? Яка структура цього оператора?

19. Як додати в таблицю новий запис?

20. Як видалити з таблиці один або кілька записів?

21. Як змінити формат представлення календарних дат?

22. Що необхідно для встановлення відношень посилальної цілісності між таблицями?

23. Як змінити тип відношень посилальної цілісності між таблицями?

24. Що таке первинний ключ таблиці? Як встановити первинний ключ для таблиці?

25. Як створити складений первинний ключ (до складу якого входять кілька полів)?

26. Що таке зовнішній ключ таблиці? Як встановити зовнішній ключ для таблиці?

27. Як для поля таблиці встановити властивість унікальності (заборона введення повторюваних значень)?

28. Як змінити порядок розташування полів у структурі таблиці?

# ТЕМА 4
# ОБРОБКА ДАНИХ ЗАСОБАМИ МОВИ SQL У СЕРЕДОВИЩІ СУБД MICROSOFT SQL SERVER

## 4.1 Теоретичні відомості

### 4.1.1 Загальні відомості про засоби обробки даних мови SQL

Мова запитів DQL (Data Query Language) найбільш відома користувачам реляційної бази даних, не дивлячись на те, що він включає всього одну команду (оператор) SELECT. Ця команда разом зі своїми численними опціями і пропозиціями використовується для формування запитів до реляційної бази даних. Досить часто мову запитів DQL розглядають як складову DML (Data Manipulation Language).

Оператор SELECT – один з найбільш важливих і найпоширеніших операторів SQL. Він дозволяє робити вибірки даних з таблиць і перетворювати до потрібного виду отримані результати. Будучи дуже потужним, він здатний виконувати дії, еквівалентні операторам реляційної алгебри, причому в межах єдиної виконуваної команди. При його допомозі можна реалізувати складні і розгалужені умови відбору даних з різних таблиць.

Оператор SELECT є засобом, який повністю абстрагований від питань представлення даних, що допомагає сконцентрувати увагу на проблемах доступу до даних та обробки даних. Приклади його використання наочно демонструють один із базових принципів великих (так званих, промислових) СУБД: засоби зберігання даних і доступу до них відокремлені від засобів представлення даних. Операції над даними робляться в масштабі наборів цих даних, а не окремих записів.

Оператор SELECT має такий загальний формат:

```
SELECT [ALL | DISTINCT ] {*|[ім'я_стовпця  [AS нове_ім'я]]} [,..n]
   FROM  ім'я_таблиці [[AS] псевдонім] [,..n]
   [WHERE  <умова_пошуку>]
   [GROUP BY  ім'я_стовпця [,..n]]
   [HAVING <критерії вибору груп>]
   [ORDER BY ім'я_стовпця [,..n]]
```

Оператор SELECT визначає поля (стовпці), які входитимуть в результат виконання запиту, тобто у результуючу таблицю. У списку вони розділяються комами і приводяться в такій послідовності, в якій мають бути представлені в результаті запиту. Якщо використовується ім'я поля, що містить пропуски або роздільники, його слід взяти в квадратних дужок. Символом * можна вибрати усі поля, а замість імені поля застосувати вираження з декількох імен.

Якщо обробляється ряд таблиць, то (за наявності однойменних полів в різних таблицях) в списку полів використовується повна специфікація поля, тобто. Ім'я_таблиці.Ім'я_поля.

Пропозиція FROM задає імена таблиць і представлень, які містять поля, перераховані в операторові SELECT. Необов'язковий параметр псевдоніма – це скорочення, що встановлюється для імені таблиці.

Обробка елементів оператора SELECT виконується в наступній послідовності:

• FROM – визначаються імена використовуваних таблиць та/або представлень;

• WHERE – виконується фільтрація рядків об'єкту відповідно до заданих умов;

• GROUP BY – утворюються групи рядків, що мають одно і те ж значення у вказаному стовпці;

• HAVING – фільтруються групи рядків об'єкту відповідно до вказаної умови;

• SELECT – встановлюється, які стовпці мають бути присутніми у вихідних даних;

• ORDER BY – визначається впорядкованість результатів виконання операторів.

Порядок пропозицій і фраз в операторові SELECT не може бути змінений. Тільки дві пропозиції SELECT і FROM є обов'язковими, усі інші можуть бути опущені. SELECT – закрита операція, тобто результат запиту до таблиці (або таблиць) завжди є таблицею.

Параметр WHERE визначає критерій відбору записів з вхідного набору. Але в таблиці можуть бути присутніми записи (дублікати), що повторюються. Предикат ALL задає включення у вихідний набір усіх дублікатів, відібраних за критерієм WHERE. Немає необхідності вказувати ALL явно, оскільки це значення діє за умовчанням.

Результат виконання запиту може містити значення, що дублюються, оскільки на відміну від операцій реляційної алгебри оператор SELECT не виключає значень, що повторюються, при виконанні вибірки даних.

Предикат DISTINCT слід застосовувати в тих випадках, коли вимагається відкинути блоки даних, що містять дублюючі записи у вибраних полях. Значення для кожного з приведених в інструкції SELECT полів мають бути унікальними, щоб запис, що утримує їх, зміг увійти до вихідного набору.

Причиною обмеження в застосуванні DISTINCT є та обставина, що його використання може різко уповільнити виконання запитів.

За допомогою параметра WHERE користувач визначає, які блоки даних з приведених в списку FROM таблиць з'являться в результаті запиту. За ключовим словом WHERE слідує перелік умов пошуку, що визначають ті рядки, які мають бути вибрані при виконанні запиту. Існує п'ять основних типів умов пошуку (або предикатів):

• порівняння: порівнюються результати обчислення одного вираження з результатами обчислення іншого;

- діапазон: перевіряється, чи потрапляє результат обчислення вираження в заданий діапазон значень;
- приналежність множині: перевіряється, чи належить результат обчислень вираження заданій множині значень;
- відповідність шаблону: перевіряється, чи відповідає деяке строкове значення заданому шаблону;
- значення NULL: перевіряється, чи містить цей стовпець визначник NULL (невідоме значення).

У мові SQL можна використовувати наступні оператори порівняння:

- = – рівність;
- < – менше;
- > – більше;
- <= – менше або рівно;
- >= – більше або рівно;
- <> – не рівно.

Складніші предикати можуть бути побудовані за допомогою логічних операторів AND, OR або NOT, а також дужок, використовуваних для визначення порядку обчислення вираження. Обчислення вираження в умовах виконується за наступними правилами:

- вираження обчислюється зліва направо;
- першими обчислюються підвирази в дужках;
- оператори NOT виконуються до виконання операторів AND і OR;
- оператори AND виконуються до виконання операторів OR.

Для усунення будь-якої можливої неоднозначності рекомендується використовувати дужки.

Оператор BETWEEN використовується для пошуку значення усередині деякого інтервалу, визначуваного своїми мінімальним і максимальним значеннями. При цьому вказані значення включаються в умову пошуку.

При використанні заперечення NOT BETWEEN потрібно, щоб значення, що перевіряється, лежало поза межами заданого діапазону.

Оператор IN використовується для порівняння деякого значення із списком заданих значень, при цьому перевіряється, чи відповідає результат обчислення вираження одному зі значень в наданому списку. За допомогою оператора IN може бути досягнутий той же результат, що і у разі застосування оператора OR, проте оператор IN виконується швидше.

NOT IN використовується для відбору будь-яких значень, окрім тих, які вказані в представленому списку.

За допомогою оператора LIKE можна виконувати порівняння вираження із заданим шаблоном, в якому допускається використання символів-замінників:

- символ % – замість цього символу може бути підставлена будь-яка кількість довільних символів;
- символ _ замінює один символ рядка;
- [ ] – замість символу рядка буде підставлений один з можливих символів, вказаний в цих обмежувачах;
- [^] – замість відповідного символу рядка будуть підставлені усі символи, окрім вказаних в обмежувачах.

Оператор IS NULL використовується для порівняння поточного значення зі значенням NULL – спеціальним значенням, що вказує на відсутність будь-якого значення. NULL – це не те ж саме, що знак пропуску (пропуск (або пробіл) – допустимий символ) або нуль (0 – допустиме число). NULL відрізняється і від рядка нульової довжини (порожнього рядка).

IS NOT NULL використовується для перевірки присутності значення в полі.

У загальному випадку рядка в результуючій таблиці SELECT-запиту ніяк не впорядковані. Проте їх можна необхідним чином відсортувати, для чого в оператор SELECT використовується фраза ORDER BY, яка сортує дані вихідного набору в заданій послідовності. Сортування може

виконуватися по декількох полях, в цьому випадку вони перераховуються за ключовим словом ORDER BY через кому. Спосіб сортування задається ключовим словом, що вказується у рамках параметра ORDER BY слідом за назвою поля, по якій виконується сортування. За умовчанням реалізується сортування за збільшенням. Явно вона задається ключовим словом ASC. Для виконання сортування в зворотній послідовності необхідно після імені поля, по якому вона виконується, вказати ключове слово DESC. Фраза ORDER BY дозволяє упорядкувати вибрані записи в порядку зростання або убування значень будь-якого стовпця або комбінації стовпців, незалежно від того, є присутніми ці стовпці в таблиці результату або ні. Фраза ORDER BY завжди має бути останнім елементом в операторові SELECT.

У фразі ORDER BY може бути вказане і більше одного елементу. Головний (перший) ключ сортування визначає загальну впорядкованість рядків результуючої таблиці. Якщо в усіх рядках результуючої таблиці значення головного ключа сортування є унікальними, немає необхідності використовувати додаткові ключі сортування. Проте, якщо значення головного ключа не унікальні, в результуючій таблиці буде присутніми декілька рядків з одним і тим же значенням старшого ключа сортування. В цьому випадку, можливо, доведеться упорядкувати рядки з одним і тим же значенням головного ключа по якому-небудь додатковому ключу сортування.

### 4.1.2 З'єднання і теоретико-множинні операції над відношеннями

Розглянемо основні операції над відношеннями, які можуть представляти інтерес з точки зору витягання даних з реляційних таблиць. Це об'єднання, перетин, різниця, розширений декартовий добуток відношень, а також спеціальні операції над відношеннями: вибірка, проекція і з'єднання.

Для ілюстрації теоретико-множинних операцій над відношеннями введемо абстрактні відношення (таблиці) з деякими атрибутами (полями). Їх структури наведені на рисунках 4.1 та 4.2 відповідно.

| **Відношення R** | |
|---|---|
| **R.a1** | **R.a2** |
| A | 1 |
| A | 2 |
| B | 1 |
| B | 3 |
| B | 4 |

Рисунок 4.1

| **Відношення S** | |
|---|---|
| **S.b1** | **S.b2** |
| 1 | h |
| 2 | g |
| 3 | h |

Рисунок 4.2

Запити SQL, за допомогою яких створюються ці таблиці, мають вигляд.

```
CREATE TABLE R
((a1 CHAR(1), a2 INT, PRIMARY KEY(a1, a2))

CREATE TABLE S
((b1 INT PRIMARY KEY, b2 CHAR(1))
```

Операції вибірки і проекції є унарними, оскільки вони працюють з одним відношенням.

Операція вибірки – побудова горизонтальної підмножини, тобто підмножини кортежів, що мають задані властивості.

Операція вибірки працює з одним відношенням R і визначає результуюче відношення, яке містить тільки ті кортежі (рядки) відношення R, які задовольняють заданій умові F (предикату).

Операція вибірки в SQL записується, наприклад, таким чином:

```
SELECT a1, a2
FROM R
WHERE a2=1
```

Операція проекції – це побудова вертикальної підмножини відношення, тобто підмножини кортежів, що отримується вибором одних і виключенням інших атрибутів.

Операція проекції працює з одним відношенням R і визначає нове відношення, яке містить вертикальну підмножину відношення R, що створюється за допомогою витягання значень вказаних атрибутів і виключення з результату рядків-дублікатів.

Операція проекції в SQL записується таким чином:

```
SELECT b2
FROM S
```

Декартовий добуток R x S двох відношень (двох таблиць) визначає нове відношення - результат конкатенації (тобто зчеплення) кожного кортежу (кожному запису) з відношення R з кожним кортежем (кожним записом) з відношення S.

```
RxS={(a, 1, 1, h) (a, 2, 1, h),
    (    (b, 1, 1, h), .. }
```

Декартовий добуток відношень в SQL має вигляд.

```
SELECT R.a1, R.a2, S.b1, S.b2
FROM R, S
```

Результат декартового добутку двох відношень показаний на рисунку 4.3.

| R x S | | | |
|---|---|---|---|
| **R.a1** | **R.a2** | **S.b1** | **S.b2** |
| a | 1 | 1 | h |
| a | 1 | 2 | g |
| a | 1 | 3 | h |
| a | 2 | 1 | h |
| a | 2 | 2 | g |
| a | 2 | 3 | h |
| b | 1 | 1 | h |
| b | 1 | 2 | g |
| b | 1 | 3 | h |
| b | 3 | 1 | h |
| b | 3 | 2 | g |
| b | 3 | 3 | h |
| b | 4 | 1 | h |
| b | 4 | 2 | g |
| b | 4 | 3 | h |

Рисунок 4.3

Якщо одно відношення має N записів і K полів, а інше M записів і L полів, то відношення з їх декартовим добутком міститиме NxM записів і K+L полів. Вихідні відношення можуть містити поля з однаковими іменами, тоді імена полів міститимуть назви таблиць у вигляді префіксів для забезпечення унікальності імен полів у відношенні, отриманому як результат виконання декартового добутку.

Проте у такому вигляді (рисунок 4.3) відношення містить більше інформації, чим зазвичай необхідно користувачеві. Як правило, користувачів цікавить лише деяка частина усіх комбінацій записів в декартовому добутку, що задовольняє деякій умові. Тому замість декартового добутку зазвичай використовується одна з найважливіших операцій реляційної алгебри – операція з'єднання, яка є похідною від операції декартового добутку. З точки зору ефективності реалізації в реляційних СУБД ця операція - одна з найважчих і часто входить до числа

головних причин, що викликають властиві усім реляційним системам проблеми з продуктивністю.

З'єднання – це процес, коли дві або більше таблиць поєднуються в одну. Здатність об'єднувати інформацію з декількох таблиць або запитів у вигляді одного логічного набору даних обумовлює широкі можливості SQL.

У мові SQL для завдання типу з'єднання таблиць в логічний набір записів, з якого вибиратиметься необхідна інформація, використовується операція JOIN в пропозиції FROM.

Формат операції:

FROM ім'я_таблиці_1 {INNER | LEFT | RIGHT}

JOIN ім'я_таблиці_2

ON умова_з'єднання

Існують різні типи операцій з'єднання :

- тета-з'єднання;
- з'єднання по еквівалентності;
- природне з'єднання;
- зовнішнє з'єднання;
- напівз'єднання.

Операція тета-з'єднання визначає відношення, яке містить кортежі з декартових добутків відношень R і S, що задовольняють предикату F. Предикат F має вигляд , де замість може бути вказаний один з операторів порівняння ( >,<=, =, <> ).

Якщо предикат F містить тільки оператор рівності ( = ), то з'єднання називається з'єднанням по еквівалентності.

Результат з'єднання двох відношень показаний на рисунку 4.4.

| R.a1 | R.a2 | S.b1 | S.b2 |
|---|---|---|---|
| a | 1 | 1 | h |
| a | 2 | 2 | g |
| b | 3 | 3 | h |
| b | 1 | 1 | h |

Рисунок 4.4

Операція тета-з'єднання в мові SQL називається INNER JOIN (внутрішнє з'єднання) і використовується, коли треба включити усі рядки з обох таблиць, що задовольняють умові об'єднання. Внутрішнє з'єднання має місце і тоді, коли в пропозиції WHERE порівнюються значення полів з різних таблиць. В цьому випадку будується декартовий добуток рядків першою і другою таблиць, а з отриманого набору даних відбираються записи, що задовольняють умовам об'єднання.

В умовах з'єднання можуть брати участь поля, що відносяться до одного і тому ж типу даних і таких, що містять один і той же вид даних, але вони не обов'язково повинні мати однакові імена.

Блоки даних з двох таблиць об'єднуються, як тільки у вказаних полях будуть знайдені співпадаючі значення.

Якщо в пропозиції FROM перераховано декілька таблиць і при цьому не вживається специфікація JOIN, а для вказівки відповідності полів з таблиць використовується умова в пропозиції WHERE, то деякі реляційні СУБД (наприклад, Access) оптимізують виконання запиту, інтерпретуючи його як з'єднання.

Якщо перераховувати ряд таблиць або запитів і не вказувати умови з'єднання, в якості початкової таблиці буде вибрано декартовий (прямий) добуток усіх таблиць.

Тета-з'єднання відношень в SQL має вигляд.

```
SELECT R.a1, R.a2, S.b1, S.b2
FROM R, S
WHERE R.a2=S.b1
```

чи

```
SELECT R.a1, R.a2, S.b1, S.b2
FROM R INNER JOIN S ON  R.a2=S.b1
```

Природним з'єднанням називається з'єднання по еквівалентності двох відношень R і S, виконане по усіх загальних атрибутах, з результатів якого виключається по одному екземпляру кожного загального атрибуту.

Результат природного з'єднання двох відношень показаний на рисунку 4.5.

| R.a1 | R.a2 або S.b1 | S.b2 |
|---|---|---|
| a | 1 | h |
| a | 2 | g |
| b | 3 | h |
| b | 1 | h |

Рисунок 4.5

Природне з'єднання відношень в SQL має вигляд.

```
SELECT R.a1, R.a2, S.b2
FROM R, S
WHERE R.a2=S.b1
```

чи

```
SELECT R.a1, S.b1, S.b2
FROM R INNER JOIN S ON  R.a2=S.b1
```

Зовнішнє з'єднання схоже на внутрішнє, але в результуючий набір даних включаються також записи провідної таблиці з'єднання, які об'єднуються з порожньою безліччю записів іншої таблиці.

Яка з таблиць буде такою, що веде, визначає вид з'єднання. LEFT – ліве зовнішнє з'єднання, ведучою є таблиця, розташована зліва від виду

з'єднання; RIGHT – праве зовнішнє з'єднання, провідна таблиця розташована праворуч від виду з'єднання.

Лівим зовнішнім з'єднанням називається з'єднання, при якому кортежі відношення R, що не мають співпадаючих значень в загальних стовпцях відношення S, також включаються в результуюче відношення.

Результат лівого зовнішнього з'єднання двох відношень показаний на рисунку 4.6.

| R.a1 | R.a2 | S.b1 | S.b2 |
|---|---|---|---|
| a | 1 | 1 | h |
| a | 2 | 2 | g |
| b | 1 | 1 | h |
| b | 3 | 3 | h |
| b | 4 | null | null |

Рисунок 4.6

Ліве зовнішнє з'єднання відношень в SQL має вигляд.

```
SELECT R.a1,  R.a2, S.b1, S.b2
FROM R LEFT JOIN S ON R.a2=S.b1
```

Існує і праве зовнішнє з'єднання, зване так тому, що в результуючому відношенні містяться усі кортежі правого відношення. Крім того, є і повне зовнішнє з'єднання, в його результуюче відношення поміщаються усі кортежі з обох відношень, а для позначення неспівпадаючих значень кортежів в нім використовуються визначники NULL.

Праве зовнішнє з'єднання відношень в SQL має вигляд.

```
SELECT R.a1,  R.a2, S.b1, S.b2
FROM R RIGHT JOIN S ON  R.a2=S.b1
```

Операція напівз'єднання визначає відношення, що містить ті кортежі відношення R, які входять в з'єднання відношень R і S .

Результат напівз'єднання двох відношень показаний на рисунку 4.7.

| | |
|---|---|
| **R.a1** | **R.a2** |
| a | 1 |
| a | 2 |
| b | 3 |
| b | 1 |

Рисунок 4.7

Напівз'єднання відношень в SQL має вигляд.

```
SELECT R.a1, R.a2
FROM R, S
WHERE R.a2=S.b1
```

чи

```
SELECT R.a1,   R.a2
FROM R INNER JOIN S ON R.a2=S.b1
```

Об'єднання ( UNION ) відношень R і S можна отримати в результаті їх конкатенації з утворенням одного відношення з виключенням кортежів-дублікатів. При цьому відношення R і S мають бути сумісні, тобто мати однакову кількість полів із співпадаючими типами даних. Інакше кажучи, відношення мають бути сумісні по об'єднанню.

Об'єднанням двох таблиць R і S є таблиця, що містить усі рядки, які є в першій таблиці R, в другій таблиці S або в обох таблицях відразу.

Об'єднання відношень в SQL має вигляд.

```
SELECT R.a1, R.a2
FROM R
UNION
SELECT S.b2, S.b1
FROM S
```

Операція перетину ( INTERSECT) визначає відношення, яке містить кортежі, присутні як у відношенні R, так і у відношенні S. Відношення R і S мають бути сумісні по об'єднанню, тобто мати однакову кількість стовпців, типи даних у яких є сумісними.

Перетином двох таблиць R і S є таблиця, що містить усі рядки, присутні в обох вихідних таблицях одночасно.

Перетин відношень в SQL має вигляд.

```
SELECT R.a1, R.a2
FROM R, S
WHERE R.a1=S.b1 AND R.a2=S.b2
```

чи

```
SELECT R.a1, R.a2
FROM R
WHERE R.a1 IN
   (    (SELECT S.b1 FROM S
         WHERE S.b1=R.a1) AND R.a2 IN
        (         (SELECT S.b2
                   FROM S
                   WHERE S.b2=R.a2)
```

Різниця (EXCEPT) двох відношень R і S складається з кортежів, які є відносно R, але відсутні відносно S. Причому відношення R і S мають бути сумісні по об'єднанню.

Різницею двох таблиць R і S є таблиця, що містить усі рядки, які є присутніми в таблиці R, але відсутні в таблиці S.

Різниця відношень в SQL має вигляд.

```
SELECT R.a1, R.a2
FROM R
WHERE NOT EXISTS
```

```
(    (SELECT S.b1, S.b2
      FROM S
      WHERE S.b1=R.a2 AND S.b2=R.a1)
```

### 4.1.3 Обчислення і підведення підсумків в запитах

У загальному випадку для створення обчислюваного (похідного) поля в списку SELECT слід вказати деяке вираження мови SQL. У цих виразах застосовуються арифметичні операції складання, віднімання, множення і ділення, а також вбудовані функції мови SQL. Можна вказати ім'я будь-якого стовпця (поля) таблиці або запиту, але використовувати ім'я стовпця тільки тієї таблиці або запиту, які вказані в списку пропозиції FROM відповідної інструкції. При побудові складних виразів можуть знадобитися дужки.

Стандарти SQL дозволяють явним чином задавати імена стовпців результуючої таблиці, для чого застосовується фраза AS.

За допомогою підсумкових (агрегатних) функцій у рамках SQL-запиту можна отримати ряд узагальнювальних статистичних відомостей про множину відібраних значень вихідного набору.

Користувачеві доступні наступні основні підсумкові функції:

- Count (Вираження) – визначає кількість записів у вихідному наборі SQL-запиту;
- Min/Max (Вираження) – визначають найменше і найбільше з множини значень в деякому полі запиту;
- Avg (Вираження) – ця функція дозволяє розрахувати середнє значення множини значень, що зберігаються в певному полі відібраних запитом записів. Воно є арифметичним середнім значенням, тобто сумою значень, що ділиться на їх кількість.
- Sum (Вираження) – обчислює суму множини значень, що містяться в певному полі відібраних запитом записів.

Найчастіше вираженням виступають імена стовпців. Вираження може обчислюватися і по значеннях декількох таблиць.

Усі ці функції оперують зі значеннями в єдиному стовпці таблиці або з арифметичним вираженням і повертають єдине значення. Функції COUNT, MIN і MAX застосовані як до числових, так і до нечислових полів, тоді як функції SUM і AVG можуть використовуватися тільки у разі числових полів, за винятком COUNT(*). При обчисленні результатів будь-яких функцій спочатку виключаються усі порожні значення, після чого необхідна операція застосовується тільки до конкретних значень стовпця, що залишилися. Варіант COUNT(*) – особливий випадок використання функції COUNT, його призначення полягає в підрахунку усіх рядків в результуючій таблиці, незалежно від того, містяться там порожні, такі, що дублюються або будь-які інші значення.

Якщо до застосування узагальнювальної функції необхідно виключити значення, що дублюються, слід перед ім'ям стовпця у визначенні функції помістити ключове слово DISTINCT. Воно не має сенсу для функцій MIN і MAX, проте його використання може вплинути на результати виконання функцій SUM і AVG, тому необхідно заздалегідь обдумати, чи повинне воно бути присутнім у кожному конкретному випадку. Крім того, ключове слово DISTINCT може бути вказане у будь-якому запиті не більше одного разу.

Дуже важливо відмітити, що підсумкові функції можуть використовуватися тільки в списку пропозиції SELECT і у складі пропозиції HAVING. У усіх інших випадках це неприпустимо. Якщо список в пропозиції SELECT містить підсумкові функції, а в тексті запиту відсутня фраза GROUP BY, що забезпечує об'єднання даних в групи, то жоден з елементів списку пропозиції SELECT не може включати яких-небудь посилань на поля, за винятком ситуації, коли поля виступають аргументами підсумкових функцій.

Часто в запитах вимагається формувати проміжні підсумки, що зазвичай відображається появою в запиті фрази «для кожного». Для цієї мети в операторові SELECT використовується пропозиція GROUP BY. Запит, в якому є присутнім GROUP BY, називається групуючим запитом,

оскільки в нім групуються дані, отримані в результаті виконання операції SELECT, після чого для кожної окремої групи створюється єдиний сумарний рядок. Стандарт SQL вимагає, щоб пропозиція SELECT і фраза GROUP BY були тісно пов'язані між собою. За наявності в операторові SELECT фрази GROUP BY кожен елемент списку в пропозиції SELECT повинен мати єдине значення для усієї групи. Більше того, пропозиція SELECT може включати тільки наступні типи елементів : імена полів, підсумкові функції, константи і вирази, що включають комбінації перелічених вище елементів.

Усі імена полів, приведені в списку пропозиції SELECT, мають бути присутніми і у фразі GROUP BY за винятком випадків, коли ім'я стовпця використовується в підсумковій функції. Зворотне правило не є справедливим – у фразі GROUP BY можуть бути імена стовпців, відсутні в списку пропозиції SELECT.

Якщо спільно з GROUP BY використовується пропозиція WHERE, то воно обробляється першим, а групуванню піддаються тільки ті рядки, які задовольняють умові пошуку.

Стандартом SQL визначено, що при проведенні групування усі відсутні значення розглядаються як рівні. Якщо два рядки таблиці в одному і тому ж груповаому стовпці містять значення NULL і ідентичні значення в усіх інших непорожніх групованих стовпцях, вони поміщаються в одну і ту ж групу.

За допомогою HAVING відбиваються усі попередньо згруповані за допомогою GROUP BY блоки даних, задовольняючі заданим в HAVING умовам. Це додаткова можливість "профільтрувати" вихідний набір.

Умови в HAVING відрізняються від умов в WHERE:

- HAVING виключає з результуючого набору даних групи з результатами агрегованих значень;
- WHERE виключає з розрахунку агрегатних значень по угрупуванню записи, що не задовольняють умові;
- у умові пошуку WHERE не можна задавати агрегатні функції.

### 4.1.4 Використання підзапитів та CTE

Часто неможливо вирішити поставлене завдання шляхом виконання одного запиту. Це особливо актуально, коли при використанні умови пошуку в пропозиції WHERE значення, з яким потрібно порівнювати, заздалегідь не визначене і має бути вичислене у момент виконання оператора SELECT. У такому разі приходять на допомогу закінчені оператори SELECT, впроваджені в тіло іншого оператора SELECT. Внутрішній підзапит є також оператор SELECT, а кодування його пропозицій підкоряється тим же правилам, що і основного оператора SELECT. Зовнішній оператор SELECT використовує результат виконання внутрішнього оператора для визначення змісту остаточного результату усієї операції. Внутрішні запити можуть бути поміщені безпосередньо після оператора порівняння ( =,<=, >=, <> ) в пропозиції WHERE і HAVING зовнішнього оператора SELECT - вони отримують назву підзапитів або вкладених запитів. Крім того, внутрішні оператори SELECT можуть застосовуватися в операторах INSERT, UPDATE і DELETE.

Підзапит – це інструмент створення тимчасової таблиці, вміст якої витягається і обробляється зовнішнім оператором. Текст підзапиту має бути поміщений в дужки. До підзапитів застосовуються наступні правила і обмеження :

- фраза ORDER BY не використовується, хоча і може бути присутньою в зовнішньому підзапиті ;
- список в пропозиції SELECT складається з імен окремих стовпців або складених з них виразів - за винятком випадку, коли в підзапиті є присутнім ключове слово EXISTS ;
- за умовчанням імена стовпців в підзапиті відносяться до таблиці, ім'я якої вказане в пропозиції FROM. Проте допускається посилання і на стовпці таблиці, вказаної у фразі FROM зовнішнього запиту, для чого застосовуються кваліфіковані імена стовпців (тобто з вказівкою таблиці);

- якщо підзапит є одним з двох операндів, що беруть участь в операції порівняння, то запит повинен вказуватися в правій частині цієї операції.

Існує таки типи підзапитів :

- скалярний підзапит повертає єдине значення. В принципі, він може використовуватися скрізь, де вимагається вказати єдине значення;
- рядковий підзапит повертає множину значень, тобто значення одного стовпця таблиці, розміщені у більш ніж одному рядку;
- табличний підзапит повертає безліч значень, тобто значення одного або декількох стовпців таблиці, розміщені у більш ніж одному рядку. Він можливий скрізь, де допускається наявність таблиці.

У багатьох випадках підзапити, що використовуються у порівняннях в пропозиціях WHERE або HAVING, повертають не одно, а декілька значень. Вкладені підзапити генерують непойменоване проміжне відношення, тобто тимчасову таблицю. Таке проміжне відношення може використовуватися тільки в тому місці, де з'являється в підзапиті. До такого відношення неможливо звернутися по імені з якого-небудь іншого місця запиту. Застосовувані до підзапиту операції засновані на тих операціях, які, у свою чергу, застосовуються до множини, а саме:

- { WHERE | HAVING } вираження [ NOT ] IN ( підзапит );
- { WHERE | HAVING } вираження оператор_порівняння { ALL | SOME | ANY } ( підзапит );
- {WHERE | HAVING } [ NOT ] EXISTS ( підзапит );

Оператор IN використовується для порівняння деякого значення із списком значень, при цьому перевіряється, чи входить значення в наданий список або порівнюване значення не є елементом представленого списку. Те ж саме стосується операторів ALL, SOME, ANY. Оператори IN, ALL, SOME, ANY зазвичай використовують для роботи з результатами рядкових підзапитів.

Оператор [ NOT ] EXISTS використовується для перевірки наявності змісту (тобто рядків) у таблиці, яка утворюється в результаті виконання

підзапиту. Цей оператор зазвичай використовують для роботи з результатами табличних підзапитів.

CTE (Common Table Expression) або загальне табличне вираження – це іменований тимчасовий набір даних, який визначається за допомогою ключового слова WITH і існує лише в межах одного SQL-запиту. CTE дозволяє спростити складні запити, розбиваючи їх на логічні частини, зробити код більш читабельним, а також багаторазово використовувати проміжний результат в одному запиті, що може оптимізувати його виконання. CTE можна використовувати в операторах SELECT, INSERT, UPDATE та DELETE.

Загальний синтаксис CTE наведений на рисунку 4.8.

```
WITH cte_name AS (    SELECT query   )
SELECT *
FROM cte_name;
```

Рисунок 4.8

Основні характеристики CTE:

1 Тимчасовий характер: CTE створює віртуальну таблицю, яка існує лише під час виконання запиту і не зберігається в базі даних на постійній основі.

2 Іменування: Кожен CTE має унікальне ім'я, що робить запит більш зрозумілим.

3 Множинне використання: Результати CTE можуть бути задіяні кілька разів в межах одного основного запиту, наприклад, у SELECT, INSERT, UPDATE чи DELETE операціях.

4 Рекурсивність: CTE також можуть звертатися самі до себе, що дозволяє створювати рекурсивні запити для роботи з ієрархічними або складними структурами даних.

Переваги використання CTE:

1 Читабельність: Дозволяє розділити складні запити на менші, логічні блоки, що покращує їх зрозумілість.

2 Структурованість: Допомагає організувати код, роблячи його більш структурованим.

3 Оптимізація: Може покращити продуктивність, оскільки проміжні дані можуть зберігатися в кеші, а не обчислюватися щоразу.

4 Уникнення дублювання: Дозволяє уникнути повторення складних підзапитів у різних частинах основного запиту.

Звичайно, що наведений вище опис оператора SELECT є стислим та спрощеним. Більш докладні відомості про особливості реалізації оператора SELECT у мові Transact-SQL у разі потреби можна переглянути, наприклад, на веб-ресурсі:

https://learn.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql?view=sql-server-ver16

### 4.2 Практичне опанування теми 4

#### 4.2.1 Передумови виконання завдань теми 4

При виконанні практичних завдань теми 4 передбачається використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи Microsoft SQL Server. При цьому передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Для початку виконання практичних завдань треба підключити базу даних (delivery або dlvr), яку було створено при виконанні практичних завдань попередньої теми, тобто, «Ознайомлення з основними

особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

Увага! Всі запити, що розглядаються далі, повинні бути результативними, тобто в результаті виконання запиту повинні бути виведені один або кілька рядків. Відсутність результату запиту є ознакою помилок під час побудови запиту, невідповідності запиту наявним даним тощо. Такий запит потребує аналізу та перевірки. Також передбачається, що результати запитів ґрунтуються на тих даних, які були введені до бази даних при виконанні практичних завдань попередньої теми.

### 4.2.2 Побудова та виконання запитів

Розглянемо послідовність дій при створенні та виконанні першого запиту.

Мета запиту: вивести на екран список товарів, поставлених постачальником 1 (ПП Іваненко І.І.) за договором 1.

Для створення та виконання запиту треба.

1. Натиснути кнопку New Query на панелі інструментів.

2. Ввести текст запиту, наведений рисунку 4.9.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Suppliers.*, Contracts.ContractDate
FROM Supplied,Contracts,Suppliers
WHERE Supplied.ContractNumber=Contracts.ContractNumber AND Suppliers.SupplierID=Contracts.SupplierID AND
      (Contracts.ContractNumber=1 AND Contracts.SupplierID=1)
```

Рисунок 4.9

3. Натиснути кнопку Execute. У разі, якщо в тексті запиту немає помилок, буде виведено результат запиту. Цей результат має вигляд, наведений на рисунку 4.10.

| | ContractNumber | Product | SupplierID | SupplierName | Note | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 2 | 1 | комп'ютер | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 3 | 1 | магнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 4 | 1 | стереосистема | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 5 | 1 | телевізор | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |

Рисунок 4.10

4. Текст запиту можна зберегти як файл (наприклад, SQLQuery01_1.sql). У разі, якщо цей запит потрібно буде виконати повторно або змінити, можна відкрити файл запиту. Для цього у головному меню потрібно вибрати пункт File, а потім у вертикальному меню вибрати пункт Open, підпункт File та вибрати відповідний файл.

Як видно з тексту запиту, цей запит є багатотабличним, причому таблиці з'єднуються на основі використання природного з'єднання. У разі використання з'єднання INNER JOIN цей запит мав би вигляд, наведений на рисунку 4.11. Цей запит також необхідно створити та виконати для перевірки працездатності, а потім зберегти у файлі з ім'ям SQLQuery01_2.sql. Результат виконання, звичайно, повинен співпадати з результатом, наведеним на рисунку 4.10.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Suppliers.*, Contracts.ContractDate
FROM (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.ContractNumber=1 AND Contracts.SupplierID=1
```

Рисунок 4.11

Створення та виконання інших запитів виконується аналогічно. Тому далі для кожного запиту буде наведено лише його порядковий номер, умову запиту, його текст, результат та рекомендоване ім'я файлу для збереження.

Запит 2.

Мета запиту: вивести на екран список товарів, поставлених постачальником 1 (ПП Іваненко І.І.) у період з 05/09/1999 по 12/09/1999.

Текст запиту та результат виконання запиту наведені на рисунках 4.12 та 4.13 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery02.sql.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Suppliers.*, Contracts.ContractDate
 FROM (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
       INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
 WHERE Contracts.ContractDate BETWEEN '19990905' AND '19990912' AND Suppliers.SupplierID=1
```

Рисунок 4.12

| | ContractNumber | Product | SupplierID | SupplierName | Note | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 2 | відеомагнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 2 | 2 | комп'ютер | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 3 | 2 | магнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 4 | 2 | стереосистема | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |

Рисунок 4.13

Запит 3.

Мета запиту: вивести на екран список товарів, поставлених у 9 місяці 1999 року з виведенням найменування постачальника та дати постачання.

Текст запиту та результат виконання запиту наведені на рисунках 4.14 та 4.15 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery03.sql.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Suppliers.*, Contracts.ContractDate
 FROM (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
       INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
 WHERE MONTH(Contracts.ContractDate)=9 AND YEAR(Contracts.ContractDate)=1999
```

Рисунок 4.14

| | ContractNumber | Product | SupplierID | SupplierName | Note | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 2 | 1 | комп'ютер | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 3 | 1 | магнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 4 | 1 | стереосистема | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 5 | 1 | телевізор | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-01 00:00:00.000 |
| 6 | 2 | відеомагнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 7 | 2 | комп'ютер | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 8 | 2 | магнітофон | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 9 | 2 | стереосистема | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | 1999-09-10 00:00:00.000 |
| 10 | 3 | магнітофон | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-10 00:00:00.000 |
| 11 | 3 | монітор | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-10 00:00:00.000 |
| 12 | 3 | телевізор | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-10 00:00:00.000 |
| 13 | 4 | магнітофон | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-23 00:00:00.000 |
| 14 | 4 | принтер | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-23 00:00:00.000 |
| 15 | 4 | стереосистема | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-23 00:00:00.000 |
| 16 | 4 | телевізор | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | 1999-09-23 00:00:00.000 |
| 17 | 5 | відеомагнітофон | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 | 1999-09-24 00:00:00.000 |
| 18 | 5 | магнітофон | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 | 1999-09-24 00:00:00.000 |
| 19 | 5 | монітор | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 | 1999-09-24 00:00:00.000 |
| 20 | 5 | телевізор | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 | 1999-09-24 00:00:00.000 |

Рисунок 4.15

Запит 4.

Мета запиту: вивести на екран список договорів (номер, дата, назва договору, назва постачальника) та загальну вартість постачання за кожним договором (розмір партії помножити на ціну за штуку та підсумувати за договором). Список має бути відсортований у порядку зростання номерів договорів.

Текст запиту та результат виконання запиту наведені на рисунках 4.16 та 4.17 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery04.sql.

```
USE delivery

SELECT  Contracts.ContractNumber, Contracts.ContractDate, CAST(ContractName AS CHAR(20)) AS НазваДоговору,
        CAST(SupplierName AS CHAR(20)) AS Постачальник, SUM(Amount*PricePerItem) AS вартість
FROM  (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
GROUP BY Contracts.ContractNumber, Contracts.ContractDate, CAST(ContractName AS CHAR(20)), CAST(SupplierName AS CHAR(20))
ORDER BY Contracts.ContractNumber
```

Рисунок 4.16

| | ContractNumber | ContractDate | НазваДоговору | Постачальник | вартість |
|---|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | Договір № 1 | ПП Іваненко І.І | 77527.14 |
| 2 | 2 | 1999-09-10 00:00:00.000 | Договір № 2 | ПП Іваненко І.І | 73993.53 |
| 3 | 3 | 1999-09-10 00:00:00.000 | Договір № 3 | ПП Петренко П.П | 99135.68 |
| 4 | 4 | 1999-09-23 00:00:00.000 | Договір № 4 | ПП Петренко П.П | 85887.96 |
| 5 | 5 | 1999-09-24 00:00:00.000 | Договір № 5 | ТОВ «Інтерфрут» | 71793.89 |
| 6 | 6 | 1999-10-01 00:00:00.000 | Договір № 6 | ПП Іваненко І.І | 113321.23 |
| 7 | 7 | 1999-10-02 00:00:00.000 | Договір № 7 | ТОВ «Інтерфрут» | 82928.86 |

Рисунок 4.17

Запит 5.

Мета запиту: вивести на екран список договорів (номер, дата, назва договору, назва постачальника) та загальну вартість постачання за кожним договором (розмір партії помножити на ціну за штуку та підсумувати за договором). Список має бути відсортований у порядку зростання загальних сум за кожним договором. Також на список має бути накладена умова фільтрації, що полягає у виключенні з результату запиту записів, для яких номер договору менший за 4.

Текст запиту та результат виконання запиту наведені на рисунках 4.18 та 4.19 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery05.sql.

```
USE delivery

SELECT  Contracts.ContractNumber, Contracts.ContractDate, CAST(ContractName AS CHAR(20)) AS НазваДоговору,
        CAST(SupplierName AS CHAR(20)) AS Постачальник, SUM(Amount*PricePerItem) AS вартість
FROM  (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.ContractNumber>3
GROUP BY Contracts.ContractNumber, Contracts.ContractDate, CAST(ContractName AS CHAR(20)), CAST(SupplierName AS CHAR(20))
ORDER BY вартість
```

Рисунок 4.18

| | ContractNumber | ContractDate | НазваДоговору | Постачальник | вартість |
|---|---|---|---|---|---|
| 1 | 5 | 1999-09-24 00:00:00.000 | Договір № 5 | ТОВ «Інтерфрут» | 71793.89 |
| 2 | 7 | 1999-10-02 00:00:00.000 | Договір № 7 | ТОВ «Інтерфрут» | 82928.86 |
| 3 | 4 | 1999-09-23 00:00:00.000 | Договір № 4 | ПП Петренко П.П | 85887.96 |
| 4 | 6 | 1999-10-01 00:00:00.000 | Договір № 6 | ПП Іваненко І.І | 113321.23 |

Рисунок 4.19

Запит 6.

Мета запиту: вивести на екран відомості про найбільшу за розміром партію товару у всіх договорах із зазначенням назви товару та назви постачальника, а також номера та дати договору. Якщо найбільших за розміром партій буде декілька (тобто розміри партій різних товарів будуть однакові), вивести дані про усі такі партії.

Текст запиту та результат виконання запиту наведені на рисунках 4.20 та 4.21 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery06.sql.

```
USE delivery

SELECT  Contracts.ContractNumber, Contracts.ContractDate, SupplierName AS Постачальник, Amount AS партія
FROM  (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Amount=(SELECT MAX(Amount) FROM Supplied)
```

Рисунок 4.20

| | ContractNumber | ContractDate | Постачальник | партія |
|---|---|---|---|---|
| 1 | 3 | 1999-09-10 00:00:00.000 | ПП Петренко П.П | 85 |

Рисунок 4.21

Запит 7.

Мета запиту: вивести на екран список постачальників (із зазначенням усіх даних про постачальників), з якими не було укладено жодного договору.

Текст запиту та результат виконання запиту наведені на рисунках 4.22 та 4.23 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery07.sql.

```
USE delivery

SELECT  Suppliers.*
FROM  Suppliers
WHERE SupplierID NOT IN (SELECT SupplierID FROM Contracts)
```

Рисунок 4.22

| | SupplierID | SupplierName | Note |
|---|---|---|---|
| 1 | 4 | ЗАТ «Транссервіс» | м. Одеса, вул. Дерибасівська, 75 |
| 2 | 5 | ПП Сидорчук М.С. | м. Полтава, вул. Свободи, 15, кв. 43 |

Рисунок 4.23

Запит 8.

Мета запиту: вивести на екран список найменувань поставлених товарів із зазначенням середньої ціни постачання за одиницю (незалежно від постачальника).

Текст запиту та результат виконання запиту наведені на рисунках 4.24 та 4.25 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery08.sql.

```
USE delivery

SELECT Product, AVG(PricePerItem) AS СередняЦіна
FROM Supplied
GROUP BY Product
```

Рисунок 4.24

| | Product | СередняЦіна |
|---|---|---|
| 1 | відеомагнітофон | 674.373333 |
| 2 | комп'ютер | 1523.050000 |
| 3 | магнітофон | 512.624000 |
| 4 | монітор | 511.562500 |
| 5 | принтер | 350.770000 |
| 6 | стереосистема | 354.143333 |
| 7 | телевізор | 952.510000 |

Рисунок 4.25

Запит 9.

Мета запиту: вивести на екран список товарів (номер договору, найменування товару, кількість та ціна за одиницю, назва постачальника), для яких ціна за одиницю є більшою за середню ціну товару за увесь період постачання.

Текст запиту та результат виконання запиту наведені на рисунках 4.26 та 4.27 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery09.sql.

```
USE delivery

SELECT Contracts.ContractNumber, Product, Amount, PricePerItem, SupplierName AS Постачальник
FROM  (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE PricePerItem > (SELECT AVG(PricePerItem) FROM Supplied)
```

Рисунок 4.26

| | ContractNumber | Product | Amount | PricePerItem | Постачальник |
|---|---|---|---|---|---|
| 1 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І |
| 2 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І |
| 3 | 2 | комп'ютер | 43 | 1453.18 | ПП Іваненко І.І |
| 4 | 3 | телевізор | 52 | 899.99 | ПП Петренко П.П |
| 5 | 4 | телевізор | 56 | 990.56 | ПП Петренко П.П |
| 6 | 5 | відеомагнітофон | 17 | 850.12 | ТОВ «Інтерфрут» |
| 7 | 5 | телевізор | 14 | 860.33 | ТОВ «Інтерфрут» |
| 8 | 6 | комп'ютер | 32 | 1850.24 | ПП Іваненко І.І |
| 9 | 6 | телевізор | 34 | 810.15 | ПП Іваненко І.І |
| 10 | 7 | комп'ютер | 15 | 1234.56 | ТОВ «Інтерфрут» |
| 11 | 7 | телевізор | 62 | 900.58 | ТОВ «Інтерфрут» |

Рисунок 4.27

Запит 10.

Мета запиту: вивести на екран відомості про п'ять найдорожчих товарів (номер договору, найменування товару, кількість та ціна за одиницю, назва постачальника).

Текст запиту та результат виконання запиту наведені на рисунках 4.28 та 4.29 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery10.sql.

```
USE delivery

SELECT TOP 5 Contracts.ContractNumber, Product, Amount, PricePerItem, SupplierName AS Постачальник
FROM  (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
ORDER BY PricePerItem DESC
```

Рисунок 4.28

| | ContractNumber | Product | Amount | PricePerItem | Постачальник |
|---|---|---|---|---|---|
| 1 | 6 | комп'ютер | 32 | 1850.24 | ПП Іваненко І.І |
| 2 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І |
| 3 | 2 | комп'ютер | 43 | 1453.18 | ПП Іваненко І.І |
| 4 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І |
| 5 | 7 | комп'ютер | 15 | 1234.56 | ТОВ «Інтерфрут» |

Рисунок 4.29

Запит 11.

Мета запиту: сформувати список постачальників із зазначенням коду, та даних постачальника. При формуванні даних постачальника для усіх постачальників вивести контактні дані, для постачальників-фізичних осіб вивести прізвище та ініціали, для постачальників-юридичних осіб – податковий номер. Увага! Прізвище та ініціали або податковий номер повинні бути виведені в одному полі, тобто, або прізвище та ініціали, або податковий номер.

Текст запиту та результат виконання запиту наведені на рисунках 4.30 та 4.31 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery11.sql.

```
USE delivery

SELECT Suppliers.SupplierID, Suppliers.Note,
       ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
       AS Постачальник
FROM (Suppliers LEFT JOIN IndividualEntrepreneurs ON Suppliers.SupplierID=IndividualEntrepreneurs.SupplierID)
     LEFT JOIN LegalEntities ON Suppliers.SupplierID=LegalEntities.SupplierID
```

Рисунок 4.30

| | SupplierID | Note | Постачальник |
|---|---|---|---|
| 1 | 1 | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12... | Іваненко І.І. |
| 2 | 2 | м. Київ, пр. Перемоги, 154, к. 3 | 00123987 |
| 3 | 3 | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 | Петренко П.П. |
| 4 | 4 | м. Одеса, вул. Дерибасівська, 75 | 29345678 |
| 5 | 5 | м. Полтава, вул. Свободи, 15, кв. 43 | Сидорчук М.С. |

Рисунок 4.31

Запит 12.

Мета запиту: сформувати список договорів (із зазначенням номера, дати поставки та даних про постачальника), загальну кількість поставлених товарів та загальну суму за кожним договором. При формуванні даних постачальника для усіх постачальників вивести контактні дані, для постачальників-фізичних осіб вивести прізвище та ініціали, для постачальників-юридичних осіб – податковий номер. Увага! Прізвище та ініціали або податковий номер повинні бути виведені в одному полі, тобто, або прізвище та ініціали, або податковий номер. У результат запиту мають бути включені лише ті договори, на підставі яких товари дійсно поставлялися (тобто результат запиту не повинен потрапити так звані «порожні» договори).

Текст запиту та результат виконання запиту наведені на рисунках 4.32 та 4.33 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery12.sql.

```
USE delivery

SELECT Contracts.ContractNumber, ContractDate, SUM(Amount) AS кількість, SUM(Amount*PricePerItem) AS сума,
       CAST(Note AS CHAR(40)) AS КонтактиПостачальника,
       ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
       AS Постачальник
FROM ((Suppliers LEFT JOIN IndividualEntrepreneurs ON Suppliers.SupplierID=IndividualEntrepreneurs.SupplierID)
      LEFT JOIN LegalEntities ON Suppliers.SupplierID=LegalEntities.SupplierID)
      INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
GROUP BY Contracts.ContractNumber, ContractDate, CAST(Note AS CHAR(40)),
ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
```

Рисунок 4.32

| | ContractNumber | ContractDate | кількість | сума | КонтактиПостачальника | Постачальник |
|---|---|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | 83 | 77527.14 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |
| 2 | 2 | 1999-09-10 00:00:00.000 | 67 | 73993.53 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |
| 3 | 3 | 1999-09-10 00:00:00.000 | 148 | 99135.68 | м. Харків, пр. Науки, 55, к. 108, тел.32 | Петренко П.П. |
| 4 | 4 | 1999-09-23 00:00:00.000 | 146 | 85887.96 | м. Харків, пр. Науки, 55, к. 108, тел.32 | Петренко П.П. |
| 5 | 5 | 1999-09-24 00:00:00.000 | 108 | 71793.89 | м. Київ, пр. Перемоги, 154, к. 3 | 00123987 |
| 6 | 6 | 1999-10-01 00:00:00.000 | 117 | 113321.23 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |
| 7 | 7 | 1999-10-02 00:00:00.000 | 99 | 82928.86 | м. Київ, пр. Перемоги, 154, к. 3 | 00123987 |

Рисунок 4.33

Запит 13.

Мета запиту: сформувати список товарів (із зазначенням номера договору, дати поставки, назви товару, кількості, ціни за одиницю та назви постачальника), поставлених постачальниками 1 (ПП Іваненко І.І) та 2 (ТОВ «Інтерфрут»). Загальний список відсортувати за номером договору.

Примітка. Цей запит ілюструє особливості використання операції об'єднання (UNION). Неважко помітити, що цей запит може бути легко реалізований без використання операції об'єднання.

Текст запиту та результат виконання запиту наведені на рисунках 4.34 та 4.35 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery13.sql.

```
USE delivery

SELECT Supplied.ContractNumber, Contracts.ContractDate, Supplied.Product,
       Amount, PricePerItem, CAST(SupplierName AS CHAR(20)) AS Постачальник
FROM (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=1
UNION
SELECT Supplied.ContractNumber, Contracts.ContractDate, Supplied.Product,
       Amount, PricePerItem, CAST(SupplierName AS CHAR(20))  AS Постачальник
FROM (Suppliers INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID)
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=2
ORDER BY Supplied.ContractNumber
```

Рисунок 4.34

| | ContractNumber | ContractDate | Product | Amount | PricePerItem | Постачальник |
|---|---|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | відеомагнітофон | 12 | 722.33 | ПП Іваненко І.І |
| 2 | 1 | 1999-09-01 00:00:00.000 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І |
| 3 | 1 | 1999-09-01 00:00:00.000 | магнітофон | 25 | 655.12 | ПП Іваненко І.І |
| 4 | 1 | 1999-09-01 00:00:00.000 | стереосистема | 12 | 220.45 | ПП Іваненко І.І |
| 5 | 1 | 1999-09-01 00:00:00.000 | телевізор | 10 | 1253.45 | ПП Іваненко І.І |
| 6 | 2 | 1999-09-10 00:00:00.000 | відеомагнітофон | 8 | 450.67 | ПП Іваненко І.І |
| 7 | 2 | 1999-09-10 00:00:00.000 | комп'ютер | 43 | 1453.18 | ПП Іваненко І.І |
| 8 | 2 | 1999-09-10 00:00:00.000 | магнітофон | 5 | 455.14 | ПП Іваненко І.І |
| 9 | 2 | 1999-09-10 00:00:00.000 | стереосистема | 11 | 511.43 | ПП Іваненко І.І |
| 10 | 5 | 1999-09-24 00:00:00.000 | відеомагнітофон | 17 | 850.12 | ТОВ «Інтерфрут» |
| 11 | 5 | 1999-09-24 00:00:00.000 | магнітофон | 33 | 585.67 | ТОВ «Інтерфрут» |
| 12 | 5 | 1999-09-24 00:00:00.000 | монітор | 44 | 590.23 | ТОВ «Інтерфрут» |
| 13 | 5 | 1999-09-24 00:00:00.000 | телевізор | 14 | 860.33 | ТОВ «Інтерфрут» |
| 14 | 6 | 1999-10-01 00:00:00.000 | комп'ютер | 32 | 1850.24 | ПП Іваненко І.І |
| 15 | 6 | 1999-10-01 00:00:00.000 | монітор | 51 | 520.95 | ПП Іваненко І.І |
| 16 | 6 | 1999-10-01 00:00:00.000 | телевізор | 34 | 810.15 | ПП Іваненко І.І |
| 17 | 7 | 1999-10-02 00:00:00.000 | комп'ютер | 15 | 1234.56 | ТОВ «Інтерфрут» |
| 18 | 7 | 1999-10-02 00:00:00.000 | монітор | 22 | 389.75 | ТОВ «Інтерфрут» |
| 19 | 7 | 1999-10-02 00:00:00.000 | телевізор | 62 | 900.58 | ТОВ «Інтерфрут» |

Рисунок 4.35

Запит 14.

Мета запиту: сформувати номенклатуру товарів (тобто список найменувань товарів), які поставлялися лише постачальником 1 (ПП Іваненко І.І), або лише постачальником 2 (ТОВ «Інтерфрут»), або і постачальником 1, і постачальником 2.

Текст запиту та результат виконання запиту наведені на рисунках 4.36 та 4.37 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery14.sql.

```
USE delivery

SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=1
UNION
SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=2
```

Рисунок 4.36

| | Product |
|---|---|
| 1 | відеомагнітофон |
| 2 | комп'ютер |
| 3 | магнітофон |
| 4 | монітор |
| 5 | стереосистема |
| 6 | телевізор |

Рисунок 4.37

Запит 15.

Мета запиту: сформувати номенклатуру товарів (тобто список найменувань товарів), які постачалися і постачальником 1 (ПП Іваненко І.І), і постачальником 2 (ТОВ «Інтерфрут»).

Примітка. Цей запит ілюструє особливості використання операції перетину (INTERSECT).

Текст запиту та результат виконання запиту наведені на рисунках 4.38 та 4.39 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery15.sql.

```
USE delivery

SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=1
INTERSECT
SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=2
```

Рисунок 4.38

| | Product |
|---|---|
| 1 | відеомагнітофон |
| 2 | комп'ютер |
| 3 | магнітофон |
| 4 | монітор |
| 5 | телевізор |

Рисунок 4.39

Запит 16.

Мета запиту: сформувати номенклатуру товарів (тобто список найменувань товарів), які постачалися постачальником 1 (ПП Іваненко І.І), але не постачалися постачальником 2 (ТОВ «Інтерфрут»).

Примітка. Цей запит ілюструє особливості використання операції різниці (EXCEPT).

Текст запиту та результат виконання запиту наведені на рисунках 4.40 та 4.41 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery16.sql.

```
USE delivery

SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=1
EXCEPT
SELECT DISTINCT Supplied.Product
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE Contracts.SupplierID=2
```

Рисунок 4.40

| | Product |
|---|---|
| 1 | стереосистема |

Рисунок 4.41

Запит 17.

Мета запиту: сформувати список товарів, який має відображати частоту постачання товарів за увесь період постачання. До списку включити лише товари, які постачалися більше одного разу. Список повинен бути відсортований у порядку зменшення частоти поставок.

Текст запиту та результат виконання запиту наведені на рисунках 4.42 та 4.43 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery17.sql.

```
USE delivery

SELECT Product, COUNT(Product) AS КількістьПостачань
FROM Supplied
GROUP BY Product
HAVING COUNT(Product)>1
ORDER BY COUNT(Product) DESC
```

Рисунок 4.42

| | Product | КількістьПостачань |
|---|---|---|
| 1 | телевізор | 6 |
| 2 | магнітофон | 5 |
| 3 | монітор | 4 |
| 4 | комп'ютер | 4 |
| 5 | відеомагнітофон | 3 |
| 6 | стереосистема | 3 |

Рисунок 4.43

Запит 18.

Мета запиту: сформувати дані про кількісну динаміку постачання товарів протягом 1999 року. Дані мають бути агреговані по-місячно та подані у вигляді таблиці, рядками якої є назви товарів, а стовпцями – номери місяців 1999 року. На перетині рядка та стовпця має відображатися кількість товару, поставленого в цьому місяці.

Примітка. Цей запит ілюструє особливості створення та використання перехресного запиту засобами Transact-SQL.

Текст запиту та результат виконання запиту наведені на рисунках 4.44 та 4.35 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery18_1.sql.

```
USE delivery

SELECT Product, [1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12]
FROM
(
SELECT Product,MONTH(ContractDate) AS mis, Amount
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE YEAR(ContractDate)=1999
) p
PIVOT
(SUM(Amount) FOR mis IN ([1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12])) AS pvt
ORDER BY Product
```

Рисунок 4.34

| | Product | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | відеомагнітофон | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 37 | NULL | NULL | NULL |
| 2 | комп'ютер | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 67 | 47 | NULL | NULL |
| 3 | магнітофон | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 96 | NULL | NULL | NULL |
| 4 | монітор | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 129 | 73 | NULL | NULL |
| 5 | принтер | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 41 | NULL | NULL | NULL |
| 6 | стереосистема | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 50 | NULL | NULL | NULL |
| 7 | телевізор | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 132 | 96 | NULL | NULL |

Рисунок 4.35

Наведений на рисунку 4.35 результат запиту може бути незручним для сприйняття (наприклад, через наявність значень NULL). Цей недолік можна усунути, наприклад, шляхом заміни значень NULL на 0. Текст зміненого запиту та результат виконання запиту наведені на рисунках 4.36 та 4.37 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery18_2.sql

```
USE delivery

SELECT Product, ISNULL([1],0) AS [1], ISNULL([2],0) AS [2], ISNULL([3],0) AS [3], ISNULL([4],0) AS [4],
               ISNULL([5],0) AS [5], ISNULL([6],0) AS [6], ISNULL([7],0) AS [7], ISNULL([8],0) AS [8],
               ISNULL([9],0) AS [9], ISNULL([10],0) AS [10], ISNULL([11],0) AS [11], ISNULL([12],0) AS [12]
FROM
(
SELECT Product,MONTH(ContractDate) AS mis, Amount
FROM Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE YEAR(ContractDate)=1999
) p
PIVOT
(SUM(Amount) FOR mis IN ([1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12])) AS pvt
ORDER BY Product
```

Рисунок 4.36

| | Product | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | відеомагнітофон | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 37 | 0 | 0 | 0 |
| 2 | комп'ютер | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 67 | 47 | 0 | 0 |
| 3 | магнітофон | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 96 | 0 | 0 | 0 |
| 4 | монітор | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 129 | 73 | 0 | 0 |
| 5 | принтер | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 41 | 0 | 0 | 0 |
| 6 | стереосистема | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 0 | 0 | 0 |
| 7 | телевізор | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 132 | 96 | 0 | 0 |

Рисунок 4.37

Запит 19.

Мета запиту: сформувати перелік поставлених товарів, постачальниками яких були фізичні особи. Для кожного товару у цьому списку мають бути зазначені такі дані: номер договору, назва товару, кількість одиниць, ціна за одиницю, дата поставки, назва місяця поставки та номер року поставки.

Текст запиту та результат виконання запиту наведені на рисунках 4.38 та 4.39 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery19_1.sql.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Amount, PricePerItem, Contracts.ContractDate,
       DATENAME(month,ContractDate) AS Місяць, YEAR(ContractDate) AS Рік
FROM  (Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber)
       INNER JOIN IndividualEntrepreneurs ON Contracts.SupplierID=IndividualEntrepreneurs.SupplierID
```

Рисунок 4.38

| | ContractNumber | Product | Amount | PricePerItem | ContractDate | Місяць | Рік |
|---|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 12 | 722.33 | 1999-09-01 00:00:00.000 | September | 1999 |
| 2 | 1 | комп'ютер | 24 | 1554.22 | 1999-09-01 00:00:00.000 | September | 1999 |
| 3 | 1 | магнітофон | 25 | 655.12 | 1999-09-01 00:00:00.000 | September | 1999 |
| 4 | 1 | стереосистема | 12 | 220.45 | 1999-09-01 00:00:00.000 | September | 1999 |
| 5 | 1 | телевізор | 10 | 1253.45 | 1999-09-01 00:00:00.000 | September | 1999 |
| 6 | 2 | відеомагнітофон | 8 | 450.67 | 1999-09-10 00:00:00.000 | September | 1999 |
| 7 | 2 | комп'ютер | 43 | 1453.18 | 1999-09-10 00:00:00.000 | September | 1999 |
| 8 | 2 | магнітофон | 5 | 455.14 | 1999-09-10 00:00:00.000 | September | 1999 |
| 9 | 2 | стереосистема | 11 | 511.43 | 1999-09-10 00:00:00.000 | September | 1999 |
| 10 | 3 | магнітофон | 11 | 544.00 | 1999-09-10 00:00:00.000 | September | 1999 |
| 11 | 3 | монітор | 85 | 545.32 | 1999-09-10 00:00:00.000 | September | 1999 |
| 12 | 3 | телевізор | 52 | 899.99 | 1999-09-10 00:00:00.000 | September | 1999 |
| 13 | 4 | магнітофон | 22 | 323.19 | 1999-09-23 00:00:00.000 | September | 1999 |
| 14 | 4 | принтер | 41 | 350.77 | 1999-09-23 00:00:00.000 | September | 1999 |
| 15 | 4 | стереосистема | 27 | 330.55 | 1999-09-23 00:00:00.000 | September | 1999 |
| 16 | 4 | телевізор | 56 | 990.56 | 1999-09-23 00:00:00.000 | September | 1999 |
| 17 | 6 | комп'ютер | 32 | 1850.24 | 1999-10-01 00:00:00.000 | October | 1999 |
| 18 | 6 | монітор | 51 | 520.95 | 1999-10-01 00:00:00.000 | October | 1999 |
| 19 | 6 | телевізор | 34 | 810.15 | 1999-10-01 00:00:00.000 | October | 1999 |

Рисунок 4.39

Як видно з результату запиту, що наведений на рисунку 4.39, формально вимогу включення до результату запиту найменування місяця виконано шляхом використання вбудованої функції DATENAME(). Однак такі назви місяців не завжди можуть бути зручними для сприйняття, наприклад, з точки зору кінцевого користувача. Може виникнути вимога їхньої заміни на назви, які використовують у певній національній мові, наприклад, в українській. Цю проблему можна вирішити шляхом розробки функції користувача, що конвертує назви місяців. Цей підхід можна вважати досить трудомістким. Іншим варіантом вирішення проблеми може бути використання у запиті функції CASE мови Transact-SQL. Текст такого запиту та результат виконання запиту наведені на рисунках 4.40 та 4.41 відповідно. Запит можна зберегти у файлі з ім'ям SQLQuery19_1.sql.

```
USE delivery

SELECT Supplied.ContractNumber, Supplied.Product, Amount, PricePerItem, Contracts.ContractDate,
       Місяць = CASE MONTH(ContractDate)
                WHEN 1 THEN 'січень'
                WHEN 2 THEN 'лютий'
                WHEN 3 THEN 'березень'
                WHEN 4 THEN 'квітень'
                WHEN 5 THEN 'травень'
                WHEN 6 THEN 'червень'
                WHEN 7 THEN 'липень'
                WHEN 8 THEN 'серпень'
                WHEN 9 THEN 'вересень'
                WHEN 10 THEN 'жовтень'
                WHEN 11 THEN 'листопад'
                WHEN 12 THEN 'грудень'
                ELSE '???????'
                END,
            YEAR(ContractDate) AS Рік
FROM  (Contracts INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber)
       INNER JOIN IndividualEntrepreneurs ON Contracts.SupplierID=IndividualEntrepreneurs.SupplierID
```

Рисунок 4.40

| | ContractNumber | Product | Amount | PricePerItem | ContractDate | Місяць | Рік |
|---|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 12 | 722.33 | 1999-09-01 00:00:00.000 | вересень | 1999 |
| 2 | 1 | комп'ютер | 24 | 1554.22 | 1999-09-01 00:00:00.000 | вересень | 1999 |
| 3 | 1 | магнітофон | 25 | 655.12 | 1999-09-01 00:00:00.000 | вересень | 1999 |
| 4 | 1 | стереосистема | 12 | 220.45 | 1999-09-01 00:00:00.000 | вересень | 1999 |
| 5 | 1 | телевізор | 10 | 1253.45 | 1999-09-01 00:00:00.000 | вересень | 1999 |
| 6 | 2 | відеомагнітофон | 8 | 450.67 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 7 | 2 | комп'ютер | 43 | 1453.18 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 8 | 2 | магнітофон | 5 | 455.14 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 9 | 2 | стереосистема | 11 | 511.43 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 10 | 3 | магнітофон | 11 | 544.00 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 11 | 3 | монітор | 85 | 545.32 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 12 | 3 | телевізор | 52 | 899.99 | 1999-09-10 00:00:00.000 | вересень | 1999 |
| 13 | 4 | магнітофон | 22 | 323.19 | 1999-09-23 00:00:00.000 | вересень | 1999 |
| 14 | 4 | принтер | 41 | 350.77 | 1999-09-23 00:00:00.000 | вересень | 1999 |
| 15 | 4 | стереосистема | 27 | 330.55 | 1999-09-23 00:00:00.000 | вересень | 1999 |
| 16 | 4 | телевізор | 56 | 990.56 | 1999-09-23 00:00:00.000 | вересень | 1999 |
| 17 | 6 | комп'ютер | 32 | 1850.24 | 1999-10-01 00:00:00.000 | жовтень | 1999 |
| 18 | 6 | монітор | 51 | 520.95 | 1999-10-01 00:00:00.000 | жовтень | 1999 |
| 19 | 6 | телевізор | 34 | 810.15 | 1999-10-01 00:00:00.000 | жовтень | 1999 |

Рисунок 4.41

Розглянуті вище приклади SELECT-запитів, звичайно, не відображають усіх особливостей та можливостей оператора SELECT як

загалом, так і у мові Transact-SQL. Отже, для отримання більш глибоких навичок щодо використання мови SQL, потрібна самостійна робота.

### 4.2.3 Звітність про виконання практичних завдань теми 4

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) для кожного з реалізованих запитів навести умову запиту, текст запиту та результат виконання запиту;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 4.3 Питання для самоперевірки по темі 4

1. Команда SELECT-SQL. Загальна характеристика, призначення та використання.

2. Команда SELECT-SQL. Реляційні операції, які реалізуються за допомогою команди SELECT-SQL.

3. Команда SELECT-SQL. Стовпці виведення результату запиту. Використання виразів в стовпцях виведення.

4. Команда SELECT-SQL. Пропозиція FROM. Призначення і використання.

5. Команда SELECT-SQL. Пропозиція WHERE. Призначення і використання.

6. Команда SELECT-SQL. Пропозиція WHERE. Умови фільтрації і їх використання.

7. Команда SELECT-SQL. Пропозиція WHERE. Формування запиту на базі декількох таблиць. JOIN-умови і їх використання.

8. Команда SELECT-SQL. З'єднання таблиць в багатотабличних запитах. Види з'єднань – INNER JOIN, LEFT JOIN, RIGHT JOIN і їх особливості.

9. Команда SELECT-SQL. Перехресні запити і їх особливості.

10. Команда SELECT-SQL. Аргумент DISTINCT. Призначення і використання.

11. Команда SELECT-SQL. Аргумент TOP. Призначення і використання.

12. Команда SELECT-SQL. Булеві оператори AND, OR, NOT та їх використання.

13. Команда SELECT-SQL. Спеціальні оператори IN, BETWEEN. Призначення і використання.

14. Команда SELECT-SQL. Агрегуючі функції. Призначення і використання.

15. Команда SELECT-SQL. Пропозиція GROUP BY. Призначення і використання.

16. Команда SELECT-SQL. Пропозиція ORDER BY. Призначення і використання.

17. Команда SELECT-SQL. Пропозиція HAVING. Призначення і використання.

18. Команда SELECT-SQL. Підзапити. Призначення і використання.

19. Команда SELECT-SQL. Підзапити. Види підзапитів і їх особливості.

20. Якими засобами можна працювати із результатами виконання підзапитів?

21. Команда SELECT-SQL. Операції об'єднання, перетину, різниці. Призначення і використання.

22. Команда SELECT-SQL. Функція CAST() та її призначення. Чим обумовлено застосування цієї функції у тих прикладах, що розглядалися?

23. Як реалізувати запит 15 без використання операції UNION?

24. Команда SELECT-SQL. Функція ISNULL() та її призначення.

# ТЕМА 5
# СТВОРЕННЯ ТА ВИКОРИСТАННЯ ПРОГРАМНИХ ОБ'ЄКТІВ БАЗИ ДАНИХ ЗАСОБАМИ СУБД MICROSOFT SQL SERVER

## 5.1 Теоретичні відомості

Функції в мовах програмування – це конструкції, що містять часто виконуваний код. Функція виконує які-небудь дії над даними і повертає деяке значення.

Збережені процедури є групою команд SQL, об'єднаних в один модуль. Така група команд компілюється і виконується як єдине ціле.

Тригерами називається спеціальний клас збережених процедур, що автоматично запускаються при додаванні, зміні або видаленні даних з таблиці.

Курсор в SQL – це область в пам'яті бази даних, яка призначена для зберігання останнього оператора SQL. Якщо поточний оператор – запит до бази даних, в пам'яті зберігається і рядок даних запиту, званий поточним значенням, або поточним рядком курсору. Вказана область в пам'яті пойменована і доступна для прикладних програм.

### 5.1.1 Створювані користувачем функції та вбудовані функції

При реалізації за допомогою мови SQL складних алгоритмів, які можуть знадобитися більше одного разу, відразу постає питання про збереження розробленого коду для подальшого застосування. Цю задачу можна було б реалізувати за допомогою збережених процедур, однак їх архітектура не дозволяє використовувати процедури безпосередньо у виразах, тому вони вимагають проміжного присвоєння повернутого значення змінної, яка потім і вказується в вираженні. Природно, подібний метод застосування програмного коду не дуже зручний. Багато розробників вже давно хотіли мати можливість виклику розроблених алгоритмів безпосередньо в виразах.

Функції користувача являють собою самостійні об'єкти бази даних, такі, наприклад, як збережені процедури або тригери. Функція користувача розташовується в певній базі даних і доступна тільки в її контексті.

У Microsoft SQL Server є наступні класи функцій користувача:

- Scalar (scalar-valued user-defined function) – скалярні функції повертають звичайне скалярне значення; кожна функція може включати безліч команд, що об'єднуються в один блок за допомогою конструкції BEGIN ... END;
- Inline (inline table-valued function) – табличні функції містять всього одну команду SELECT і повертають користувачеві набір даних у вигляді значення типу даних TABLE;
- Multi-statement (multi-statement table-valued function) – багатооператорні табличні функції також повертають користувачеві значення типу даних TABLE, що містить набір даних, проте в тілі функції знаходиться безліч команд SQL (SELECT, INSERT, UPDATE і т.д.). Саме з їх допомогою і формується набір даних, який має бути повернуто після виконання функції.

Користувальницькі функції схожі з збереженими процедурами, але, на відміну від них, можуть застосовуватися в запитах так само, як і системні вбудовані функції. Користувальницькі функції, що повертають таблиці, можуть стати альтернативою представленням. Представлення обмежені одним виразом SELECT, а користувальницькі функції здатні включати додаткові вирази, що дозволяє створювати більш складні і потужні конструкції.

Створення та зміна функції скалярного типу (scalar-valued user-defined function) виконується за допомогою команди:

```
<Визначення_скаляр_функції> :: =
{CREATE | ALTER} FUNCTION [власник.]
   імя_функції
([{@ імя_параметра скаляр_тип_даних
   [= default]} [, ... n]])
RETURNS скаляр_тип_даних
[WITH {ENCRYPTION | SCHEMABINDING}
   [, .. . n]]
[AS]
BEGIN
<тіло_функції>
RETURN скаляр_вираз
END
```

Розглянемо призначення параметрів команди.

Функція може містити один або декілька вхідних параметрів або не містити жодного. Кожен параметр повинен мати унікальне в межах створюваної функції ім'я і починатися з символу "@". Після імені вказується тип даних параметра. Додатково можна вказати значення, яке буде автоматично присвоюватися параметру (DEFAULT), якщо користувач явно не вказав значення відповідного параметра при виклику функції .

За допомогою конструкції RETURNS скаляр_тип_даних вказується, який тип даних буде мати значення, яке повертається функцією.

Додаткові параметри, з якими повинна бути створена функція, можуть бути зазначені за допомогою ключового слова WITH. Завдяки ключовим словом ENCRYPTION код команди, використовуваний для створення функції, буде зашифровано, і ніхто не зможе переглянути його. Ця можливість дозволяє приховати логіку роботи функції. Крім того, в тілі функції може виконуватися звернення до різних об'єктів бази даних, а тому зміна або видалення відповідних об'єктів може призвести до порушення роботи функції. Щоб уникнути цього, потрібно заборонити

внесення змін, вказавши при створенні цієї функції ключове слово SCHEMABINDING.

Між ключовими словами BEGIN ... END вказується набір команд, вони й будуть тілом функції.

Коли в ході виконання коду функції зустрічається ключове слово RETURN, виконання функції завершується і як результат її обчислення повертається значення, вказане безпосередньо після слова RETURN. Відзначимо, що в тілі функції дозволяється використання безлічі команд RETURN, які можуть повертати різні значення. В якості значення, що повертається допускаються як звичайні константи, так і складні вирази. Єдина умова – тип даних значення, що повертається, повинен збігатися з типом даних, зазначених після ключового слова RETURNS.

Створення та зміна табличної функції (inline table-valued function) виконується за допомогою команди:

```
<Визначен_табл_функції> :: =
{CREATE | ALTER} FUNCTION [власник.]
   імя_функції
([{@ імя_параметра скаляр_тип_даних
   [= default]} [, ... n]])
RETURNS TABLE
[WITH {ENCRYPTION | SCHEMABINDING}
   [, .. . n]]
[AS]
RETURN [(] SELECT_оператор [)]
```

Основна частина параметрів, використовуваних при створенні табличних функцій, аналогічна параметрам скалярної функції. Проте створення табличних функцій має свою специфіку.

Після ключового слова RETURNS завжди повинно вказуватися ключове слово TABLE. Таким чином, функція даного типу повинна строго

повертати значення типу даних TABLE. Структура значення, що повертається типом TABLE, не вказується явно при описі власне типу даних. Замість цього сервер буде автоматично використовувати для значення, що повертається TABLE, структуру, що повертається запитом SELECT, який є єдиною командою функції.

Особливість функції даного типу полягає в тому, що структура значення TABLE створюється автоматично в ході виконання запиту, а не вказується явно при визначенні типу після ключового слова RETURNS.

Значення типу TABLE, що повертається функцією, може бути використано безпосередньо у запиті, тобто в розділі FROM.

Створення та зміна функцій типу Multi-statement виконується за допомогою наступної команди:

```
<Визначення_мульти_функції> :: =
{CREATE | ALTER} FUNCTION [власник.]
   імя_функції
([{@ імя_параметра скаляр_тип_даних
   [= default]} [, ... n]])
RETURNS @ імя_параметра TABLE
   <визначення_таблиці>
[WITH {ENCRYPTION | SCHEMABINDING }
   [, ... n]]
[AS]
BEGIN
<тіло_функції>
RETURN
END
```

Використання більшої частини параметрів розглядалося при описі попередніх функцій.

Відзначимо, що функції даного типу, як і табличні, повертають значення типу TABLE. Однак, на відміну від табличних функцій, при створенні функцій Multi-statement необхідно явно задати структуру значення, що повертається. Вона зазначається безпосередньо після ключового слова TABLE і, таким чином, є частиною визначення типу даних, що повертається. Синтаксис конструкції <визначен_таблиці> повністю відповідає однойменним структурам, використовуваним при створенні звичайних таблиць за допомогою команди CREATE TABLE.

Набір даних, що повертається, повинен формуватися за допомогою команд INSERT, виконуваних в тілі функції. Крім того, в тілі функції допускається використання різних конструкцій мови SQL, які можуть контролювати значення, що розміщуються у вихідному наборі рядків. При роботі з командою INSERT потрібно явно вказати ім'я того об'єкта, куди необхідно вставити рядки. Тому в функціях типу Multi-statement, на відміну від табличних, необхідно присвоїти якесь ім'я об'єкту з типом даних TABLE – воно і вказується як значення, що повертається.

Завершення роботи функції відбувається у двох випадках: якщо виникають помилки виконання і якщо з'являється ключове слово RETURN. На відміну від функцій скалярного типу, при використанні команди RETURN не потрібно вказувати значення, що повертається. Сервер автоматично поверне набір даних типу TABLE, ім'я та структура якого була вказана після ключового слова RETURNS. У тілі функції може бути вказано більше однієї команди RETURN.

Необхідно відзначити, що робота функції завершується тільки при наявності команди RETURN. Це твердження вірне і в тому випадку, коли мова йде про досягнення кінця тіла функції – самою останньою командою повинна бути команда RETURN.

Вбудовані функції, що наявні в розпорядженні користувачів при роботі з SQL, можна умовно розділити на наступні групи:

- математичні функції;
- рядкові функції;

- функції для роботи з датою і часом;
- функції конфігурування;
- функції системи безпеки;
- функції управління метаданими;
- статистичні функції.

Стислий перелік математичних функцій представлений в таблиці 5.1.

Таблиця 5.1 – Стислий перелік математичних функцій

| Функція | Призначення функції |
|---|---|
| ABS | обчислює абсолютне значення числа |
| ACOS | обчислює арккосинус |
| ASIN | обчислює арксинус |
| ATAN | обчислює арктангенс |
| ATN2 | обчислює арктангенс з урахуванням квадратів |
| CEILING | виконує округлення вгору |
| COS | обчислює косинус кута |
| COT | повертає котангенс кута |
| DEGREES | перетворить значення кута з радіан в градуси |
| EXP | повертає експоненту |
| FLOOR | виконує округлення вниз |
| LOG | обчислює натуральний логарифм |
| LOG10 | обчислює десятковий логарифм |
| PI | повертає значення "пі" |
| POWER | зводить число до ступеня |
| RADIANS | перетворить значення кута з градуса в радіани |
| RAND | повертає випадкове число |
| ROUND | виконує округлення із заданою точністю |
| SIGN | визначає знак числа |
| SIN | обчислює синус кута |
| SQUARE | виконує зведення числа в квадрат |
| SQRT | витягує квадратний корінь |
| TAN | повертає тангенс кута |

Стислий перелік рядкових функцій представлений в таблиці 5.2.

Таблиця 5.2 – Стислий перелік рядкових функцій

| Функція | Призначення функції |
|---|---|
| ASCII | повертає код ASCII лівого символу рядка |
| CHAR | за кодом ASCII повертає символ |
| CHARINDEX | визначає порядковий номер символу, з якого починається входження підрядка в рядок |
| DIFFERENCE | повертає показник збігу рядків |
| LEFT | повертає вказану кількість символів з початку рядка |
| LEN | повертає довжину рядка |
| LOWER | переводить всі символи рядка в нижній регістр |
| LTRIM | видаляє пробіли на початку рядка |
| NCHAR | повертає за кодом символ Unicode |
| PATINDEX | виконує пошук підрядка в рядку за вказаною шаблоном |
| REPLACE | замінює входження підрядка на вказане значення |
| QUOTENAME | конвертує рядок в формат Unicode |
| REPLICATE | виконує тиражування рядки певне число разів |
| REVERSE | повертає рядок, символи якої записані у зворотному порядку |
| RIGHT | повертає вказану кількість символів з кінця рядка |
| RTRIM | видаляє пробіли в кінці рядка |
| SOUNDEX | повертає код звучання рядка |
| SPACE | повертає вказану кількість пробілів |
| STR | виконує конвертування значення числового типу в символьний формат |
| STUFF | видаляє вказане число символів, замінюючи нової підрядком |
| SUBSTRING | повертає для рядка підрядок зазначеної довжини з заданого символу |
| UNICODE | повертає Unicode-код лівого символу рядка |
| UPPER | переводить всі символи рядка в верхній регістр |

Стислий перелік основних функцій для роботи з датою і часом представлений в таблиці 5.3.

Таблиця 5.3 – Стислий перелік основних функцій для роботи з датою і часом

| Функція | Призначення функції |
|---|---|
| DATEADD | додає до дати зазначена значення днів, місяців, годин і т.д. |
| DATEDIFF | повертає різницю між зазначеними частинами двох дат |
| DATENAME | виділяє з дати вказану частину і повертає її в символьному форматі |
| DATEPART | виділяє з дати вказану частину і повертає її в числовому форматі |
| DAY | повертає число з вказаної дати |
| GETDATE | повертає поточний системний час |
| ISDATE | перевіряє правильність виразу на відповідність одному з можливих форматів вводу дати |
| MONTH | повертає значення місяця з вказаної дати |
| YEAR | повертає значення року з вказаної дати |

З іншими типами вбудованих функцій у разі потреби можна ознайомитися, наприклад, на веб-ресурсі:

https://learn.microsoft.com

### 5.1.2 Збережені процедури

Збережені процедури є групами пов'язаних між собою операторів SQL, застосування яких робить роботу програміста легшою і гнучкішою, оскільки виконати процедуру, що зберігається, часто виявляється набагато простіше, ніж послідовність окремих операторів SQL. Збережені процедури є набором команд, що складається з одного або декількох операторів SQL або функцій і зберігається у базі даних у вигляді, що відкомпілювався. Виконання у базі даних процедур замість окремих операторів SQL, що зберігаються, дає користувачеві наступні переваги:

- необхідні оператори вже містяться у базі даних;
- усі вони пройшли етап синтаксичного аналізу і знаходяться у виконуваному форматі; перед виконанням збереженої процедури SQL Server генерує для неї план виконання, виконує її оптимізацію і компіляцію;

- збережені процедури підтримують модульне програмування, оскільки дозволяють розбивати великі завдання на самостійні, дрібніші і зручніші в управлінні частини;
- збережені процедури можуть викликати інші збережені процедури і функції;
- збережені процедури можуть бути викликані з прикладних програм інших типів;
- як правило, збережені процедури виконуються швидше, ніж послідовність окремих операторів;
- збережені процедури простіше використовувати: вони можуть складатися з десятків і сотень команд, але для їх запуску досить вказати усього лише ім'я потрібної збереженої процедури. Це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер, отже, і навантаження на мережу.

Зберігання процедур в тому ж місці, де вони виконуються, забезпечує зменшення об'єму даних, що передаються по мережі, і підвищує загальну продуктивність системи. Застосування процедур, що зберігаються, спрощує супровід програмних комплексів і внесення змін в них. Зазвичай усі обмеження цілісності у вигляді правил і алгоритмів обробки даних реалізуються на сервері баз даних і доступні кінцевому застосуванню у вигляді набору процедур, що зберігаються, які і представляють інтерфейс обробки даних. Для забезпечення цілісності даних, а також в цілях безпеки, застосунок зазвичай не дістає прямого доступу до даних – уся робота з ними ведеться шляхом виклику тих або інших процедур, що зберігаються.

Подібний підхід спрощує модифікацію алгоритмів обробки даних, які негайно стають доступними для усіх користувачів мережі, і забезпечує можливість розширення системи без внесення змін безпосередньо до застосунків – досить змінити процедуру, що зберігається, на сервері баз даних. Розробникові не треба перекомпілювати застосунок, створювати його копії, а також інструктувати користувачів про необхідність роботи з

новою версією. Користувачі взагалі можуть не підозрювати про те, що в систему внесені зміни.

Збережені процедури, існують незалежно від таблиць або яких-небудь інших об'єктів баз даних. Вони викликаються клієнтською програмою, іншою процедурою, що зберігається, або тригером. Розробник може управляти правами доступу до збереженої процедури, дозволяючи або забороняючи її виконання. Змінювати код збереженої процедури, дозволяється тільки її власникові або членові фіксованої ролі бази даних. При необхідності можна передати права володіння нею від одного користувача до іншого.

При роботі з SQL Server користувачі можуть створювати власні процедури, що реалізовують ті або інші дії. Збережені процедури є повноцінними об'єктами бази даних, а тому кожна з них зберігається в конкретній базі даних. Безпосередній виклик збереженої процедури можливий тільки якщо він здійснюється в контексті тієї бази даних, де знаходиться процедура.

У SQL Server є декілька типів процедур, що зберігаються.

- Системні збережені процедури, призначені для виконання різних адміністративних дій. Практично усі дії з адміністрування сервера виконуються з їх допомогою. Можна сказати, що системні Збережені процедури, є інтерфейсом, що забезпечує роботу з системними таблицями, яка, кінець кінцем, зводиться до зміни, додавання, видалення і вибірки даних з системних таблиць як призначених для користувача, так і системних баз даних. Системні збережені процедури мають префікс sp_, зберігаються в системній базі даних і можуть бути викликані в контексті будь-якої іншої бази даних.
- Призначені для користувача збережені процедури, реалізують ті або інші дії. Такі збережені процедури – повноцінний об'єкт бази даних. Внаслідок цього кожна процедура, що зберігається, розташовується в конкретній базі даних, де і виконується.

• Тимчасові збережені процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні і глобальні. Локальні тимчасові збережені процедури, можуть бути викликані тільки з того з'єднання, в якому створені. При створенні такої процедури їй необхідно дати ім'я, що починається з одного символу #. Як і усі тимчасові об'єкти, процедури цього типу, що зберігаються, автоматично віддаляються при відключенні користувача, перезапуску або зупинці сервера. Глобальні тимчасові збережені процедури, доступні для будь-яких з'єднань сервера, на якому є така ж процедура. Для її визначення досить дати їй ім'я, що починається з символів ##. Видаляються ці процедури при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті якого вони були створені.

Створення збереженої процедури, передбачає вирішення наступних завдань:

• визначення типу створюваної збереженої процедури: тимчасова або призначена для користувача. Окрім цього, можна створити свою власну системну процедуру, що зберігається, призначивши їй ім'я з префіксом sp_ і помістивши її в системну базу даних. Така процедура буде доступна в контексті будь-якої бази даних локального сервера;

• планування прав доступу. При створенні збереженої процедури, слід враховувати, що вона матиме ті ж права доступу до об'єктів бази даних, що і користувач, що створив її;

• визначення параметрів збереженої процедури. Подібно до процедур, що входять до складу більшості мов програмування, Збережені процедури, можуть бути наділені вхідними і вихідними параметрами ;

• розробка коду збереженої процедури. Код процедури може містити послідовність будь-яких команд SQL, включаючи виклик інших процедур, що зберігаються.

Створення нової і зміна наявної збереженої процедури, здійснюється за допомогою наступної команди:

```
<визначення_процедури>::=
{CREATE | ALTER } [PROCEDURE] ім'я_процедури
      [;номер]
[{@ім'я_параметра тип_даних } [VARYING ]
            [=default][OUTPUT] ][,..n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE
            ENCRYPTION }]
[FOR REPLICATION]
AS
            sql_оператор [..n]
```

Розглянемо параметри цієї команди.

Використовуючи префікси sp_, #, ##, створювану процедуру можна визначити в якості системної або тимчасової. Як видно з синтаксису команди, не допускається вказувати ім'я власника, якому належатиме створювана процедура, а також ім'я бази даних, де вона має бути розміщена. Так, щоб розмістити утворюючу збережену процедуру в конкретній базі даних, необхідно виконати команду CREATE PROCEDURE в контексті цієї бази даних. При зверненні з тіла збереженої процедури, до об'єктів тієї ж бази даних можна використовувати укорочені імена, т. е. без вказівки імені бази даних. Коли ж вимагається звернутися до об'єктів, розташованих в інших базах даних, вказівка імені бази даних обов'язково.

Номер в імені – це ідентифікаційний номер збереженої процедури, що однозначно визначає її в групі процедур. Для зручності управління процедурами логічно однотипні збережені процедури можна групувати, привласнюючи їм однакові імена, але різні ідентифікаційні номери.

Для передачі вхідних і вихідних даних в створюваній процедурі, що зберігається, можуть використовуватися параметри, імена яких, як і імена локальних змінних, повинні починатися з символу @. В одній процедурі, що зберігається, можна задати безліч параметрів, розділених комами. У

тілі процедури не повинні застосовуватися локальні змінні, чиї імена співпадають з іменами параметрів цієї процедури.

Для визначення типу даних, який матиме відповідний параметр збереженої процедури, годяться будь-які типи даних SQL, включаючи визначені користувачем. Проте тип даних CURSOR може бути використаний тільки як вихідний параметр збереженої процедури, тобто з вказівкою ключового слова OUTPUT.

Наявність ключового слова OUTPUT означає, що відповідний параметр призначений для повернення даних з збереженої процедури. Проте це зовсім не означає, що параметр не підходить для передачі значень в процедуру, що зберігається. Вказівка ключового слова OUTPUT наказує серверу при виході з збереженої процедури, присвоїти поточне значення параметра локальної змінної, яка була вказана при виклику процедури в якості значення параметра. Відмітимо, що при вказівці ключового слова OUTPUT значення відповідного параметра при виклику процедури може бути задане тільки за допомогою локальної змінної. Не дозволяється використання будь-яких виразів або констант, допустиме для звичайних параметрів.

Ключове слово VARYING застосовується спільно з параметром OUTPUT, що має тип CURSOR. Воно визначає, що вихідним параметром буде результуюча множина.

Ключове слово DEFAULT є значенням, яке прийматиме відповідний параметр за умовчанням. Таким чином, при виклику процедури можна не вказувати явно значення відповідного параметра.

Оскільки сервер кешує план виконання запиту і компілює код, при наступному виклику процедури використовуватимуться вже готові значення. Проте в деяких випадках все ж вимагається виконувати перекомпіляцію коду процедури. Вказівка ключового слова RECOMPILE наказує системі створювати план виконання збереженої процедури, при кожному її виклику.

Параметр FOR REPLICATION затребуваний при реплікації даних і включенні створюваної збереженої процедури, в якості статті в публікацію.

Ключове слово ENCRYPTION наказує серверу виконати шифрування коду збереженої процедури, що може забезпечити захист від використання авторських алгоритмів, що реалізовують роботу збереженої процедури.

Ключове слово AS розміщується на початку власне тіла збереженої процедури, тобто набору команд SQL, за допомогою яких і реалізовуватиметься те або інша дія. У тілі процедури можуть застосовуватися практично усі команди SQL, оголошуватися транзакції, встановлюватися блокування і викликатися інші Збережені процедури. Вихід зі збереженої процедури можна здійснити за допомогою команди RETURN.

Видалення збереженої процедури, здійснюється командою:

```
DROP PROCEDURE {ім'я_процедури} [,..n]
```

Для виконання збереженої процедури, використовується команда:

```
[[ EXEC [ UTE] ім'я_процедури [;номер]
[[@ім'я_параметра=]{значення | @ім'я_змінної}
                [OUTPUT ]|[DEFAULT ]][,..n]
```

Якщо виклик збереженої процедури, не є єдиною командою в пакеті, то присутність команди EXECUTE обов'язкова. Більше того, цю команду потрібно для виклику процедури з тіла іншої процедури або тригера.

Використання ключового слова OUTPUT при виклику процедури дозволяється тільки для параметрів, які були оголошені при створенні процедури з ключовим словом OUTPUT.

Коли ж при виклику процедури для параметра вказується ключове слово DEFAULT, то буде використано значення за умовчанням. Природно, вказане слово DEFAULT дозволяється тільки для тих параметрів, для яких визначено значення за умовчанням.

З синтаксису команди EXECUTE видно, що імена параметрів можуть бути опущені при виклику процедури. Проте в цьому випадку користувач повинен вказувати значення для параметрів в тому ж порядку, в якому вони перераховувалися при створенні процедури. Присвоїти параметру значення за умовчанням, просто пропустивши його при перерахуванні не можна. Якщо ж вимагається опустити параметри, для яких визначено значення за умовчанням, досить явної вказівки імен параметрів при виклику збереженої процедури. Більше того, у такий спосіб можна перераховувати параметри і їх значення в довільному порядку.

Також слід зазначити, що при виклику процедури вказуються або імена параметрів зі значеннями, або тільки значення без імені параметра. Їх комбінування не допускається.

### 5.1.3 Тригери

Тригери є одним з різновидів процедур, що зберігаються. Їх виконання відбувається при виконанні для таблиці якого-небудь оператора мови маніпулювання даними (DML). Тригери використовуються для перевірки цілісності даних, а також для відкату транзакцій.

Тригер – це збережена процедура, виконання якої обумовлене настанням певних подій усередині реляційної бази даних. Застосування тригерів переважно дуже зручне для користувачів бази даних. Та все ж їх використання часто пов'язане з додатковими витратами ресурсів на операції введення/виводу. У тому випадку, коли тих же результатів (з набагато меншими непродуктивними витратами ресурсів) можна добитися за допомогою процедур, що зберігаються, або прикладних програм, застосування тригерів недоцільне.

Тригери – особливий інструмент SQL-сервера, використовуваний для підтримки цілісності даних у базі даних. За допомогою обмежень цілісності, правил і значень за умовчанням не завжди можна добитися потрібного рівня функціональності. Часто вимагається реалізувати складні алгоритми перевірки даних, що гарантують їх достовірність і реальність. Крім того, іноді необхідно відстежувати зміни значень таблиці, щоб потрібним чином змінити пов'язані дані. Тригери можна розглядати як свого роду фільтри, що починають діяти після виконання усіх операцій відповідно до правил, стандартних значень і так далі.

Тригер є спеціальним типом процедур, що зберігаються, які запускаються сервером автоматично при спробі зміни даних в таблицях, з якими тригери зв'язані. Кожен тригер прив'язується до конкретної таблиці. Усі вироблювані ним модифікації даних розглядаються як одна транзакція. У разі виявлення помилки або порушення цілісності даних відбувається відкат цієї транзакції. Тим самим внесення змін забороняється. Відміняються також усі зміни, вже зроблені тригером.

Створює тригер тільки власник бази даних. Це обмеження дозволяє уникнути випадкової зміни структури таблиць, способів зв'язку з ними інших об'єктів і тому подібне

Тригер є дуже корисним і в той же час небезпечним засобом. Так, при неправильній логікі його роботи можна легко знищити цілу базу даних, тому тригери необхідно дуже ретельно відлагоджувати.

На відміну від звичайної підпрограми, тригер виконується неявно в кожному випадку виникнення тригерної події, до того ж він не має аргументів. Приведення його в дію іноді називають запуском тригера. За допомогою тригерів досягаються наступні цілі:

- перевірка коректності введених даних і виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати за допомогою обмежень цілісності, встановлених для таблиці;
- видача попереджень, що нагадують про необхідність виконання деяких дій при оновленні таблиці, реалізованому певним чином;

- накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни і тих осіб, які їх виконали;
- підтримка реплікації.

Основний формат команди CREATE TRIGGER показаний нижче:

<Визначення_тригера>::=
CREATE TRIGGER ім'я_тригера
BEFORE | AFTER <тригерна_подія>
ON <ім'я_таблиці>
[REFERENCING
<список_старих_або_нових_псевдонімів>]
[FOR EACH { ROW | STATEMENT}]
[WHEN(умова_тригера)]
<тіло_тригера>

Тригерні події складаються зі вставки, видалення і оновлення рядків в таблиці. У останньому випадку для тригерної події можна вказати конкретні імена стовпців таблиці. Час запуску тригера визначається за допомогою ключових слів BEFORE (тригер запускається до виконання пов'язаних з ним подій) або AFTER (після їх виконання).

Виконувані тригером дії задаються для кожного рядка (FOR EACH ROW), охопленого цією подією, або тільки один раз для кожної події (FOR EACH STATEMENT).

Позначення <список_старих_або_нових_псевдонімів> відноситься до таких компонент, як старий або новий рядок (OLD / NEW) або стара або нова таблиця (OLD TABLE / NEW TABLE). Ясно, що старі значення не застосовні для подій вставки, а нові – для подій видалення.

За умови правильного використання тригери можуть стати дуже потужним механізмом. Основна їх перевага полягає в тому, що стандартні функції зберігаються усередині бази даних і погоджено активізуються при

кожному її оновленні. Це може істотно спростити додатки. Проте слід згадати і про присутні тригеру недоліки:

- складність: при переміщенні деяких функцій у базу даних ускладнюються завдання її проектування, реалізації і адміністрування;
- прихована функціональність: перенесення частини функцій у базу даних і збереження їх у вигляді одного або декількох тригерів іноді призводить до приховання від користувача деяких функціональних можливостей. Хоча це певною мірою спрощує його роботу, але, на жаль, може стати причиною незапланованих, потенційно небажаних і шкідливих побічних ефектів, оскільки в цьому випадку користувач не в змозі контролювати усі процеси, що відбуваються у базі даних;
- вплив на продуктивність: перед виконанням кожної команди по зміні стану бази цих СУБД повинна перевірити тригерну умову з метою з'ясування необхідності запуску тригера для цієї команди. Виконання подібних обчислень позначається на загальній продуктивності СУБД, а в моменти пікового навантаження її зниження може стати особливо помітним. Очевидно, що при зростанні кількості тригерів збільшуються і накладні витрати, пов'язані з такими операціями.

Неправильно написані тригери можуть привести до серйозних проблем, таким, наприклад, як поява «мертвих» блокувань. Тригери здатні тривалий час блокувати безліч ресурсів, тому слід звернути особливу увагу на зведення до мінімуму конфліктів доступу.

У СУБД MS SQL Server використовується наступний оператор створення або зміни тригера:

```
<Визначення_тригера>::=
{CREATE | ALTER} TRIGGER ім'я_тригера
ON {ім'я_таблиці | ім'я_перегляду }
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
```

```
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
     sql_оператор[..n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
   [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(ім'я_стовпця)
[ {AND | OR} UPDATE(ім'я_стовпця)] [..n]
|
IF (COLUMNS_UPDATES(){оператор_біт_обробки}
   біт_маска_зміни)
{оператор_біт_порівняння }біт_маска [..n]}
sql_оператор [..n]}}
```

Тригер може бути створений тільки в поточній базі даних, але допускається звернення усередині тригера до інших баз даних, у тому числі і розташованим на видаленому сервері.

Розглянемо призначення аргументів з команди CREATE | ALTER TRIGGER.

Ім'я тригера має бути унікальним в межах бази даних. Додатково можна вказати ім'я власника.

При вказівці аргументу WITH ENCRYPTION сервер виконує шифрування коду тригера, щоб ніхто, включаючи адміністратора, не міг отримати до нього доступ і прочитати його. Шифрування часто використовується для приховання авторських алгоритмів обробки даних, що є інтелектуальною власністю програміста або комерційною таємницею.

У MS SQL Server існує два параметри, визначальних поведінку тригерів:

- AFTER. Тригер виконується після успішного виконання команд, що викликали його. Якщо ж команди з якої-небудь причини не можуть бути успішно завершені, тригер не виконується. Слід зазначити, що зміни даних в результаті виконання запиту користувача і виконання тригера здійснюється в тілі однієї транзакції: якщо станеться відкат тригера, то будуть відхилені і призначені для користувача зміни. Можна визначити декілька AFTER-тригерів для кожної операції (INSERT, UPDATE, DELETE). Якщо для таблиці передбачено виконання декількох AFTER-тригерів, то за допомогою системної процедури sp_settriggerorder, що зберігається, можна вказати, який з них виконуватиметься першим, а який останнім. За умовчанням в SQL Server усі тригери являються AFTER-тригерами.
- INSTEAD OF. Тригер викликається замість виконання команд. На відміну від AFTER-тригера INSTEAD OF-тригер може бути визначений як для таблиці, так і для представлення. Для кожної операції INSERT, UPDATE, DELETE можна визначити тільки один INSTEAD OF-тригер.

Тригери розрізняють за типом команд, на які вони реагують.

Існує три типи тригерів :

- INSERT TRIGGER – запускаються при спробі вставки даних за допомогою команди INSERT.
- UPDATE TRIGGER – запускаються при спробі зміни даних за допомогою команди UPDATE.
- DELETE TRIGGER – запускаються при спробі видалення даних за допомогою команди DELETE.

Конструкції [ DELETE] [,] [ INSERT] [,] [ UPDATE] і FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] визначають, на яку команду реагуватиме тригер. При його створенні має бути вказана хоч би одна команда. Допускається створення тригера, що реагує на дві або на усі три команди.

Аргумент WITH APPEND дозволяє створювати декілька тригерів кожного типу.

При створенні тригера з аргументом NOT FOR REPLICATION забороняється його запуск під час виконання модифікації таблиць механізмами реплікації.

Конструкція AS sql_оператор[..n] визначає набір SQL-операторів і команд, які будуть виконані при запуску тригера.

Відмітимо, що усередині тригера не допускається виконання низки операцій, таких, наприклад, як:

- створення, зміна і видалення бази даних;
- відновлення резервної копії бази даних або журналу транзакцій.

Виконання цих команд не дозволене, оскільки вони не можуть бути скасовані у разі відкату транзакції, в якій виконується тригер. Ця заборона навряд чи може якимсь чином позначитися на функціональності створюваних тригерів. Важко знайти таку ситуацію, коли, наприклад, після зміни рядка таблиці потрібно буде виконати відновлення резервної копії журналу транзакцій.

При виконанні команд додавання, зміни і видалення записів сервер створює дві спеціальні таблиці: inserted і deleted . В них містяться списки рядків, які будуть вставлені або видалені після закінчення транзакції. Структура таблиць inserted і deleted ідентична структурі таблиць, для якої визначається тригер. Для кожного тригера створюється свій комплект таблиць inserted і deleted, тому ніякий інший тригер не зможе отримати до них доступ. Залежно від типу операції, виконання тригера, що викликала, вміст таблиць inserted і deleted може бути різним:

- команда INSERT – в таблиці inserted містяться усі рядки, які користувач намагається вставити в таблицю; у таблиці deleted не буде жодного рядка; після завершення тригера усі рядки з таблиці inserted перемістяться в початкову таблицю;

- команда DELETE – в таблиці deleted міститимуться усі рядки, які користувач спробує видалити; тригер може перевірити кожен рядок і визначити, чи дозволено її видалення; у таблиці inserted не опиниться жодного рядка;
- команда UPDATE – при її виконанні в таблиці deleted знаходяться старі значення рядків, які будуть видалені при успішному завершенні тригера. Нові значення рядків містяться в таблиці inserted. Ці рядки додадуться в початкову таблицю після успішного виконання тригера.

Для отримання інформації про кількість рядків, яка буде змінено при успішному завершенні тригера, можна використовувати функцію @@ROWCOUNT; вона повертає кількість рядків, оброблених останньою командою. Слід підкреслити, що тригер запускається не при спробі змінити конкретний рядок, а у момент виконання команди зміни. Одна така команда впливає на безліч рядків, тому тригер повинен обробляти усі ці рядки.

Якщо тригер виявив, що з 100 рядків, що вставляються, змінюваних або таких, що видаляються, тільки одна не задовольняє тим або іншим умовам, то ніякий рядок не буде вставлений, змінений або видалений. Така поведінка обумовлена вимогами транзакції - мають бути виконані або усі модифікації, або жодній.

Тригер виконується як неявно певна транзакція, тому усередині тригера допускається застосування команд управління транзакціями. Зокрема, при виявленні порушення обмежень цілісності для переривання виконання тригера і відміни усіх змін, які намагався виконати користувач, необхідно використовувати команду ROLLBACK TRANSACTION.

Для отримання списку стовпців, змінених при виконанні команд INSERT або UPDATE, що викликали виконання тригера, можна використовувати функцію COLUMNS_UPDATED(). Вона повертає двійкове число, кожен біт якого, починаючи з молодшого, відповідає одному стовпцю таблиці (в порядку дотримання стовпців при створенні

таблиці). Якщо біт встановлений в значення "1", то відповідний стовпець був змінений. Крім того, факт зміни стовпця визначає і функція UPDATE(ім'я_стовпця).

Для видалення тригера використовується команда:

DROP TRIGGER {ім'я_тригера} [,..n]

### 5.1.4 Курсори

Запит до реляційної бази даних зазвичай повертає декілька рядків (записів) даних, але застосунок за один раз обробляє лише один запис. Навіть якщо він має справу одночасно з декількома рядками (наприклад, виводить дані у формі електронних таблиць), їх кількість як і раніше обмежена. Крім того, при модифікації, видаленні або додаванні даних робочою одиницею являється рядок. У цій ситуації на перший план виступає концепція курсору, і в такому контексті курсор – покажчик на рядок.

Курсор в SQL – це область в пам'яті бази даних, яка призначена для зберігання останнього оператора SQL. Якщо поточний оператор – запит до бази даних, в пам'яті зберігається і рядок даних запиту, званий поточним значенням, або поточним рядком курсору. Вказана область в пам'яті пойменована і доступна для прикладних програм.

Зазвичай курсори використовуються для вибору з бази даних деякої підмножини інформації, що зберігається в ній. У кожен момент часу прикладною програмою може бути перевірений один рядок курсору. Курсори часто застосовуються в операторах SQL, вбудованих в написані на мовах процедурного типу прикладні програми. Деякі з них неявно створюються сервером бази даних, тоді як інші визначаються програмістами.

Відповідно до стандарту SQL при роботі з курсорами можна виділити наступні основні дії:

- створення або оголошення курсору;

- відкриття курсору, тобто наповнення його даними, які зберігаються у багаторівневій пам'яті;
- вибірка з курсору і зміна з його допомогою рядків даних;
- закриття курсору, після чого він стає недоступним для призначених для користувача програм;
- звільнення курсору, тобто видалення курсору як об'єкту, оскільки його закриття необов'язково звільняє асоційовану з ним пам'ять.

У різних реалізаціях визначення курсору може мати деякі відмінності. Так, наприклад, іноді розробник повинен явним чином звільнити пам'ять, що виділяється для курсору. Після звільнення курсору асоційована з ним пам'ять також звільняється. При цьому стає можливим повторне використання його імені. У інших реалізаціях при закритті курсору звільнення пам'яті відбувається неявним чином. Відразу після відновлення вона стає доступною для інших операцій: відкриття іншого курсору і так далі

В деяких випадках застосування курсору неминуче. Проте по можливості цього слід уникати і працювати із стандартними командами обробки даних: SELECT, UPDATE, INSERT, DELETE. Окрім того, що курсори не дозволяють проводити операції зміни над усім об'ємом даних, швидкість виконання операцій обробки даних за допомогою курсору помітно нижче, ніж у стандартних засобів SQL.

SQL Server підтримує три види курсорів :

- курсори SQL застосовуються в основному усередині тригерів, процедур, що зберігаються, і сценаріїв;
- курсори сервера діють на сервері і реалізують програмний інтерфейс застосунків для ODBC, OLE DB, DB_Library;
- курсори клієнта реалізуються на самому клієнтові. Вони вибирають увесь результуючий набір рядків з сервера і зберігають його локально, що дозволяє прискорити операції обробки даних за рахунок зниження втрат часу на виконання мережевих операцій.

Різні типи розрахованих на багато користувачів застосунків вимагають і різних типів організації паралельного доступу до даних. Деяким застосункам потрібний негайний доступ до інформації про зміни у базі даних. Це характерно, наприклад, для систем резервування квитків. У інших випадках, наприклад, в системах статистичної звітності, важлива стабільність даних, адже якщо вони постійно модифікуються, програми не зможуть ефективно відображати інформацію. Різним застосункам потрібні різні реалізації курсорів.

У середовищі SQL Server типи курсорів розрізняються по можливостях, що надаються. Тип курсору визначається на стадії його створення і не може бути змінений. Деякі типи курсорів можуть виявляти зміни, зроблені іншими користувачами в рядках, включених в результуючий набір. Проте SQL Server відстежує зміни таких рядків тільки на стадії звернення до рядка і не дозволяє відстежувати зміни, коли рядок вже лічений.

Курсори діляться на дві категорії: послідовні і прокручувані. Послідовні дозволяють вибирати дані тільки в одному напрямі – від початку до кінця. Прокручувані ж курсори надають велику свободу дій – допускається переміщення в обох напрямах і перехід до довільного рядка результуючого набору курсору. Якщо програма здатна модифікувати дані, на які вказує курсор, він називається прокручуваним і таким, що модифікується. Говорячи про курсори, не слід забувати про ізольованість транзакцій. Коли один користувач модифікує запис, інший читає її за допомогою власного курсору, більше того, він може модифікувати той же запис, що робить необхідним дотримання цілісності даних.

SQL Server підтримує курсори статичні, динамічні, послідовні і керовані набором ключів.

У схемі із статичним курсором інформація читається з бази даних один раз і зберігається у вигляді моментального знімка (за станом на деякий момент часу), тому зміни, внесені у базу даних іншим користувачем, не видно. На час відкриття курсору сервер встановлює

блокування на усі рядки, включені в його повний результуючий набір. Статичний курсор не змінюється після створення і завжди відображає той набір даних, який існував на момент його відкриття.

Якщо інші користувачі змінять в початковій таблиці включені в курсор дані, це ніяк не вплине на статичний курсор.

У статичний курсор вносити зміни неможливо, тому він завжди відкривається в режимі «тільки для читання».

Динамічний курсор підтримує дані в «живому» стані, але це вимагає витрат мережевих і програмних ресурсів. При використанні динамічних курсорів не створюється повна копія початкових даних, а виконується динамічна вибірка з початкових таблиць тільки при зверненні користувача до тих або інших даних. На час вибірки сервер блокує рядки, а усі зміни, що вносяться користувачем в повний результуючий набір курсору, будуть видні в курсорі. Проте якщо інший користувач вносить зміни вже після вибірки даних курсором, то вони не відіб'ються в курсорі .

Курсор, керований набором ключів, знаходиться посередині між цими крайнощами. Записи ідентифікуються на момент вибірки, і тим самим відстежуються зміни. Такий тип курсору корисний при реалізації прокрутки назад – тоді додавання і видалення рядів не видно, поки інформація не оновиться, а драйвер вибирає нову версію запису, якщо в неї були внесені зміни.

Послідовні курсори не дозволяють виконувати вибірку даних у зворотному напрямі. Користувач може вибирати рядки тільки від початку до кінця курсору . Послідовний курсор не зберігає набір усіх рядків. Вони прочитуються з бази даних, як тільки вибираються в курсорі, що дозволяє динамічно відбивати усі зміни, що вносяться користувачами у базу даних за допомогою команд INSERT, UPDATE, DELETE. У курсорі видно самий останній стан даних.

Статичні курсори забезпечують стабільний погляд на дані. Вони хороші для систем «складування» інформації: застосунків для систем звітності або для статистичних і аналітичних цілей. Крім того, статичний

курсор краще за інших справляється з вибіркою великої кількості даних. Навпаки, в системах електронних покупок або резервування квитків потрібне динамічне сприйняття оновлюваної інформації у міру внесення змін. У таких випадках використовується динамічний курсор. У цих застосуваннях об'єм передаваних даних, як правило, невеликий, а доступ до них здійснюється на рівні рядів (окремих записів). Груповий доступ зустрічається дуже рідко.

Управління курсором реалізується шляхом виконання наступних команд :

- DECLARE – створення або оголошення курсору ;
- OPEN – відкриття курсору, тобто наповнення його даними;
- FETCH – вибірка з курсору і зміна рядків даних за допомогою курсору;
- CLOSE – закриття курсору ;
- DEALLOCATE – звільнення курсору, тобто видалення курсору як об'єкту.

У стандарті SQL для створення курсору передбачена наступна команда:

```
<створення_курсору>::=
   DECLARE ім'я_курсору
      [INSENSITIVE][SCROLL] CURSOR
   FOR SELECT_оператор
   [FOR { READ_ONLY | UPDATE
      [OF ім'я_стовпця[,..n]]}]
```

При використанні ключового слова INSENSITIVE буде створений статичний курсор. Зміни даних не дозволяються, крім того, не відображаються зміни, зроблені іншими користувачами. Якщо ключове слово INSENSITIVE відсутнє, створюється динамічний курсор.

При вказівці ключового слова SCROLL створений курсор можна прокручувати у будь-якому напрямі, що дозволяє застосовувати будь-які команди вибірки. Якщо цей аргумент опускається, то курсор виявиться послідовним, тобто його перегляд буде можливий тільки в одному напрямі – від початку до кінця.

SELECT-оператор задає тіло запиту SELECT, за допомогою якого визначається результуючий набір рядків курсору.

При вказівці аргументу FOR READ_ONLY створюється курсор «тільки для читання», і ніякі модифікації даних не дозволяються. Він відрізняється від статичного, хоча останній також не дозволяє міняти дані. В якості курсору "тільки для читання" може бути оголошений динамічний курсор, що дозволить відображати зміни, зроблені іншим користувачем.

Створення курсору з аргументом FOR UPDATE дозволяє виконувати в курсорі зміну даних або у вказаних стовпцях, або, за відсутності аргументу OF ім'я_стовпця, в усіх стовпцях.

У середовищі MS SQL Server прийнятий наступний синтаксис команди створення курсору :

```
<створення_курсору>::=
   DECLARE ім'я_курсору CURSOR [LOCAL | GLOBAL]
   [FORWARD_ONLY | SCROLL]
   [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
   [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
   [TYPE_WARNING]
   FOR SELECT_оператор
   [FOR UPDATE [OF ім'я_стовпця[,..n]]]
```

При використанні ключового слова LOCAL буде створений локальний курсор, який видно тільки в межах пакету, що створив його, тригера, збереженої процедури, або призначеної для користувача функції. Після закінчення роботи пакету, тригера, процедури або функції курсор

неявно знищується. Щоб передати вміст курсору за межі конструкції, що створила його, необхідно присвоїти його параметру аргумент OUTPUT.

Якщо вказано ключове слово GLOBAL, створюється глобальний курсор; він існує до закриття поточного з'єднання.

При вказівці FORWARD_ONLY створюється послідовний курсор ; вибірку даних можна здійснювати тільки в напрямі від першого рядка до останнього.

При вказівці SCROLL створюється прокручуваний курсор, звертатися до даних можна у будь-якому порядку і у будь-якому напрямі.

При вказівці STATIC створюється статичний курсор. При вказівці KEYSET створюється ключовий курсор. При вказівці DYNAMIC створюється динамічний курсор.

Якщо для курсору READ_ONLY вказати аргумент FAST_FORWARD, то створений курсор буде оптимізований для швидкого доступу до даних. Цей аргумент не може бути використаний спільно з аргументами FORWARD_ONLY і OPTIMISTIC.

У курсорі, створеному з вказівкою аргументу OPTIMISTIC, забороняється зміна і видалення рядків, які були змінені після відкриття курсору.

При вказівці аргументу TYPE_WARNING сервер інформуватиме користувача про неявну зміну типу курсору, якщо він несумісний із запитом SELECT.

Для відкриття курсору і наповнення його даними з вказаного при створенні курсору запиту SELECT використовується наступна команда:

```
OPEN {{[GLOBAL]ім'я_курсору }
        |@ім'я_змінної_курсору}
```

Після відкриття курсору відбувається виконання пов'язаного з ним оператора SELECT, вихідні дані якого зберігаються у багаторівневій пам'яті.

Відразу після відкриття курсору можна вибрати його вміст (результат виконання відповідного запиту) за допомогою наступної команди:

```
FETCH [[NEXT | PRIOR | FIRST | LAST
   | ABSOLUTE {номер_рядка
   | @змінна_номера_рядка}
   | RELATIVE {номер_рядка |
      @змінна_номера_рядка}]
   FROM ]{{[GLOBAL ]ім'я_курсору }|
      @ім'я_змінної_курсору }
   [INTO @ім'я_змінної [,..n]]
```

При вказівці FIRST буде повернена найперший рядок повного результуючого набору курсору, яка стає поточним рядком.

При вказівці LAST повертається самий останній рядок курсору. Вона ж стає поточним рядком.

При вказівці NEXT повертається рядок, що знаходиться в повному результуючому наборі відразу ж після поточної. Тепер вона стає поточною. За умовчанням команда FETCH використовує саме цей спосіб вибірки рядків.

Ключове слово PRIOR повертає рядок, що знаходиться перед поточною. Вона і стає поточною.

Аргумент ABSOLUTE {номер_рядка | @змінна_номера_рядка} повертає рядок по її абсолютному порядковому номеру в повному результуючому наборі курсору. Номер рядка можна задати за допомогою константи або як ім'я змінної, в якій зберігається номер рядка. Змінна повинна мати цілочисельний тип даних. Вказуються як позитивні, так і негативні значення. При вказівці позитивного значення рядок відлічується від початку набору, негативного, - від кінця. Вибраний рядок стає поточним. Якщо вказано нульове значення, рядок не повертається.

Аргумент RELATIVE {кілок_рядка | @змінна_кілок_рядка} повертає рядок, що знаходиться через вказану кількість рядків після поточної. Якщо вказати негативне значення числа рядків, то буде повернений рядок, що знаходиться за вказану кількість рядків перед поточною. При вказівці нульового значення повернеться поточний рядок. Повернений рядок стає поточним.

Щоб відкрити глобальний курсор, перед його ім'ям вимагається вказати ключове слово GLOBAL. Ім'я курсору також може бути вказане за допомогою змінної.

У конструкції INTO @ім'я_змінної [,..n] задається список змінних, в яких будуть збережені відповідні значення стовпців поверненого рядка. Порядок вказівки змінних повинен відповідати порядку стовпців в курсорі, а тип даних змінної - типу даних в стовпці курсору. Якщо конструкція INTO не вказана, то поведінка команди FETCH нагадуватиме поведінку команди SELECT - дані виводяться на екран.

Для виконання змін за допомогою курсору необхідно виконати команду UPDATE в наступному форматі:

```
UPDATE ім'я_таблиці SET {ім'я_стовпця={
   DEFAULT | NULL | вираження}}[,..n]
   WHERE CURRENT OF {{[GLOBAL] ім'я_курсору}
   |@ім'я_змінної_курсору}
```

За одну операцію можуть бути змінені декілька стовпців поточного рядка курсору, але усі вони повинні належати одній таблиці.

Для видалення даних за допомогою курсору використовується команда DELETE в наступному форматі:

```
DELETE ім'я_таблиці
   WHERE CURRENT OF {{[GLOBAL] ім'я_курсору}
   |@ім'я_змінної_курсору}
```

В результаті буде видалений рядок, встановлений поточним в курсорі.

Для закриття курсору використовується команда CLOSE в наступному форматі:

CLOSE {ім'я_курсору | @ім'я_змінної_курсору}

Після закриття курсор стає недоступним для користувачів програми. При закритті знімаються усі блокування, встановлені в процесі його роботи. Закриття може застосовуватися тільки до відкритих курсорів. Закритий, але не звільнений курсор може бути повторно відкритий. Не допускається закривати невідкритий курсор.

Закриття курсору необов'язково звільняє асоційовану з ним пам'ять. У деяких реалізаціях треба явним чином звільнити її за допомогою оператора DEALLOCATE. Після звільнення курсору звільняється і пам'ять, при цьому стає можливим повторне використання імені курсору.

DEALLOCATE { ім'я_курсору |
@ім'я_змінної_курсору }

Для контролю досягнення кінця курсору рекомендується застосовувати функцію:

@@FETCH_STATUS

Функція @@FETCH_STATUS повертає:

0, якщо вибірка завершилася успішно;

–1, якщо вибірка завершилася невдало внаслідок спроби вибірки рядка, що знаходиться за межами курсору ;

–2, якщо вибірка завершилася невдало внаслідок спроби звернення до видаленого або зміненого рядка.

## 5.2 Практичне опанування теми 5

### 5.2.1 Передумови виконання завдань теми 5

При виконанні практичних завдань теми 5 передбачається використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи Microsoft SQL Server. При цьому передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Для початку виконання практичних завдань треба підключити базу даних (delivery або dlvr), яку було створено при виконанні практичних завдань теми «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

Увага! Результати використання програмних об'єктів, що розглядаються далі, повинні бути результативними (тобто в результаті виконання запиту повинні бути виведені один або кілька записів). Відсутність результату запиту є ознакою помилок під час побудови запиту, невідповідності запиту наявним даним тощо. Такий запит потребує аналізу та перевірки. Також передбачається, що результати запитів ґрунтуються на тих даних, які були введені до бази даних при виконанні практичних завдань теми «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

### 5.2.2 Побудова та використання створюваних користувачем функцій

Для отримання доступу до переліку функцій, потрібно у списку об'єктів бази даних відкрити пункт Programmability і в ньому відкрити пункт Functions (рисунок 5.1). Потім треба відкрити пункт, що відповідає певному типу функцій.

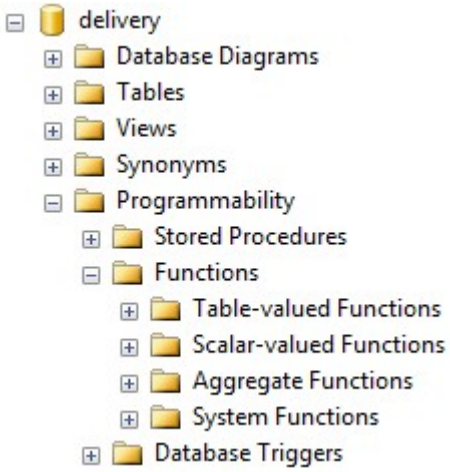



Рисунок 5.1

Розглянемо послідовність дій при створенні та використанні скалярної функції.

Призначення функції: розрахунок загальної кількості одиниць продукції, закупленої на підставі певного договору (при цьому номер договору повинен передаватися до функції як параметр).

Для створення функції треба.

1. Клацнути правою кнопкою миші по пункту Scalar-valued Functions і в меню вибрати пункт New Scalar-valued Function…. В результаті буде створено запит, що містить «заготівку» функції.

2. Замість «заготівки» ввести текст запиту, наведений на рисунку 5.2. При цьому зберігати коментарі, які залишилися від «заготівки» (вони позначені зеленим кольором), не обов'язково.

3. Створити функцію, натиснувши кнопку Execute. У разі, якщо в тексті запиту немає помилок, буде створено функцію. Переглянути список скалярних функцій та пересвідчитися, що функцію було успішно створено.

4. Перевірити працездатність функції. Для цього створити запит, який наведено на рисунку 5.3.

5. Виконати запит. Результат виконання, тобто кількість одиниць продукції, закупленої на підставі договору 1, наведений на рисунку 5.4.

```sql
-- ================================================
-- Template generated from Template Explorer using:
-- Create Scalar Function (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the function.
-- ================================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date, ,>
-- Description: <Description, ,>
-- =============================================
CREATE FUNCTION dbo.supplied_amount (@nomer int)
Returns INT
AS
BEGIN
DECLARE @c INT
SET @c = (SELECT SUM (Amount)
        FROM Supplied
        WHERE ContractNumber = @nomer)
RETURN (@c)
END
GO
```

Рисунок 5.2

```sql
USE delivery

DECLARE @Kil INT
SET @kil = dbo.supplied_amount(1)
SELECT @kil AS кількість
```

Рисунок 5.3

| | кількість |
|---|---|
| 1 | 83 |

Рисунок 5.4

Розглянемо послідовність дій при створенні та використанні табличної функції (inline table-valued function).

Призначення функції: розрахунок загальної кількості одиниць продукції та вартості продукції, закупленої на підставі кожного договору за певний період часу, який визначається датою початку та датою закінчення (при цьому дати початку та закінчення періоду часу повинні передаватися до функції як параметри). При цьому також повинні бути зазначені дані про постачальників (для юридичних осіб – податковий номер, для фізичних осіб – прізвище та ініціали) та їх контактні дані.

Для створення функції треба клацнути правою кнопкою миші по пункту Table-valued Functions і в меню вибрати пункт New Inline Table-valued Function…. Послідовність дій при створенні та використанні такої функції аналогічна розглянутим вище діям при створенні скалярної функції.

Текст запиту для створення функції наведений на рисунку 5.5. Текст запиту для перевірки роботи функції наведений на рисунку 5.6. Результат роботи функції при використанні вказаних значень параметрів наведений на рисунку 5.7.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- ============================================
CREATE FUNCTION dbo.supplied_aggregate (@date1 DATETIME, @date2 DATETIME)
Returns TABLE
AS
RETURN
(
SELECT Contracts.ContractNumber, ContractDate, SUM(Amount) AS кількість, SUM(Amount*PricePerItem) AS сума,
       CAST(Note AS CHAR(40)) AS КонтактиПостачальника,
       ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
       AS Постачальник
FROM ((Suppliers LEFT JOIN IndividualEntrepreneurs ON Suppliers.SupplierID=IndividualEntrepreneurs.SupplierID)
      LEFT JOIN LegalEntities ON Suppliers.SupplierID=LegalEntities.SupplierID)
      INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID
      INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
WHERE ContractDate BETWEEN @date1 AND @date2
GROUP BY Contracts.ContractNumber, ContractDate, CAST(Note AS CHAR(40)),
ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
)
GO
```

Рисунок 5.5

```
USE delivery
SELECT * FROM supplied_aggregate('19990901','19990915')
```

Рисунок 5.6

| | ContractNumber | ContractDate | кількість | сума | КонтактиПостачальника | Постачальник |
|---|---|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | 83 | 77527.14 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |
| 2 | 2 | 1999-09-10 00:00:00.000 | 67 | 73993.53 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |
| 3 | 3 | 1999-09-10 00:00:00.000 | 148 | 99135.68 | м. Харків, пр. Науки, 55, к. 108, тел.32 | Петренко П.П. |

Рисунок 5.7

Розглянемо послідовність дій при створенні та використанні багатооператорної табличної функції (multi-statement table-valued function).

Призначення функції: розрахунок загальної кількості одиниць продукції та вартості продукції, закупленої на підставі кожного договору. При цьому також повинні бути зазначені дані про постачальників (для юридичних осіб – податковий номер, для фізичних осіб – прізвище та ініціали) та їх адреса. У результаті виконання функції повинні повертатися дані лише про ті договори, за якими значення загальної кількості одиниць продукції знаходиться у певному інтервалі значень (при цьому значення початку та закінчення цього інтервалу повинні передаватися до функції як параметри).

Для створення функції треба клацнути правою кнопкою миші по пункту Table-valued Functions і в меню вибрати пункт New Multi-statement Table-valued Function…. Послідовність дій при створенні та використанні такої функції аналогічна розглянутим вище діям при створенні скалярної функції.

Текст запиту для створення функції наведений на рисунку 5.8. Зверніть увагу на те, що на відміну від табличної функції, у цій функції застосовано маніпулювання даними за допомогою оператора INSERT-SQL. Текст запиту для перевірки роботи функції наведений на рисунку 5.9. Результат роботи функції при використанні вказаних значень параметрів наведений на рисунку 5.10.

```
CREATE FUNCTION [dbo].[supplied_aggr_amount] (@amount1 int, @amount2 int)
Returns @retFindAmount TABLE
(
   ContractNumber int primary key NOT NULL,
   ContractDate DATETIME NOT NULL,
   sum_amount int NOT NULL,
   sum_cost float NOT NULL,
   supplier_contacts char(40) NOT NULL,
   supplier_data char(20) NOT NULL
)
AS
BEGIN
WITH amount_cte(ContractNumber,ContractDate,sum_amount,sum_cost,supplier_contacts,supplier_data)
   AS (
       SELECT Contracts.ContractNumber, ContractDate, SUM(Amount) AS sum_amount, SUM(Amount*PricePerItem) AS sum_cost,
              CAST(Note AS CHAR(40)) AS supplier_contacts,
              ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
              AS supplier_data
      FROM ((Suppliers LEFT JOIN IndividualEntrepreneurs ON Suppliers.SupplierID=IndividualEntrepreneurs.SupplierID)
             LEFT JOIN LegalEntities ON Suppliers.SupplierID=LegalEntities.SupplierID)
              INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID
              INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
      GROUP BY Contracts.ContractNumber, ContractDate, CAST(Note AS CHAR(40)),
               ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
      HAVING SUM(Amount) BETWEEN @amount1 AND @amount2
      )
      INSERT @retFindAmount
          SELECT ContractNumber,ContractDate,sum_amount,sum_cost,supplier_contacts,supplier_data FROM amount_cte
      RETURN
END;
GO
```

Рисунок 5.8

```
USE delivery
SELECT ContractNumber,ContractDate,sum_amount,sum_cost,supplier_contacts,supplier_data FROM supplied_aggr_amount(100,150)
```

Рисунок 5.9

| | ContractNumber | ContractDate | sum_amount | sum_cost | supplier_contacts | supplier_data |
|---|---|---|---|---|---|---|
| 1 | 3 | 1999-09-10 00:00:00.000 | 148 | 99135,68 | м. Харків, пр. Науки, 55, к. 108, тел.32 | Петренко П.П. |
| 2 | 4 | 1999-09-23 00:00:00.000 | 146 | 85887,96 | м. Харків, пр. Науки, 55, к. 108, тел.32 | Петренко П.П. |
| 3 | 5 | 1999-09-24 00:00:00.000 | 108 | 71793,89 | м. Київ, пр. Перемоги, 154, к. 3 | 00123987 |
| 4 | 6 | 1999-10-01 00:00:00.000 | 117 | 113321,23 | м. Харків, вул. Пушкінська, 77 (тел.33-3 | Іваненко І.І. |

Рисунок 5.10

Наведені вище приклади функцій є дуже простими. Більш детальну інформацію щодо побудови та використання створюваних користувачем функцій засобами мови Transact-SQL, можна, наприклад, отримати за посиланням:

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver16

### 5.2.3 Побудова та використання збережених процедур

Для отримання доступу до переліку збережених процедур потрібно у списку об'єктів бази даних відкрити пункт Programmability і в ньому відкрити пункт Stored Procedures. В результаті з'явиться список використовуваних процедур (якщо такі процедури були створені раніше). Пункт System Stored Procedures відкривати не потрібно.

Розглянемо послідовність дій при створенні та використанні простої збереженої процедури без параметрів.

Призначення збереженої процедури: забезпечити вибірку даних з таблиць «Contracts», «IndividualEntrepreneurs», «LegalEntities».

Для створення збереженої процедури треба.

1. Клацнути правою кнопкою миші по пункту Stored Procedures і в меню вибрати пункт New Stored Procedure…. В результаті буде створено запит, що містить «заготівку» збереженої процедури.

2. Замість «заготівки» ввести текст запиту, наведений на рисунку 5.11. При цьому зберігати коментарі, які залишилися від «заготівки» (вони позначені зеленим кольором), не обов'язково.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[sp_Contracts]
AS
BEGIN
    SET NOCOUNT ON;
    SELECT *
         FROM (Contracts LEFT JOIN LegalEntities ON Contracts.SupplierID=LegalEntities.SupplierID)
              LEFT JOIN IndividualEntrepreneurs ON Contracts.SupplierID=IndividualEntrepreneurs.SupplierID
END
GO
```

Рисунок 5.11

3. Створити збережену процедуру, натиснувши кнопку Execute. У разі, якщо запит виконано успішно, у вікні Messages з'явиться повідомлення Command(s) successfully completed. У цьому випадку вікно запиту з текстом збереженої процедури можна закрити, причому запит зберігати у вигляді файлу не потрібно. Збережена процедура повинна

з'явитися в списку збережених процедур. Якщо процедура відсутня у списку, потрібно клацнути правою кнопкою миші по пункту Stored Procedures і в меню вибрати пункт Refresh.

4. Перевірити працездатність збереженої процедури. Для цього створити запит, який наведено на рисунку 5.12. Цей запит можна потім закрити без збереження. Іншим способом виконання збереженої процедури є її вибір правою кнопкою миші у списку збережених процедур і потім вибір в меню пункту Execute Stored Procedure….

5. Виконати запит. Результат виконання наведений на рисунку 5.13.

```
USE delivery
exec sp_Contracts
```

Рисунок 5.12

| | ContractNumber | ContractDate | SupplierID | ContractName | Comment | SupplierID | TaxNumber | VATNumber | SupplierID | LastNar |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | 1 | Договір № 1 | Підстава - накладна №34 від 30/08/99 | NULL | NULL | NULL | 1 | Іванен |
| 2 | 2 | 1999-09-10 00:00:00.000 | 1 | Договір № 2 | Підстава - рахунок-фактура № 08-78 від 28/08/99 | NULL | NULL | NULL | 1 | Іванен |
| 3 | 3 | 1999-09-10 00:00:00.000 | 3 | Договір № 3 | Підстава - рахунок-фактура № 08-78 від 28/08/99 | NULL | NULL | NULL | 3 | Петрен |
| 4 | 4 | 1999-09-23 00:00:00.000 | 3 | Договір № 4 | Підстава - замовлення № 56 від 28/08/99 | NULL | NULL | NULL | 3 | Петрен |
| 5 | 5 | 1999-09-24 00:00:00.000 | 2 | Договір № 5 | Підстава - накладна № 74 від 11/09/99 | 2 | 00123987 | 19848521 | NULL | NULL |
| 6 | 6 | 1999-10-01 00:00:00.000 | 1 | Договір № 6 | Підстава - рахунок-фактура № 09-12 від 28/09/99 | NULL | NULL | NULL | 1 | Іванен |
| 7 | 7 | 1999-10-02 00:00:00.000 | 2 | Договір № 7 | Підстава - накладна № 85 від 21/09/99 | 2 | 00123987 | 19848521 | NULL | NULL |

Рисунок 5.13

6. При необхідності внесення змін до тексту збереженої процедури, її можна відкрити в режимі редагування, для чого потрібно клацнути правою кнопкою миші на ім'я збереженої процедури в списку збережених процедур і в меню вибрати пункт Modify. В результаті на екран буде виведений запит, що містить текст збереженої процедури. Якщо в текст збереженої процедури вносилися зміни і ці зміни потрібно зберегти, запит, що містить текст збереженої процедури, потрібно виконати. Після успішного виконання запит у вигляді файлу не потрібно зберігати. Для перевірки зміни змісту збереженої процедури, її потрібно знову відкрити в режимі редагування.

Розглянемо послідовність дій при створенні та використанні наступної збереженої процедури, яка, на відміну від попередньої, використовує параметри.

Призначення збереженої процедури: розрахунок загальної кількості одиниць продукції та вартості продукції, закупленої на підставі кожного договору за певний період часу, який визначається датою початку та датою закінчення (при цьому дати початку та закінчення періоду часу повинні передаватися до збереженої процедури як параметри).

Послідовність дій при створенні та використанні такої збереженої процедури аналогічна розглянутим вище діям при створенні збереженої процедури. Текст запиту для створення збереженої процедури наведений на рисунку 5.14. Тексти запитів для виконання збереженої процедури та результат її виконання наведені на рисунках 5.15 та 5.16 відповідно.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[sp_Contracts_aggr]
      @var1 datetime, @var2 datetime
AS
BEGIN
    SET NOCOUNT ON;
    SELECT Contracts.ContractNumber, Contracts.ContractDate,SUM(Amount) AS Amount,SUM(Amount*PricePerItem) AS Cost
         FROM Contracts LEFT JOIN Supplied ON Contracts.ContractNumber=Supplied.ContractNumber
         WHERE ContractDate BETWEEN @var1 AND @var2
         GROUP BY Contracts.ContractNumber,ContractDate
END
GO
```

Рисунок 5.14

```
USE delivery
exec sp_Contracts_aggr '1999/01/01','1999/10/31'
```

Рисунок 5.15

| | ContractNumber | ContractDate | Amount | Cost |
|---|---|---|---|---|
| 1 | 1 | 1999-09-01 00:00:00.000 | 83 | 77527.14 |
| 2 | 2 | 1999-09-10 00:00:00.000 | 67 | 73993.53 |
| 3 | 3 | 1999-09-10 00:00:00.000 | 148 | 99135.68 |
| 4 | 4 | 1999-09-23 00:00:00.000 | 146 | 85887.96 |
| 5 | 5 | 1999-09-24 00:00:00.000 | 108 | 71793.89 |
| 6 | 6 | 1999-10-01 00:00:00.000 | 117 | 113321.23 |
| 7 | 7 | 1999-10-02 00:00:00.000 | 99 | 82928.86 |

Рисунок 5.16

Розглянемо послідовність дій при створенні та використанні наступної збереженої процедури, яка також використовує параметри.

Призначення збереженої процедури: виконання операцій модифікації даних для таблиці «Contracts», тобто додавання нового договору або модифікація даних вже існуючого договору, або видалення існуючого договору.

Послідовність дій при створенні та використанні такої збереженої процедури аналогічна розглянутим вище діям при створенні збереженої процедури. Текст запиту для створення збереженої процедури наведений на рисунку 5.17. Тексти запитів для виконання збереженої процедури з різними параметрами наведені на рисунках 5.18, 5.19, 5.20 відповідно. При виконанні збереженої процедури рекомендується звернути увагу на використовувані значення параметрів. Також рекомендується звернути увагу на те, як будуть змінюватися дані у таблиці «Contracts».

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[sp_dgvr_mdf]
    @Action char(1), @nom_dgvr int, @dgvr_date datetime, @dgvr_kod_post int, @dgvr_comment text
AS
SET NOCOUNT ON
BEGIN
    IF @Action='I'
       BEGIN
          PRINT 'insert'
          INSERT INTO Contracts (ContractDate, SupplierID, Comment)
                                 VALUES (GETDATE(), @dgvr_kod_post, @dgvr_comment)
       END
      ELSE
       IF @Action='U'
          BEGIN
             PRINT 'update'
             UPDATE Contracts SET ContractDate=@dgvr_date, SupplierID=@dgvr_kod_post, Comment=@dgvr_comment
                                 WHERE ContractNumber=@nom_dgvr
          END
         ELSE
          IF @Action='D'
          BEGIN
             PRINT 'delete'
             DELETE FROM Contracts WHERE ContractNumber=@nom_dgvr
          END

END
GO
```

Рисунок 5.17

```
use delivery
exec sp_dgvr_mdf 'I',0,'2008/12/16',2,''
```

Рисунок 5.18

```
use delivery
exec sp_dgvr_mdf 'U',8,'2008/12/31',2,'88888888'
```

Рисунок 5.19

```
use delivery
exec sp_dgvr_mdf 'D',8,'2008/12/31',0,''
```

Рисунок 5.20

Розглянемо послідовність дій при створенні та використанні наступної збереженої процедури, яка використовує не тільки вхідні, але й вихідні параметри.

Призначення збереженої процедури: розрахунок загальної кількості одиниць продукції та вартості продукції, закупленої на підставі певного договору (при номер певного договору повинен передаватися до збереженої процедури як вхідний параметр, а загальну кількість одиниць продукції та вартість продукції збережена процедура повинна повертати як вихідні параметри).

Послідовність дій при створенні та використанні такої збереженої процедури аналогічна розглянутим вище діям при створенні збереженої процедури. Текст запиту для створення збереженої процедури наведений на рисунку 5.21. Текст запиту для виконання збереженої процедури наведений на рисунку 5.22. Результат її виконання наведений на рисунку 5.23.

Наведені приклади збережених процедур є досить простими. Більш детальну інформацію щодо побудови та використання збережених процедур засобами мови Transact-SQL, можна, наприклад, отримати за посиланням:

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver16

```
CREATE PROCEDURE [dbo].[sp_sum_dgvr]
    @dgvrNumber int,
    @sum_kol_vo int output,
    @sum_summa decimal(8,2) out
AS
BEGIN
    SET NOCOUNT ON
    SELECT @sum_kol_vo=SUM(Amount),
           @sum_summa= SUM(Amount*PricePerItem)
        FROM Supplied
        WHERE ContractNumber=@dgvrNumber
END
GO
```

Рисунок 5.21

```
USE [delivery]
GO

DECLARE @return_value int,
        @sum_kol_vo int,
        @sum_summa decimal(8, 2)

EXEC    @return_value = [dbo].[sp_sum_dgvr]
        @dgvrNumber = 1,
        @sum_kol_vo = @sum_kol_vo OUTPUT,
        @sum_summa = @sum_summa OUTPUT

SELECT  @sum_kol_vo as N'@sum_kol_vo',
        @sum_summa as N'@sum_summa'

SELECT  'Return Value' = @return_value

GO
```

Рисунок 5.22

| | @sum_kol_vo | @sum_summa |
|---|---|---|
| 1 | 83 | 77527.14 |

| | Return Value |
|---|---|
| 1 | 0 |

Рисунок 5.23

### 5.2.4 Побудова та використання тригерів

#### 5.2.4.1 Побудова та використання тригерів DML

Розглянемо задачу створення тригера, який контролює наявність дати договору на постачання продукції

Припустимо, що при введенні даних у таблицю «Contracts», в якій зберігається інформація про договори на поставку продукції, поле «ContractDate», в якому зберігається дата укладання договору, має бути обов'язково заповнене, причому в тому випадку, якщо при введенні нового договору це поле залишається незаповненим, у нього має бути автоматично записана поточна дата. Це завдання можна вирішити різними засобами, у тому числі і за допомогою тригера.

Для отримання доступу до переліку тригерів рівня таблиці потрібно у списку таблиць відкрити список об'єктів необхідної таблиці (у даному випадку – Договори) та у цьому списку відкрити пункт Triggers. У результаті з'явиться список тригерів (якщо тригери для таблиці вже було створено раніше).

Для створення нового тригера виконаємо такі дії.

1. Клацнути правою кнопкою миші по пункту Triggers і в меню вибрати пункт New Trigger…. В результаті буде створено запит, який містить «заготівку» тригера. Цю «заготівку» слід змінити, шляхом введення тексту тригера, який наведено рисунку 5.24

```
SET ANSI_NULLS ON
SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [not_null_date] ON  [dbo].[Contracts]
   AFTER INSERT NOT FOR REPLICATION
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @new_dgvr_date datetime, @new_dgvr_nomer int
    SELECT @new_dgvr_date=ContractDate, @new_dgvr_nomer=ContractNumber FROM inserted
    IF @new_dgvr_date IS NULL
       BEGIN
          UPDATE Contracts SET ContractDate=GETDATE() WHERE ContractNumber=@new_dgvr_nomer
       END
END
GO
```

Рисунок 5.24

2. Для створення цього тригера потрібно натиснути кнопку Execute на панелі інструментів. У разі, якщо запит виконано успішно, у вікні Messages з'явиться повідомлення Command(s) successfully completed. У цьому випадку вікно запиту з текстом тригера можна закрити, причому запит зберігати як файл не потрібно. Тригер повинен з'явитися у списку тригерів таблиці. Якщо тригер відсутній у списку, потрібно клацнути правою кнопкою миші по пункту Triggers і в меню вибрати пункт Refresh.

3. Для перевірки роботи тригера потрібно додати новий договір до списку договорів. Це можна зробити, наприклад, за допомогою запиту, або відкривши таблицю у режимі редагування даних. Після успішного виконання запиту слід перевірити стан таблиці «Contracts». У полі «ContractDate» запису, що відповідає новому договору, має бути записана поточна календарна дата.

Увага! Створення такого тригера не є гарним варіантом вирішення цієї задачі. У даному випадку це було зроблено з метою створення прикладу дуже простого тригера. Більш ефективним варіантом рішення є використання значення за замовчуванням для поля «ContractDate». Це можна зробити, наприклад, змінивши властивості поля таблиці. Приклад наведений на рисунку 5.25.

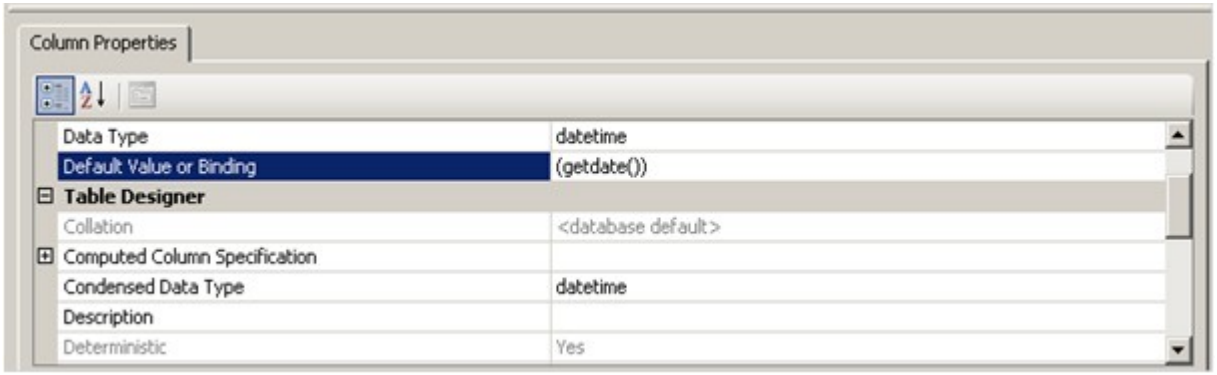



Рисунок 5.25

Також значення за замовчуванням можна визначити у запиті для створення таблиці. Приклад такого запиту наведений на рисунку 5.26.

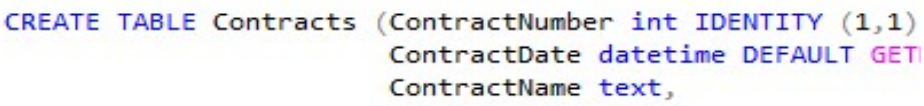



Рисунок 5.26

Розглянемо наступну задачу, пов'язану із контролем цілісності даних. Вирішення цієї задачі потребує створення тригера, який контролює наявність даних про постачальника, як юридичної особи.

Потреба у створенні тригера обумовлена такою причиною. У базі даних зберігається як загальна інформація про постачальників, так і інформація, яка стосується лише постачальників – фізичних осіб або постачальників – юридичних осіб. Кожен постачальник може бути або юридичною або фізичною особою. Це означає, що одночасна наявність даних про постачальника в таблицях «IndividualEntrepreneurs» та «LegalEntities» не допускається з погляду вимог логіки управління бізнесом. Таким чином, виникає необхідність складного контролю відносин цілісності посилань.

Для вирішення цього завдання створимо тригер, який при введенні інформації в таблицю «IndividualEntrepreneurs» контролюватиме наявність коду відповідного постачальника в таблиці «LegalEntities» і блокуватиме введення даних про постачальника як про фізичну особу в тому випадку, якщо вже є дані про цього постачальника як про юридичну особу.

Послідовність дій під час створення тригера аналогічна описаної раніше. Текст тригера наведено на рисунку 5.27. Після введення тексту тригер потрібно зберегти, виконавши запит. Потім слід перевірити працездатність тригера. Для цього спробуємо додати до таблиці «IndividualEntrepreneurs» дані про постачальника, який вже є юридичною особою. Це можна зробити шляхом створення відповідного запиту, або у режимі модифікації даних, які зберігаються у таблиці. Це має ініціювати виконання тригера і операцію буде заблоковано тригером. Приклад спроби введення даних про постачальника як фізичну особу наведений на рисунку 5.28 (дані, що вводяться, мають виключно тестове призначення). Однак про цього постачальника раніше вже було введено дані як про

юридичну особу. Отже, користувач побачить повідомлення, що наведене на рисунку 5.29.

```
SET ANSI_NULLS ON
SET QUOTED_IDENTIFIER ON
GO

CREATE TRIGGER [check_yur_l] ON [dbo].[IndividualEntrepreneurs]
   AFTER INSERT NOT FOR REPLICATION
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @new_fiz_code int, @var1 int
    SELECT @new_fiz_code=SupplierID FROM inserted
    PRINT 'Спроба додати нову фізичну особу з кодом '+STR(@new_fiz_code)
    SELECT @var1=COUNT(SupplierID) FROM LegalEntities WHERE SupplierID=@new_fiz_code
    IF @var1>0
       BEGIN
          DECLARE @DBID int, @DBNAME nvarchar(128)
          SET @DBID=DB_ID()
          SET @DBNAME=DB_NAME()
          RAISERROR(N'The current database ID is:%d, the database name is:%s.',
                    10, 1, @DBID, @DBNAME)
          PRINT 'Постачальник з кодом '+STR(@new_fiz_code)+' вже є юридичною особою'
          ROLLBACK
       END
      ELSE
       PRINT 'Нову фізичну особу з кодом '+STR(@new_fiz_code)+' введено'
END
GO
```

Рисунок 5.27

| | SupplierID | LastName | FirstName | SecondName | RegistrationNumber |
|---|---|---|---|---|---|
| | 1 | Іваненко | Ілля | Іванович | 00143987 |
| | 3 | Петренко | Павло | Петрович | 12345678 |
| | 5 | Сидорчук | Микита | Степанович | 09876541 |
| | 2 | 2222 | 2222 | 2222 | 22222 |
| * | NULL | NULL | NULL | NULL | NULL |

Рисунок 5.28

Слід звернути увагу, що використання у тригері функції RAISERROR не є обов'язковим і зроблено лише з метою ілюстрації можливостей використання цієї функції.

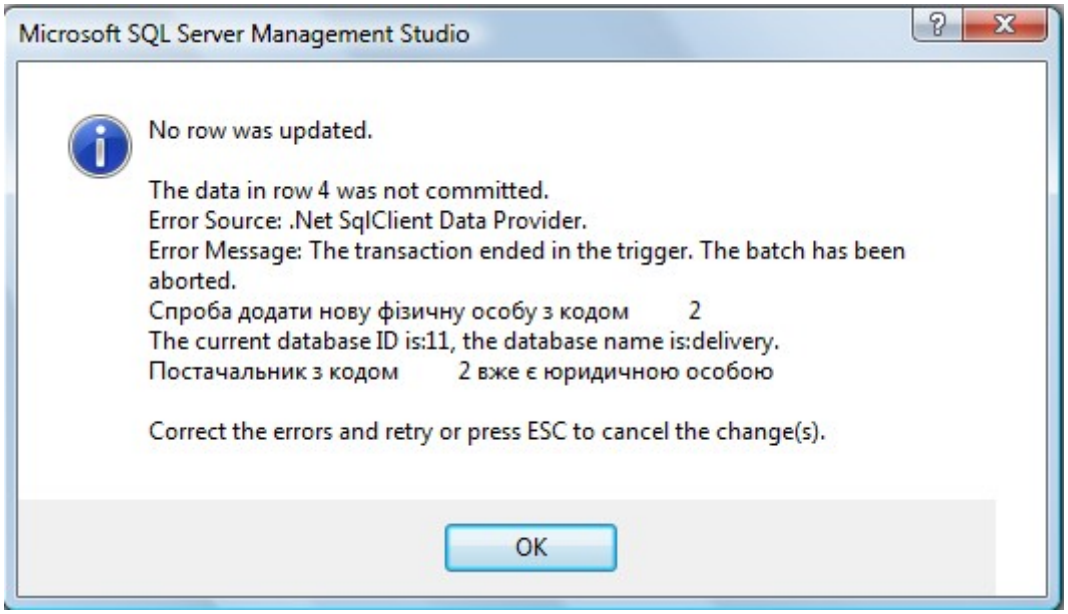



Рисунок 5.29

Контроль стану даних у таблиці «IndividualEntrepreneurs» повинен підтвердити те, що нові дані в цій таблиці не з'явилися. Після цього треба пересвідчитися, що у разі відсутності даних про постачальника як юридичної особи, додавання даних про постачальника як фізичну особу, проходить без перешкод.

Розглянемо наступну задачу, пов'язану із контролем цілісності даних. Цей приклад є досить типовим для практичних задач, але, на жаль, у рамках поточної бази даних розглянути цей приклад не можна, оскільки у цій базі даних зберігаються дані лише про закупівлі продукції. Звичайно, будь-яке підприємство не може здійснювати лише закупівлі, бо закуплена продукція має або перероблюватися, або продаватися. Для бази даних, що розглядається, це було зроблено з метою спрощення, оскільки ця база даних є навчальною. Однак для практично будь-якого бізнесу є актуальною задача контролю руху ресурсів, зокрема задача контролю відповідності вхідних та вихідних матеріальних потоків, зокрема відповідність кількості закуплених та проданих товарів.

На рисунку 5.30 наведено модель даних, яка у дуже спрощеному вигляді відображає структуру бази даних, у якій зберігаються дані про

закупівлі та продажі товарів. При цьому для спрощення дані про постачальників та покупців не зберігаються.

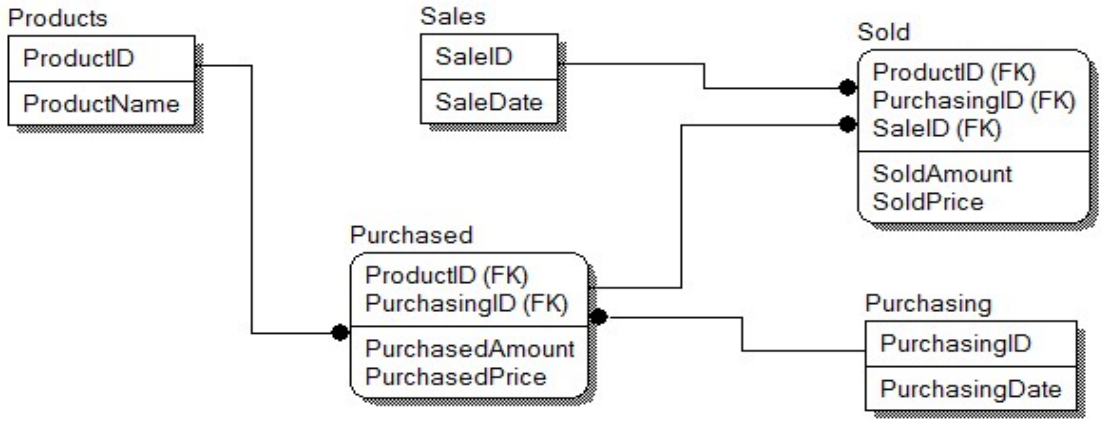



Рисунок 5.30

Текст запиту, за допомогою якого буде створено таку базу даних та введено до ней мінімальну кількість даних, наведений на рисунку 5.31. Після успішного виконання запит можна зберегти, наприклад, з ім'ям salesdb_create.sql.

```
USE [master]
GO
IF  EXISTS (SELECT name FROM sys.databases WHERE name = N'salesdb')
DROP DATABASE [salesdb]
GO
USE [master]
GO
CREATE DATABASE [salesdb] ON  PRIMARY
( NAME = N'salesdb', FILENAME = N'D:\salesdb.mdf' , SIZE = 3072KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'salesdb_log', FILENAME = N'D:\salesdb_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [salesdb] SET COMPATIBILITY_LEVEL = 100
GO
USE salesdb
CREATE TABLE Products (ProductID int PRIMARY KEY, ProductName char(20) NOT NULL)
CREATE TABLE Purchasing (PurchasingID int PRIMARY KEY, PurchasingDate datetime)
CREATE TABLE Purchased (ProductID int FOREIGN KEY REFERENCES Products (ProductID),
                        PurchasingID int FOREIGN KEY REFERENCES Purchasing (PurchasingID),
                        PurchasedAmount decimal(4,0) NOT NULL CHECK (PurchasedAmount>0),
                        PurchasedPrice  decimal(8,2) NOT NULL CHECK (PurchasedPrice>0),
                        PRIMARY KEY (ProductID,PurchasingID))
CREATE TABLE Sales (SaleID int PRIMARY KEY, SaleDate datetime)
CREATE TABLE Sold (SaleID int FOREIGN KEY REFERENCES Sales (SaleID), ProductID int, PurchasingID int,
                   SoldAmount decimal(4,0) NOT NULL CHECK (SoldAmount>0),
                   SoldPrice  decimal(8,2) NOT NULL CHECK (SoldPrice>0),
                   PRIMARY KEY (ProductID,PurchasingID,SaleID),
                   FOREIGN KEY (ProductID,PurchasingID) REFERENCES Purchased(ProductID,PurchasingID))
INSERT INTO Products VALUES (1,'Product1'), (2,'Product2')
INSERT INTO Purchasing (PurchasingID) VALUES (1),(2)
INSERT INTO Purchased VALUES (1,1,10,111), (1,2,20,222), (2,1,30,333), (2,2,40,444)
INSERT INTO Sales (SaleID) VALUES (1),(2),(3),(4)
INSERT INTO Sold VALUES (1,1,1,5,121),(2,1,1,4,122),(3,1,1,1,123)
```

Рисунок 5.31

Після створення бази даних salesdb треба переключитися на неї та перевірити наявність даних у таблицях бази даних.

Діаграма даних, яку можна створити після створення бази даних, наведена на рисунку 5.32.

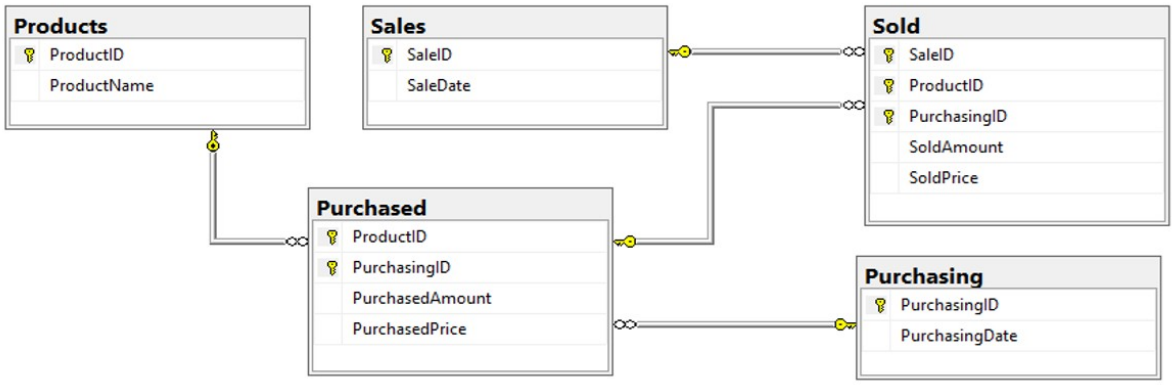



Рисунок 5.32

Як структурну особливість цієї бази даних можна визначити те, що в облікові дані про продану продукцію (таблиця «Sold») може потрапити лише продукція, яку перед цих було закуплено, що відображено у відповідних даних (таблиця «Purchased»). Це гарантується наявністю відповідного зв'язку між таблицями «Purchased» та «Sold». Однак виникає інша проблема, а саме контроль відповідності кількості закупленої та проданої продукції. Тобто кількість проданої продукції повинна дорівнювати кількості закупленої продукції, якщо усе, що було закуплено, продано, або бути менше кількості закупленої продукції, що відповідає ситуації наявності товарного запасу. Проконтролювати це за допомогою зв'язку неможливо.

Для вирішення цієї проблеми можна застосувати тригер. Текст запиту, за допомогою якого буде створено такий тригер, наведений на рисунку 5.33.

```
CREATE TRIGGER [dbo].[chk_sale_amount] ON [dbo].[Sold]
      AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @nom_prodagi int, @nom_postavki int, @kod_tovara int, @kol_vo int, @postavleno int, @prodano int
    SELECT @nom_postavki=PurchasingID, @kod_tovara=ProductID, @kol_vo=SoldAmount  FROM inserted
    PRINT 'new '+STR(@nom_postavki)+STR(@kod_tovara)+STR(@kol_vo)
    SELECT @postavleno=PurchasedAmount FROM Purchased WHERE PurchasingID=@nom_postavki AND ProductID=@kod_tovara
    PRINT 'PurchasedAmount '+STR(@postavleno)
    SELECT @prodano=SUM(SoldAmount) FROM Sold WHERE PurchasingID=@nom_postavki AND ProductID=@kod_tovara
    PRINT 'SoldAmount '+STR(@prodano)
    IF @prodano>@postavleno
       BEGIN
           PRINT STR(@postavleno)+' '+STR(@prodano)
           PRINT 'Перевищення товарного залишку'
           RAISERROR('***********',16,1)
           ROLLBACK
       END
END
```

Рисунок 5.33

Цей тригер призначений для контролю відповідності даних у таблицях «Purchased» та «Sold» при виконанні операції додавання або модифікації даних. Після успішного виконання запит можна зберегти, наприклад, з ім'ям salesdb_Trigger_chk_sale_amount.sql.

Розглянемо приклад ситуації контролю даних при виконанні операції додавання даних. Ілюстрація цього прикладу наведена на рисунку 5.34. Як видно з даних, які вже введено до таблиць бази даних, в рамках закупівлі з кодом 1 було закуплено 10 одиниць продукції з кодом 1. Потім, в рамках продаж з кодами 1, 2, 3, було продано загалом 10 одиниць цієї продукції, тобто цю продукцію було продано повністю. Після цього, в рамках продажу з кодом 4, робиться спроба продати ще 2 одиниці цієї продукції. При цьому спрацьовує тригер та блокує виконання цієї операції.

Розглянемо приклад ситуації контролю даних при виконанні операції модифікації даних. Ілюстрація цього прикладу наведена на рисунку 2.35. Як видно з наведених даних, робиться спроба змінити кількість вже проданої продукції, а саме замість 4 одиниць ввести 8. Але при цьому загальна продана кількість цієї продукції буде дорівнювати 14, що неможливо, оскільки було закуплено 10. При цьому також спрацьовує тригер та блокує виконання цієї операції.

| | ProductID | ProductName |
|---|---|---|
| 1 | 1 | Product1 |
| 2 | 2 | Product2 |

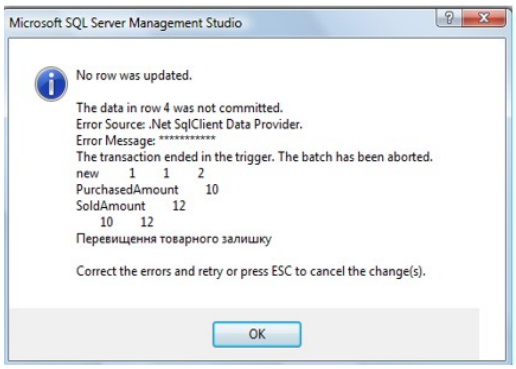
| | PurchasingID | PurchasingDate |
|---|---|---|
| 1 | 1 | NULL |
| 2 | 2 | NULL |

| | ProductID | PurchasingID | PurchasedAmount | PurchasedPrice |
|---|---|---|---|---|
| 1 | 1 | 1 | 10 | 111.00 |
| 2 | 1 | 2 | 20 | 222.00 |
| 3 | 2 | 1 | 30 | 333.00 |
| 4 | 2 | 2 | 40 | 444.00 |

| | SaleID | SaleDate |
|---|---|---|
| 1 | 1 | NULL |
| 2 | 2 | NULL |
| 3 | 3 | NULL |
| 4 | 4 | NULL |

| | SaleID | ProductID | PurchasingID | SoldAmount | SoldPrice |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 5 | 121.00 |
| 2 | 2 | 1 | 1 | 4 | 122.00 |
| 3 | 3 | 1 | 1 | 1 | 123.00 |

AFTER INSERT trigger




| | SaleID | ProductID | PurchasingID | SoldAmount | SoldPrice |
|---|---|---|---|---|---|
| | 1 | 1 | 1 | 5 | 121,00 |
| | 2 | 1 | 1 | 4 | 122,00 |
| | 3 | 1 | 1 | 1 | 123,00 |
| | 4 | 1 | 1 | 2 | 125 |
| * | NULL | NULL | NULL | NULL | NULL |

Рисунок 5.34

| | ProductID | ProductName |
|---|---|---|
| 1 | 1 | Product1 |
| 2 | 2 | Product2 |

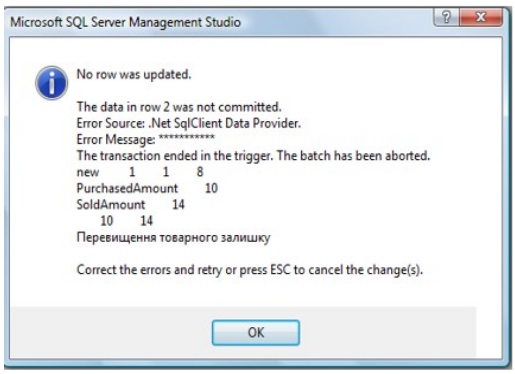
| | PurchasingID | PurchasingDate |
|---|---|---|
| 1 | 1 | NULL |
| 2 | 2 | NULL |

| | ProductID | PurchasingID | PurchasedAmount | PurchasedPrice |
|---|---|---|---|---|
| 1 | 1 | 1 | 10 | 111.00 |
| 2 | 1 | 2 | 20 | 222.00 |
| 3 | 2 | 1 | 30 | 333.00 |
| 4 | 2 | 2 | 40 | 444.00 |

| | SaleID | SaleDate |
|---|---|---|
| 1 | 1 | NULL |
| 2 | 2 | NULL |
| 3 | 3 | NULL |
| 4 | 4 | NULL |

| | SaleID | ProductID | PurchasingID | SoldAmount | SoldPrice |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 5 | 121.00 |
| 2 | 2 | 1 | 1 | 4 | 122.00 |
| 3 | 3 | 1 | 1 | 1 | 123.00 |

AFTER UPDATE trigger




| | SaleID | ProductID | PurchasingID | SoldAmount | SoldPrice |
|---|---|---|---|---|---|
| | 1 | 1 | 1 | 5 | 121,00 |
| | 2 | 1 | 1 | 8 | 122,00 |
| | 3 | 1 | 1 | 1 | 123,00 |
| * | NULL | NULL | NULL | NULL | NULL |

Рисунок 5.35

Після перевірки правильності роботи тригера базу даних salesdb можна вилучити та повернутися до тієї бази даних, з якою працювали раніше.

**5.2.4.2 Побудова та використання тригеру DDL**

Розглянемо задачу створення тригера DDL, який забороняє виконання структурних змін у базі даних. У даному прикладі це стосується заборони структурних операцій з таблицями, тобто заборону створення нових таблиць, видалення існуючих таблиць та зміни структури існуючих таблиць.

Для створення нового тригера DDL виконаємо такі дії.

1. Клацнути правою кнопкою миші по пункту Database Triggers і в меню вибрати пункт New Database Trigger…. В результаті буде створено запит, який містить «заготівку» тригера. Цю «заготівку» слід змінити, шляхом введення тексту тригера, який наведено рисунку 5.36. Після успішного виконання запит можна зберегти, наприклад, з ім'ям SQLQuery_DDL_trigger.sql.

```
USE delivery

IF EXISTS(SELECT 1 FROM sys.triggers WHERE parent_class_desc='DATABASE' AND name='tr1')
   DROP TRIGGER tr1 ON DATABASE
GO
CREATE TRIGGER tr1 ON DATABASE FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE AS
     SELECT EventData()
     RAISERROR('Усі операції щодо таблиць тимчасово заборонені. Адміністрація',16,1)
     ROLLBACK
GO
```

Рисунок 5.36

2. Для створення цього тригера потрібно натиснути кнопку Execute на панелі інструментів. У разі, якщо запит виконано успішно, у вікні Messages з'явиться повідомлення Command(s) successfully completed. У цьому випадку вікно запиту з текстом тригера можна закрити, причому запит зберігати як файл не потрібно. Тригер повинен з'явитися у списку тригерів (рисунок 5.37). Якщо тригер відсутній у списку, потрібно

клацнути правою кнопкою миші по пункту Database Triggers і в меню вибрати пункт Refresh.

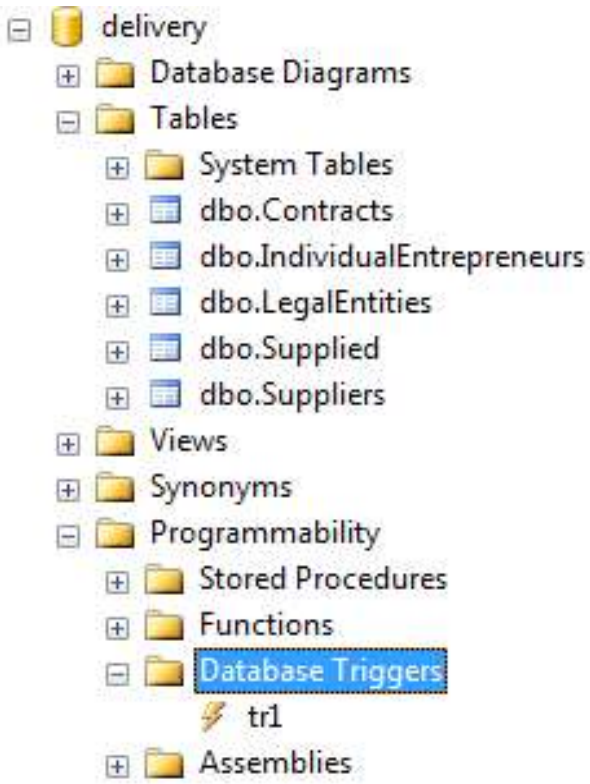



Рисунок 5.37

3. Для перевірки роботи цього тригера потрібно спробувати виконати структурні зміни у базі даних. Це можна зробити, наприклад, шляхом створення та виконання запиту, за допомогою якого буде створено нову таблицю. Цей запит наведений на рисунку 5.38. Спроба його виконання призведе до заборони такої операції. Користувач отримає повідомлення, яке наведено на рисунку 5.38.

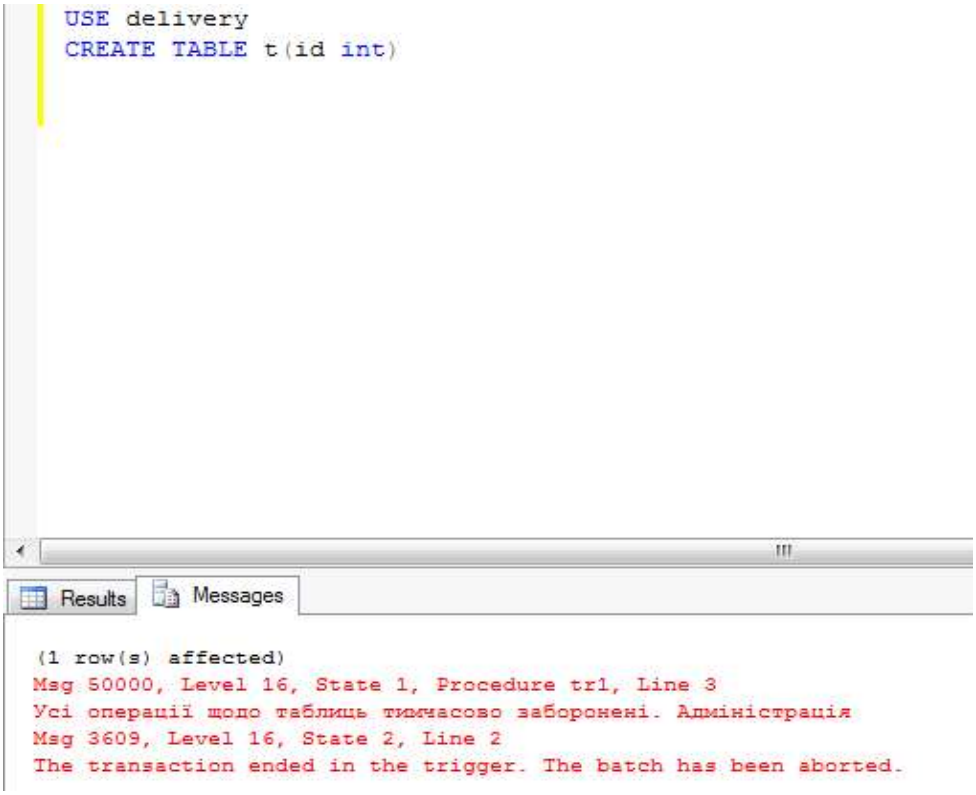



Рисунок 5.38

Наведені вище приклади тригерів є досить простими. Більш детальну інформацію щодо побудови та використання тригерів засобами мови Transact-SQL, можна, наприклад, отримати за посиланнями:

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-trigger-transact-sql?view=sql-server-ver16

### 5.2.5 Побудова та використання курсорів

Розглянемо приклад процедури створення та використання курсора, за допомогою якого можна продивлятися та модифікувати дані у певній таблиці бази даних. Цю процедуру реалізуємо у вигляді запиту. Текст запиту наведений рисунку 5.39.

За допомогою цього запиту буде створено курсор, до якого потраплять усі записи з таблиці «Supplied», у яких назва товару

починається з літери «м». Також за допомогою цього курсору кількість усіх товарі, назви яких починаються зі сполучення літер «ма», буде збільшена на 10 одиниць.

```
USE delivery
DECLARE @nom_dgvr int, @nazv_tov char(20), @kol_vo decimal(4,0), @cena decimal(8,2), @message varchar(80)
DECLARE tovar_cursor CURSOR GLOBAL SCROLL KEYSET FOR
    SELECT ContractNumber, Product, Amount, PricePerItem FROM Supplied WHERE SUBSTRING(Product,1,1)='м'
        ORDER BY ContractNumber, Product FOR UPDATE
OPEN tovar_cursor
FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
WHILE @@FETCH_STATUS=0
   BEGIN
      SELECT @message='Договір: '+STR(@nom_dgvr,5)+' Товар: '+@nazv_tov+' Кількість: '+STR(@kol_vo,4)+' Ціна:'+LTRIM(STR(@cena,8,2))
      PRINT @message
      IF @nazv_tov LIKE 'ма%'
         UPDATE Supplied SET Amount=Amount+10 WHERE CURRENT OF tovar_cursor
      FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
   END
CLOSE tovar_cursor
DEALLOCATE tovar_cursor
PRINT '--------'
DECLARE tovar_cursor CURSOR SCROLL KEYSET FOR
    SELECT ContractNumber, Product, Amount, PricePerItem FROM Supplied WHERE SUBSTRING(Product,1,1)='м'
        ORDER BY ContractNumber, Product
OPEN tovar_cursor
FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
WHILE @@FETCH_STATUS=0
   BEGIN
      SELECT @message='Договір: '+STR(@nom_dgvr,5)+' Товар: '+@nazv_tov+' Кількість: '+STR(@kol_vo,4)+' Ціна:'+LTRIM(STR(@cena,8,2))
      PRINT @message
      FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
   END
CLOSE tovar_cursor
DEALLOCATE tovar_cursor
```

Рисунок 5.39

Результат виконання запиту і, відповідно, роботи курсора, наведений рисунку 5.40. Слід звернути увагу на збільшення кількості одиниць товару. Також треба проконтролювати наявність зміни даних безпосередньо у таблиці «Supplied».

У наведеному прикладі курсор створюється два рази, причому з різними властивостями. Повторне створення курсору у даному випадку зроблено з метою лише перегляду змінених даних. Це не є обов'язковим і зроблено виключно із демонстраційною метою.

Після успішного виконання і пересвідчення у правильності роботи, запит можна зберегти, наприклад, з ім'ям SQLQuery_cursor1.sql.

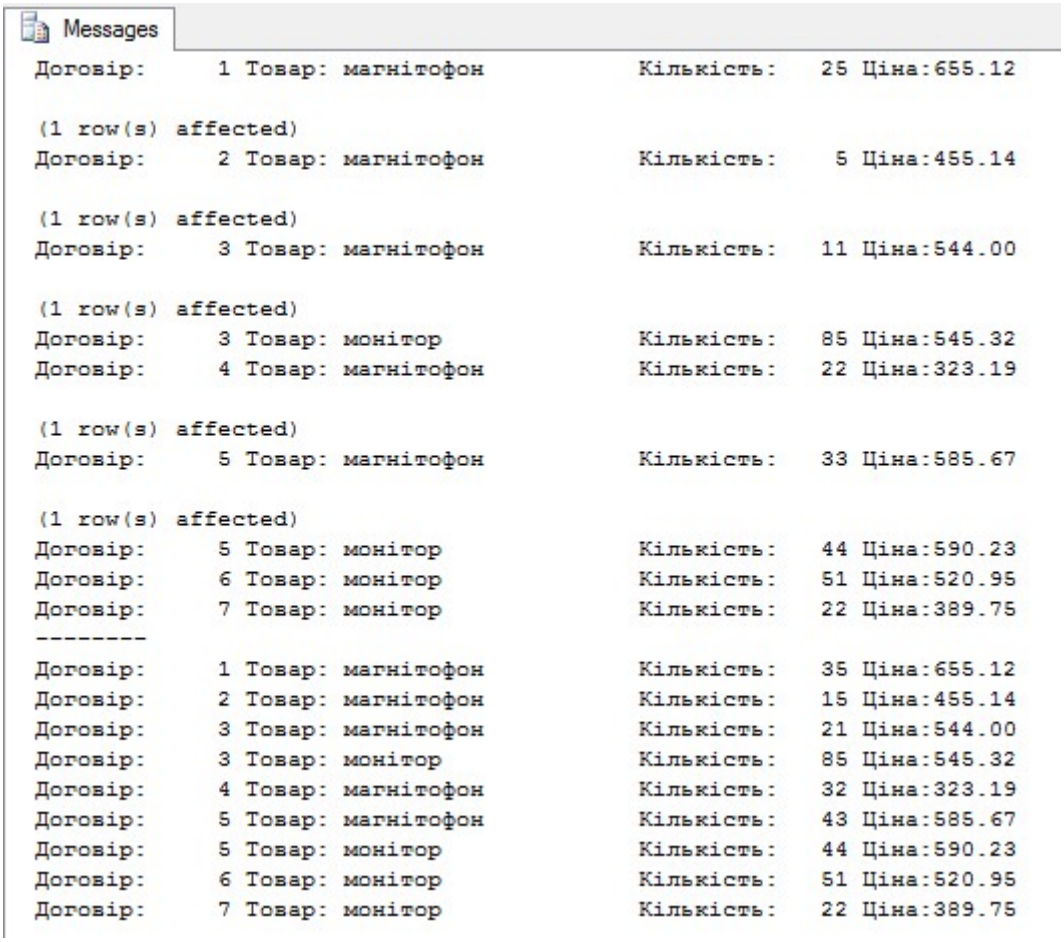



Рисунок 5.40

Повторне створення курсору може потребувати додаткових обчислювальних ресурсів. Тому розглянутий вище приклад можна спростити. Текст зміненого запиту наведений на рисунку 5.41. Результат виконання цього запиту аналогічний попередньому.

Після успішного виконання і пересвідчення у правильності роботи, запит можна зберегти, наприклад, з ім'ям SQLQuery_cursor2.sql.

```
USE delivery
DECLARE @nom_dgvr int, @nazv_tov char(20), @kol_vo decimal(4,0), @cena decimal(8,2), @message varchar(80)
DECLARE tovar_cursor CURSOR GLOBAL SCROLL KEYSET FOR
    SELECT ContractNumber, Product, Amount, PricePerItem FROM Supplied WHERE SUBSTRING(Product,1,1)='м'
        ORDER BY ContractNumber, Product FOR UPDATE
OPEN tovar_cursor
FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
WHILE @@FETCH_STATUS=0
   BEGIN
      SELECT @message='Договір: '+STR(@nom_dgvr,5)+' Товар: '+@nazv_tov+' Кількість: '+STR(@kol_vo,4)+' Ціна:'+LTRIM(STR(@cena,8,2))
      PRINT @message
      IF @nazv_tov LIKE 'ма%'
         UPDATE Supplied SET Amount=Amount+10 WHERE CURRENT OF tovar_cursor
      FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
   END
PRINT '--------'
DECLARE tovar_cursor CURSOR SCROLL KEYSET FOR
    SELECT ContractNumber, Product, Amount, PricePerItem FROM Supplied WHERE SUBSTRING(Product,1,1)='м'
        ORDER BY ContractNumber, Product
OPEN tovar_cursor
FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
WHILE @@FETCH_STATUS=0
   BEGIN
      SELECT @message='Договір: '+STR(@nom_dgvr,5)+' Товар: '+@nazv_tov+' Кількість: '+STR(@kol_vo,4)+' Ціна:'+LTRIM(STR(@cena,8,2))
      PRINT @message
      FETCH NEXT FROM tovar_cursor INTO @nom_dgvr,@nazv_tov,@kol_vo,@cena
   END
CLOSE tovar_cursor
DEALLOCATE tovar_cursor
```

Рисунок 5.41

Наведені вище приклади курсорів є досить простими. Більш детальну інформацію щодо побудови та використання курсорів засобами мови Transact-SQL, можна, наприклад, отримати за посиланнями:

https://learn.microsoft.com/en-us/sql/t-sql/language-elements/cursors-transact-sql?view=sql-server-ver16

https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-cursor/

### 5.2.6 Звітність про виконання практичних завдань теми 5

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) для кожного з реалізованих програмних об'єктів навести призначення, текст запиту для створення та результат використання програмного об'єкту;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 5.3 Питання для самоперевірки по темі 5

1. Створювані користувачем функції. Загальна характеристика, призначення та використання.

2. Вбудовані функції. Загальна характеристика, призначення та використання.

3. Класифікація функцій користувача.

4. Скалярні функції. Загальна характеристика, призначення та використання.

5. Табличні функції. Загальна характеристика, призначення та використання.

6. Багатооператорні табличні функції. Загальна характеристика, призначення та використання.

7. Команди SQL для роботи з функціями користувача.

8. Вбудовані математичні функції. Загальна характеристика, призначення та використання.

9. Вбудовані строкові функції. Загальна характеристика, призначення та використання.

10. Вбудовані функції для роботи з датою і часом. Загальна характеристика, призначення та використання.

11. Збережені процедури. Загальна характеристика, призначення та використання.

12. Класифікація збережених процедур.

13. Основні завдання при створенні збереженої процедури.

14. Команди SQL для роботи зі збереженими процедурами.

15. Параметри збережених процедур.

16. Тригери. Загальна характеристика, призначення та використання.

17. Тригери DML та DDL. Різниця у призначенні і використанні.

18. Команди SQL для роботи з тригерами.

19. Тригери INSTEAD OF. Загальна характеристика, призначення та використання.

20. Курсори. Загальна характеристика, призначення та використання.

21. Класифікація курсорів.

22. Команди SQL для роботи з курсорами.

23. Призначення функції @@FETCH_STATUS.

# ТЕМА 6
# СТВОРЕННЯ ТА ВИКОРИСТАННЯ ПРЕДСТАВЛЕНЬ (VIEW) ЗАСОБАМИ СУБД MICROSOFT SQL SERVER

## 6.1 Теоретичні відомості

### 6.1.1 Поняття представлення

Представлення, або уявлення, або перегляди (VIEW), є тимчасовими, похідними (інакше – віртуальні) таблицями і є об'єктами бази даних, інформація в яких не зберігається постійно, як у базових таблицях, а формується динамічно при зверненні до них. Звичайні таблиці відносяться до базових, тобто що містить дані і постійно знаходиться на пристрої зберігання інформації. Представлення не може існувати само по собі, а визначається тільки в термінах однієї або декількох таблиць. Застосування представлень дозволяє розробникові бази даних забезпечити кожному користувачеві або групі користувачів найбільш відповідні способи роботи з даними, що вирішує проблему простоти їх використання і безпеки. Вміст представлень вибирається з інших таблиць за допомогою виконання запиту, причому при зміні значень в таблицях дані в представленні автоматично міняються. Представлення – це фактично той же запит, який виконується всякий раз при участі в якій-небудь команді. Результат виконання цього запиту в кожен момент часу стає змістом представлення. У користувача створюється враження, що він працює із справжньою, реально існуючою таблицею.

У СУБД є дві можливості реалізації представлень. Якщо його визначення просте, то система формує кожен запис представлення в міру необхідності, поступово прочитуючи початкові дані з базових таблиць. У разі складного визначення СУБД доводиться спочатку виконати таку операцію, як матеріалізація представлення, тобто зберегти інформацію, з якої складається представлення, в тимчасовій таблиці. Потім система приступає до виконання призначеної для користувача команди і формування її результатів, після чого тимчасова таблиця видаляється.

Представлення – це зумовлений запит, що зберігається у базі даних, який виглядає подібно до звичайної таблиці і не вимагає для свого зберігання дискової пам'яті. Для зберігання представлення використовується тільки оперативна пам'ять. На відміну від інших об'єктів бази даних представлення не займає дискової пам'яті за винятком пам'яті, необхідної для зберігання визначення самого представлення.

Створення і зміни представлень в стандарті мови і реалізації в MS SQL Server співпадають і представлені наступною командою:

```
<визначення_представлення> ::=
    { CREATE| ALTER} VIEW ім'я_представлення
    [(ім'я_стовпця [,..n])]
    [WITH ENCRYPTION]
    AS SELECT_оператор
    [WITH CHECK OPTION]
```

Розглянемо призначення основних параметрів.

За умовчанням імена стовпців в представленні відповідають іменам стовпців в початкових таблицях. Явну вказівку імені стовпця потрібно для обчислюваних стовпців або при об'єднанні декількох таблиць, що мають стовпці з однаковими іменами. Імена стовпців перераховуються через кому, відповідно до порядку їх дотримання в представленні.

Параметр WITH ENCRYPTION наказує серверу шифрувати SQL-код запиту, що гарантує неможливість його несанкціонованого перегляду і використання. Якщо при визначенні представлення необхідно приховати імена початкових таблиць і стовпців, а також алгоритм об'єднання даних, необхідно застосувати цей аргумент.

Параметр WITH CHECK OPTION наказує серверу виконувати перевірку змін, вироблюваних через представлення, на відповідність критеріям, визначеним в операторові SELECT. Це означає, що не допускається виконання змін, які приведуть до зникнення рядка з

представлення. Таке трапляється, якщо для представлення встановлений горизонтальний фільтр і зміна даних призводить до невідповідності рядка встановленим фільтрам. Використання аргументу WITH CHECK OPTION гарантує, що зроблені зміни будуть відображені в представленні. Якщо користувач намагається виконати зміни, що призводять до виключення рядка із представлення, при заданому аргументі WITH CHECK OPTION сервером буде сформовано повідомлення про помилку і усі зміни будуть відхилені.

Представлення можна використовувати в команді SQL так само, як і будь-яку іншу таблицю. До представлення можна будувати запит, модифікувати його (якщо воно відповідає певним вимогам), сполучати з іншими таблицями. Зміст представлення не фіксований і оновлюється кожного разу, коли на нього посилаються в команді. Представлення значно розширюють можливості управління даними. Зокрема, це прекрасний спосіб дозволити доступ до інформації в таблиці, приховавши частину даних.

Таким чином, представлення може змінюватися командами модифікації DML, але фактично модифікація впливає не на само представлення, а на базову таблицю.

Представлення видаляється командою:

DROP VIEW ім'я_представлення [,..n]

### 6.1.2 Оновлення даних в представленнях

Не усі представлення в SQL можуть бути модифіковані, тобто використані для модифікації даних, які в них містяться. Представлення, що модифікується, визначається наступними критеріями:

- ґрунтується тільки на одній базовій таблиці;
- містить первинний ключ цієї таблиці;
- не містить DISTINCT у своєму визначенні;
- не використовує GROUP BY або HAVING у своєму визначенні;

- по можливості не застосовує у своєму визначенні підзапити;
- не використовує константи або вираження значень серед вибраних полів виводу;
- у представлення має бути включений кожен стовпець таблиці, що має атрибут NOT NULL;
- оператор SELECT перегляду не використовує агрегуючі (підсумкові) функції, з'єднання таблиць, Збережені процедури, і функції, визначені користувачем;
- ґрунтується на поодинокому запиті, тому об'єднання (UNION) не дозволене.

Якщо представлення задовольняє цим умовам, до нього можуть застосовуватися оператори INSERT, UPDATE, DELETE. Відмінності між представленнями, що модифікуються, і представленнями, призначеними тільки для читання, не випадкові. Цілі, для яких їх використовують, різні. З представленнями, що модифікуються, в основному обходяться точно так, як і з базовими таблицями. Фактично, користувачі не можуть навіть усвідомити, чи являється об'єкт, з яким вони працюють, базовою таблицею або представленням, тобто передусім це засіб захисту для приховання конфіденційних частин таблиці або таких частин таблиці, що не відносяться до потреб цього користувача. Представлення в режимі «тільки для читання» дозволяють отримувати і форматувати дані раціональніше. Вони створюють цілий арсенал складних запитів, які можна виконати і повторити знову, зберігаючи отриману інформацію. Результати цих запитів можуть потім використовуватися в інших запитах, що дозволить уникнути складних предикатів і понизити вірогідність помилкових дій.

### 6.1.3 Переваги і недоліки представлень

Механізм представлення – потужний засіб СУБД, що дозволяє приховати реальну структуру бази даних від деяких користувачів шляхом визначення представлень. Будь-яка реалізація представлення повинна гарантувати, що стан відношення, що представляється, точно відповідає

стану даних, на яких визначено це представлення. Звичайне обчислення представлення робиться кожного разу при його використанні. Коли представлення створюється, інформація про нього записується в каталог БД під власним ім'ям. Будь-які зміни в даних адекватно відобразяться в представленні - в цьому його відмінність від дуже схожого на нього запиту до БД. В той же час запит є як би «миттєвою фотографією» даних і при зміні останніх запит до БД необхідно повторити. Наявність представлень у БД потрібна для забезпечення логічної незалежності даних. Якщо система забезпечує фізичну незалежність даних, то зміни у фізичній структурі БД не впливають на роботу призначених для користувача програм. Логічна незалежність має на увазі той факт, що при зміні логічної структури даних вплив на призначені для користувача програми також не виявляється, а значить, система повинна уміти вирішувати проблеми, пов'язані з ростом і реструктуризацією БД. Очевидно, що зі збільшенням кількості даних, що зберігаються у БД, виникає необхідність її розширення за рахунок додавання нових атрибутів або стосунків – це називається ростом БД. Реструктуризація даних має на увазі збереження тієї ж самої інформації, але змінюється її розташування, наприклад, за рахунок перегрупування атрибутів в стосунках. Припустимо, деяке відношення через які-небудь причини необхідно розділити на два. З'єднання отриманих стосунків в представленні відтворює початкове відношення, а у користувача складається враження, що ніякої реструктуризації не робилася. Окрім вирішення проблеми реструктуризації представлення можна застосовувати для перегляду одних і тих же даних різними користувачами і в різних варіантах. За допомогою представлень користувач має можливість обмежити об'єм даних для зручності роботи. Нарешті, механізм уявлень дозволяє приховати службові дані, не цікаві користувачам.

У разі виконання СУБД на персональному комп'ютері, що працює автономно, використання представлень зазвичай має на меті лише спрощення структури запитів до бази даних. Проте у разі розрахованої на багато користувачів мережевої СУБД представлення грають ключову роль

у визначенні структури бази даних і організації захисту інформації. Розглянемо основні **переваги** застосування представлень саме у такому середовищі.

1. Незалежність від даних.

За допомогою представлень можна створити погоджену, незмінну картину структури бази даних, яка залишатиметься стабільною навіть у разі зміни формату початкових таблиць (наприклад, додавання або видалення стовпців, зміни зв'язків, розподілу таблиць, їх реструктуризації або перейменування). Якщо в таблицю додаються або з неї видаляються не використовувані в представленні стовпці, то змінювати визначення цього представлення буде не потрібно. Якщо структура початкової таблиці перевпорядковується або таблиця розділяється, можна створити представлення, що дозволяє працювати з віртуальною таблицею колишнього формату. У разі розподілу початкової таблиці, колишній формат може бути віртуально відтворений за допомогою представлення, побудованого на основі з'єднання знову створених таблиць, - звичайно, якщо це виявиться можливим. Останню умову можна забезпечити за допомогою приміщення в усі знову створені таблиці первинного ключа колишньої таблиці.

2. Актуальність.

Зміни даних у будь-якій з таблиць бази даних, вказаних у визначальному запиті, негайно відображається на вмісті представлення.

3. Підвищення захищеності даних.

Права доступу до даних можуть бути надані виключно через обмежений набір представлень, що містять тільки ту підмножину даних, яка потрібна користувачеві. Подібний підхід дозволяє істотно посилити контроль за доступом окремих категорій користувачів до інформації у базі даних.

4. Зниження складності.

Представлення дозволяють спростити структуру запитів за рахунок об'єднання даних з декількох таблиць в єдину віртуальну таблицю. В

результаті багатотабличні запити зводяться до простих запитів до одного представлення.

5. Додаткові зручності.

Створення представлень може забезпечувати користувачів додатковими зручностями – наприклад, можливістю роботи тільки з дійсно потрібною частиною даних. В результаті можна добитися максимального спрощення тієї моделі даних, яка знадобиться кожному кінцевому користувачеві.

6. Можливість налаштування.

Представлення є зручним засобом налаштування індивідуального образу бази даних. В результаті одні і ті ж таблиці можуть бути пред'явлені користувачам в абсолютно різному вигляді.

7. Забезпечення цілісності даних.

Якщо в операторові CREATE VIEW буде вказана фраза WITH CHECK OPTION, то СУБД стане здійснювати контроль за тим, щоб в початкові таблиці бази даних не був введений жоден з рядків, що не задовольняють пропозиції WHERE у визначальному запиті. Цей механізм гарантує цілісність даних в представленні.

Практика обмеження доступу деяких користувачів до даних за допомогою створення спеціалізованих представлень, безумовно, має значні переваги перед наданням їм прямого доступу до таблиць бази даних.

Проте використання представлень в середовищі SQL не позбавлене **недоліків**.

1. Обмежені можливості оновлення.

В деяких випадках представлення не дозволяють вносити зміни в дані, що містяться в них.

2. Структурні обмеження.

Структура представлення встановлюється у момент його створення. Якщо визначальний запит представлений у формі SELECT * FROM , то

символ * посилається на усі стовпці, існуючі в вихідній таблиці на момент створення представлення. Якщо згодом в вихідну таблицю бази даних додадуться нові стовпці, то вони не з'являться в цьому представленні до тих пір, поки це представлення не буде видалено і знову створене.

3. Зниження продуктивності.

Використання представлень пов'язане з певним зниженням продуктивності. У одних випадках вплив цього чинника абсолютно трохи, тоді як в інших воно може послужити джерелом істотних проблем. Наприклад, представлення, визначене за допомогою складного багататабличного запиту, може зажадати значних витрат часу на обробку, оскільки при його формуванні потрібно буде виконувати з'єднання таблиць всякий раз, коли знадобиться доступ до даного представлення. Виконання формування представлень пов'язане з використанням додаткових обчислювальних ресурсів.

## 6.2 Практичне опанування теми 6

### 6.2.1 Передумови виконання завдань теми 6

При виконанні практичних завдань теми 6 передбачається використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи Microsoft SQL Server. При цьому передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Для початку виконання треба підключити базу даних (delivery або dlvr), з якою працювали при виконанні при виконанні практичних завдань теми «Створення та використання програмних об'єктів бази даних засобами СУБД Microsoft SQL Server».

Увага! Результати використання програмних об'єктів, що розглядаються далі, повинні бути результативними (тобто в результаті виконання запиту повинні бути виведені один або кілька записів). Відсутність результату запиту є ознакою помилок під час побудови запиту, невідповідності запиту наявним даним тощо. Такий запит потребує аналізу та перевірки. Також передбачається, що результати запитів ґрунтуються на тих даних, які були введені до бази даних при виконанні практичних завдань теми «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

### 6.2.2 Побудова та використання представлення для перегляду даних

Як відомо, при нормалізації відношень виконується декомпозиція відношення, яке потребує нормалізації. При цьому замість одного вихідного відношення створюється два або більше відношень, які зв'язуються між собою за допомогою первинних та зовнішніх ключів. Нормалізовані відношення не мають таких недоліків, як, наприклад, аномалії оновлення, але їх зміст для кінцевого користувача не є зручним для сприйняття. Наприклад, відношення (тобто) таблиця «Contracts» містить посилання на таблицю «Suppliers» у вигляді коду постачальника. Уся інша інформація про постачальників зберігається у відповідних окремих таблицях. Припустимо, що у користувача цієї бази даних виникла потреба при перегляді списку договорів бачити більш докладну інформацію щодо постачальників, а саме назву постачальника, прізвище, ім'я та по батькові для фізичних осіб або податковий номер для юридичних осіб. Для вирішення цієї задачі можна застосувати представлення.

Для створення представлення слід виконати таку послідовність дій.

1. Клацнути правою кнопкою миші по пункту Views і в меню вибрати пункт New View…

2. У списку таблиць вибрати таблиці «Contracts», «Suppliers», «IndividualEntrepreneurs», «LegalEntities». Список таблиць закрити. В

результаті з'явиться графічне зображення таблиць, що використовуються як джерело даних для представлення, та зв'язків між ними (рисунок 6.1)

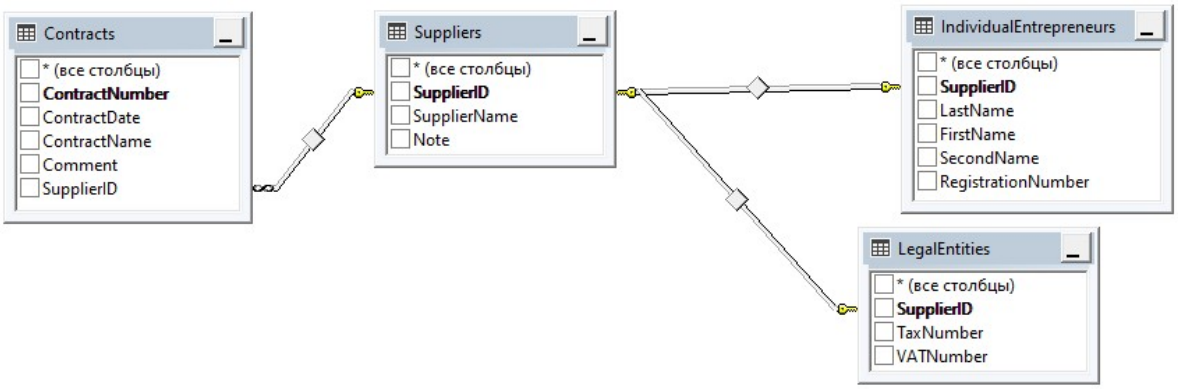



Рисунок 6.1

3. Клацнути правою кнопкою миші по зв'язку між таблицями «Suppliers» та «IndividualEntrepreneurs» та вибрати пункт Select All Rows from Suppliers. Клацнути правою кнопкою миші по зв'язку між таблицями «Suppliers» та «LegalEntities» та вибрати пункт Select All Rows from Suppliers. В результаті зв'язки набудуть вигляду, наведеного на рисунку 6.2.

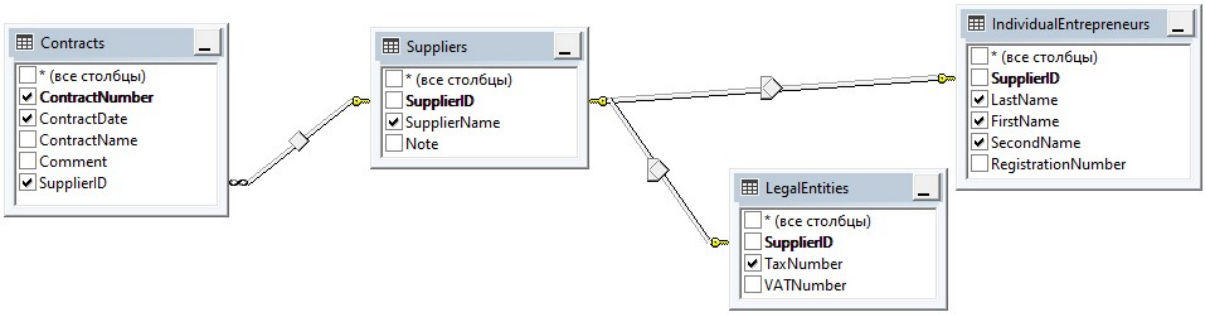



Рисунок 6.2

4. Вибрати поля таблиць, які включаються до результату запиту, поставивши позначки для відповідних полів (рисунок 6.2). У результаті текст запиту представлення матиме вигляд, наведений рисунку 6.3. Натиснувши на панелі інструментів кнопку Execute SQL, можна отримати

результат запиту. Цей результат має певний недолік – дані постачальників як суб'єктів господарювання (тобто, як юридичних та фізичних осіб) перебувають у різних полях (рисунок 6.4). Цей недолік можна виправити, змінивши текст запиту (рисунок 6.5).

```
SELECT     dbo.Contracts.ContractNumber, dbo.Contracts.ContractDate, dbo.Contracts.SupplierID, dbo.Suppliers.SupplierName, dbo.LegalEntities.TaxNumber,
           dbo.IndividualEntrepreneurs.LastName, dbo.IndividualEntrepreneurs.FirstName, dbo.IndividualEntrepreneurs.SecondName
FROM       dbo.Contracts INNER JOIN
             dbo.Suppliers ON dbo.Contracts.SupplierID = dbo.Suppliers.SupplierID LEFT OUTER JOIN
             dbo.LegalEntities ON dbo.Suppliers.SupplierID = dbo.LegalEntities.SupplierID LEFT OUTER JOIN
             dbo.IndividualEntrepreneurs ON dbo.Suppliers.SupplierID = dbo.IndividualEntrepreneurs.SupplierID
```

Рисунок 6.3

| | ContractNum... | ContractDate | SupplierID | SupplierName | TaxNumber | LastName | FirstName | SecondName |
|---|---|---|---|---|---|---|---|---|
| ▶ | 1 | 1999-09-01 00:0... | 1 | ПП Іваненко І.І | *NULL* | Іваненко | Ілля | Іванович |
| | 2 | 1999-09-10 00:0... | 1 | ПП Іваненко І.І | *NULL* | Іваненко | Ілля | Іванович |
| | 3 | 1999-09-10 00:0... | 3 | ПП Петренко ... | *NULL* | Петренко | Павло | Петрович |
| | 4 | 1999-09-23 00:0... | 3 | ПП Петренко ... | *NULL* | Петренко | Павло | Петрович |
| | 5 | 1999-09-24 00:0... | 2 | ТОВ «Інтерфрут» | 00123987 | *NULL* | *NULL* | *NULL* |
| | 6 | 1999-10-01 00:0... | 1 | ПП Іваненко І.І | *NULL* | Іваненко | Ілля | Іванович |
| | 7 | 1999-10-02 00:0... | 2 | ТОВ «Інтерфрут» | 00123987 | *NULL* | *NULL* | *NULL* |

Рисунок 6.4

```
SELECT     dbo.Contracts.ContractNumber, dbo.Contracts.ContractDate, dbo.Contracts.SupplierID, dbo.Suppliers.SupplierName,
           ISNULL(dbo.LegalEntities.TaxNumber + SPACE(30), RTRIM(dbo.IndividualEntrepreneurs.LastName)
             + ' ' + RTRIM(dbo.IndividualEntrepreneurs.FirstName) + ' ' + RTRIM(dbo.IndividualEntrepreneurs.SecondName)) AS Supplier
FROM       dbo.Contracts INNER JOIN
             dbo.Suppliers ON dbo.Contracts.SupplierID = dbo.Suppliers.SupplierID LEFT OUTER JOIN
             dbo.LegalEntities ON dbo.Suppliers.SupplierID = dbo.LegalEntities.SupplierID LEFT OUTER JOIN
             dbo.IndividualEntrepreneurs ON dbo.Suppliers.SupplierID = dbo.IndividualEntrepreneurs.SupplierID
```

Рисунок 6.5

5. Змінений результат виконання запиту наведений на рисунку 6.6. Можна вважати, що цей результат вже задовольняє умовам, які висувалися для перегляду списку договорів. Зверніть увагу на появу у тексті запиту функцій ISNULL, SPACE, RTRIM.

| | ContractNum... | ContractDate | SupplierID | SupplierName | Supplier |
|---|---|---|---|---|---|
| ▶ | 1 | 1999-09-01 00:0... | 1 | ПП Іваненко І.І | Іваненко Ілля Іванович |
| | 2 | 1999-09-10 00:0... | 1 | ПП Іваненко І.І | Іваненко Ілля Іванович |
| | 3 | 1999-09-10 00:0... | 3 | ПП Петренко П.П | Петренко Павло Петрович |
| | 4 | 1999-09-23 00:0... | 3 | ПП Петренко П.П | Петренко Павло Петрович |
| | 5 | 1999-09-24 00:0... | 2 | ТОВ «Інтерфрут» | 00123987 |
| | 6 | 1999-10-01 00:0... | 1 | ПП Іваненко І.І | Іваненко Ілля Іванович |
| | 7 | 1999-10-02 00:0... | 2 | ТОВ «Інтерфрут» | 00123987 |

Рисунок 6.6

6. Зберегти представлення з ім'ям View_1.

7. Перевірити роботу представлення, для чого клацнути правою кнопкою миші на ім'я представлення і у меню вибрати команду Open View.

### 6.2.3 Оновлення даних за допомогою представлень

#### 6.2.3.1 Використання параметру WITH CHECK OPTION

Припустимо, що виникла потреба роботи з договорами, які було укладено з певним постачальником. Для отримання такого списку договорів можна застосувати представлення. Запит, за допомогою якого можна створити таке представлення, наведений на рисунку 6.7.

```
CREATE VIEW [dbo].[View_11]
AS
SELECT        ContractNumber, ContractDate, Supplie
FROM            dbo.Contracts
WHERE        (SupplierID = 3)
```

Рисунок 6.7

За допомогою такого представлення можна оновлювати дані. Спробуємо це зробити за допомогою запиту, який наведено на рисунку 6.8. Як видно з повідомлення, що наведено на рисунку 6.9, дані у жодному запису у результаті виконання цього запиту змінено не буде.

```
USE delivery
UPDATE View_11 SET SupplierID=2 WHERE Contra
```

Рисунок 6.8

```
Messages

(0 row(s) affected)
```

Рисунок 6.9

Наступний приклад оновлення даних за допомогою цього представлення наведено на рисунку 6.10. При спробі виконання цього запиту буде виведено повідомлення, яке наведено на рисунку 6.11. Поява цього повідомлення обумовлена саме застосуванням параметру WITH CHECK OPTION при визначенні представлення.

```
USE delivery
UPDATE View_11 SET SupplierID=2 WHERE Contra
```

Рисунок 6.10

```
Messages
Msg 550, Level 16, State 1, Line 2
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION
or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation
did not qualify under the CHECK OPTION constraint.
The statement has been terminated.
```

Рисунок 6.11

Для того, щоб з'ясувати, що зміниться у цьому представленні, якщо прибрати параметр WITH CHECK OPTION, змінимо попереднє представлення. Запит, за допомогою якого можна створити таке представлення, наведений на рисунку 6.12.

```
CREATE VIEW [dbo].[View_2]
AS
SELECT        ContractNumber, ContractDate, Supplie
FROM            dbo.Contracts
```

Рисунок 6.12

Перевіримо роботу цього представлення, застосувавши такі ж запити, як і для попереднього представлення. Результат першого запиту (рисунок 6.13) будем таким, як і у попередньому випадку (рисунок 6.14).

```
USE delivery
UPDATE View 2 SET SupplierID=2 WHERE Contrac
```

Рисунок 6.13

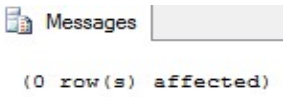



Рисунок 6.14

А у другому випадку (рисунок 6.15) оновлення даних, на відміну від попередньої спроби, буде результативним (рисунок 6.16). Також, перевіривши дані у таблиці «Contracts», можна побачити, що код постачальника у договорі 3 змінено з 3 на 2. Це стало можливим саме завдяки відсутності параметра WITH CHECK OPTION.

```
USE delivery
UPDATE View 2 SET SupplierID=2 WHERE Contrac
```

Рисунок 6.15


Рисунок 6.16

### 6.2.3.2 Використання тригеру INSTEAD OF

У базі даних, що розглядається, дані про постачальників зберігаються у трьох окремих таблицях. Тому, наприклад, при введенні даних про нового постачальника, який є фізичною особою, відповідні операції треба буде виконувати стосовно двох таблиць. Це може бути не дуже зручним, тому може виникнути задача розробки таких засобів роботи

з даними, застосування яких може зменшити обсяг роботи кінцевого користувача.

Для вирішення цієї задачі можна створити представлення, що оновлюється, для таблиць «Suppliers» та «IndividualEntrepreneurs». При додаванні в представлення нового запису про постачальника, що є фізичною особою (ФОП – фізична особа-підприємець або ПП – приватний підприємець), додані дані повинні бути записані і до таблиці «Suppliers», і до таблиці «IndividualEntrepreneurs». Таке представлення наведене на рисунку 6.17.

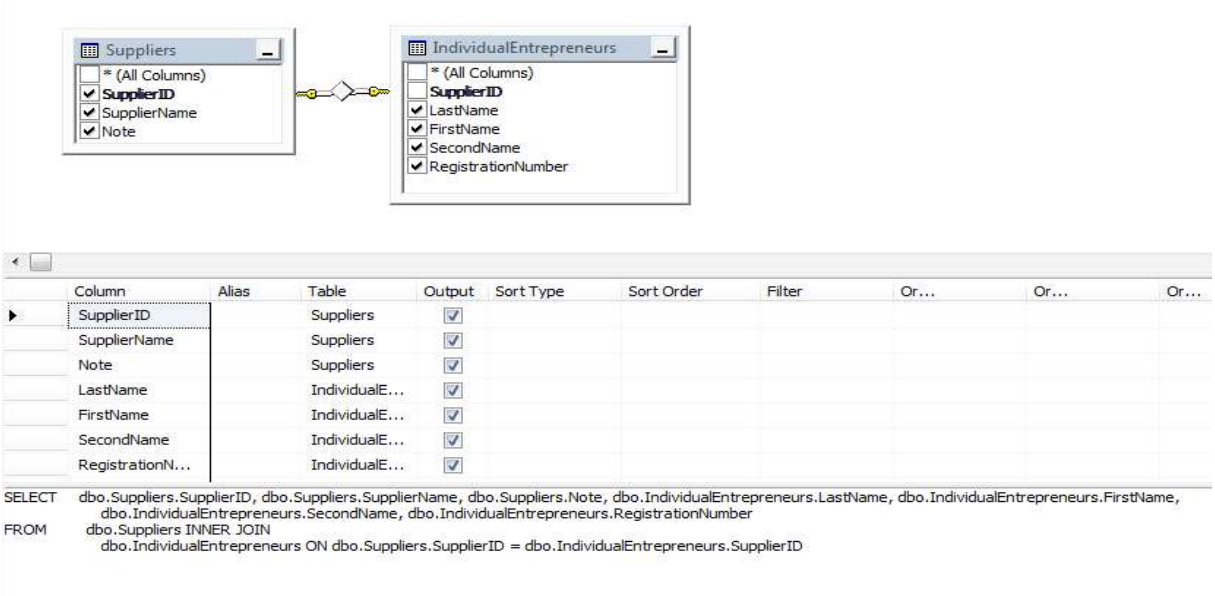



Рисунок 6.17

Стан даних у таблицях «Suppliers» та «IndividualEntrepreneurs» та дані, що сформовані за допомогою представлення, наведені на рисунку 6.18.

| | SupplierID | SupplierName | Note |
|---|---|---|---|
| | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12-34-56, факс 22-12-33) |
| | 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 |
| | 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 |
| | 4 | ЗАТ «Транссервіс» | м. Одеса, вул. Дерибасівська, 75 |
| | 5 | ПП Сидорчук М.С. | м. Полтава, вул. Свободи, 15, кв. 43 |
| ▸* | NULL | NULL | NULL |

| | SupplierID | LastName | FirstName | SecondName | RegistrationN... |
|---|---|---|---|---|---|
| ▸ | 1 | Іваненко | Ілля | Іванович | 00143987 |
| | 3 | Петренко | Павло | Петрович | 12345678 |
| | 5 | Сидорчук | Микита | Степанович ... | 09876541 |
| * | NULL | NULL | NULL | NULL | NULL |

| | SupplierID | SupplierName | Note | LastName | SecondName | FirstName | RegistrationN... |
|---|---|---|---|---|---|---|---|
| ▸ | 1 | ПП Іваненко І.І | м. Харків, вул. Пушкі... | Іваненко | Іванович | Ілля | 00143987 |
| | 3 | ПП Петренко ... | м. Харків, пр. Науки, ... | Петренко | Петрович | Павло | 12345678 |
| | 5 | ПП Сидорчук ... | м. Полтава, вул. Сво... | Сидорчук | Степанович ... | Микита | 09876541 |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Рисунок 6.18

Для цього представлення створимо тригер INSTEAD OF. Текст відповідного запиту наведений на рисунку 6.19.

```
CREATE TRIGGER [dbo].[Insert_IE] ON [dbo].[View_IE]
   INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @new_SupplierID int, @new_SupplierName char(20), @new_SuppliersNote char(120),
            @new_LastName char(20), @new_SecondName char(20), @new_FirstName char(20),
            @new_RegistrationNumber char(20)
    SELECT  @new_SupplierID=SupplierID, @new_SupplierName=SupplierName, @new_SuppliersNote=Note,
            @new_LastName=LastName, @new_SecondName=SecondName, @new_FirstName=FirstName,
            @new_RegistrationNumber=RegistrationNumber FROM inserted
    IF NOT @new_SupplierID IS NULL AND NOT @new_LastName IS NULL
       IF (NOT EXISTS (SELECT * FROM Suppliers WHERE SupplierID=@new_SupplierID))
          INSERT INTO Suppliers VALUES (@new_SupplierID, @new_SupplierName, @new_SuppliersNote)
          INSERT INTO IndividualEntrepreneurs VALUES (@new_SupplierID,@new_LastName, @new_SecondName,
                                                      @new_FirstName, @new_RegistrationNumber)
END
GO
```

Рисунок 6.19

Для перевірки роботи представлення та тригера, введемо тестові дані про нового постачальника, про якого відразу відомо, що він є фізичною особою. Приклад введення таких даних у представлення та їх поява у таблицях, які є джерелом даних для представлення, наведений на рисунку 6.20.

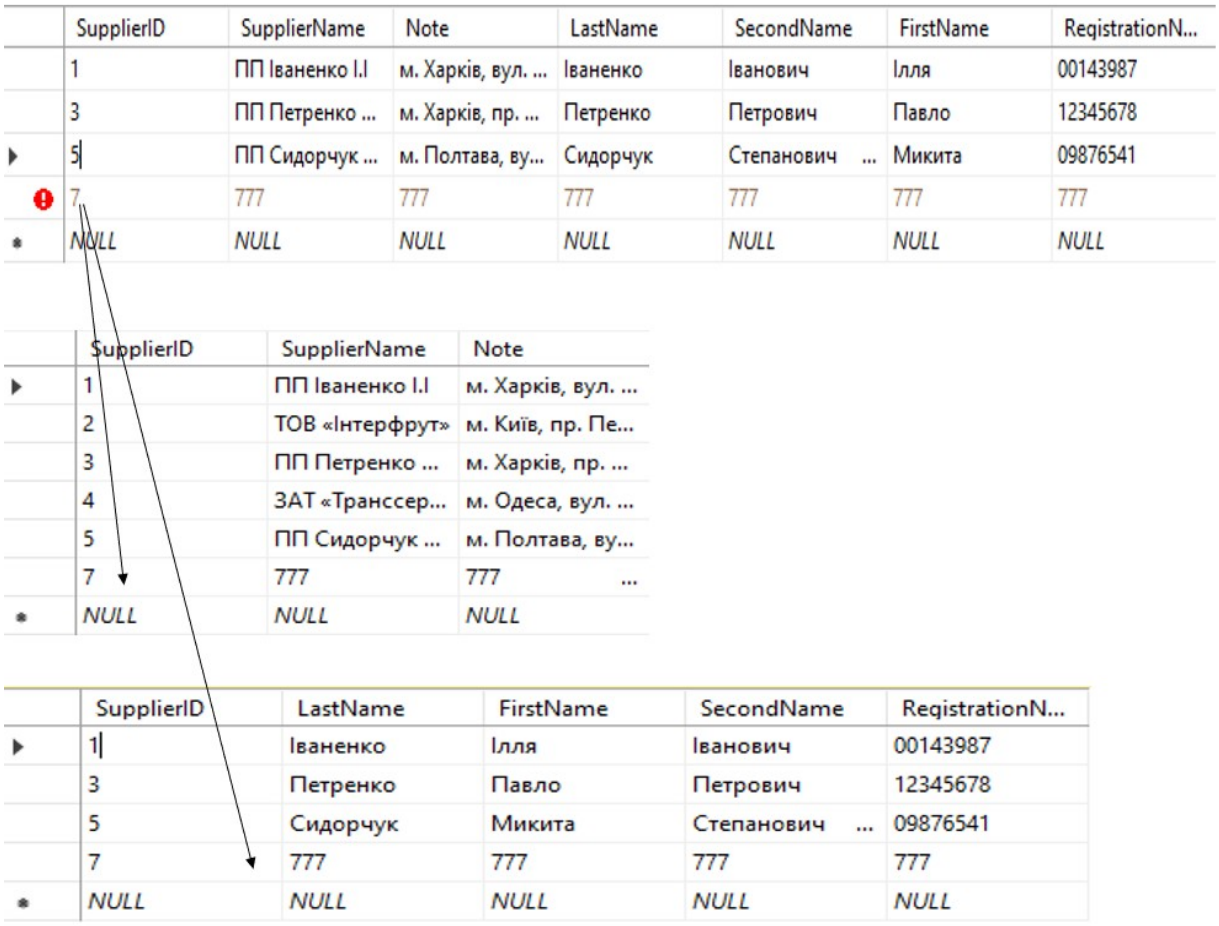



Рисунок 6.20

Наведені вище приклади представлень є досить простими. Більш детальну інформацію щодо побудови та використання представлень засобами мови Transact-SQL, можна, наприклад, отримати за посиланням:

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver16

### 6.2.4 Звітність про виконання практичних завдань теми 6

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) для кожного з реалізованих представлень навести призначення, текст запиту для створення та результат використання представлення;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 6.3 Питання для самоперевірки по темі 6

1. Представлення в SQL. Загальна характеристика, призначення та особливості використання.

2. Які можливості реалізації представлень підтримують СУБД?

3. Команди мови SQL для роботи з представленнями.

4. Оновлення даних в представленнях. Яким умовам повинно відповідати представлення, за допомогою якого можна оновлювати дані?

5. Які представлення не можуть бути використані для оновлення даних? Чим це обумовлено?

6. Параметр представлення WITH CHECK OPTION. Призначення і використання.

7. Параметр представлення WITH ENCRYPTION. Призначення і використання.

8. Тригери INSTEAD OF. Загальна характеристика, призначення та використання у представленнях.

9. Переваги використання представлень.

10. Недоліки використання представлень.

# ТЕМА 7
# ВИВЧЕННЯ ОСНОВ РОБОТИ ІЗ ЗАСОБАМИ КОНТРОЛЮ ЦІЛІСНОСТІ ДАНИХ У СЕРЕДОВИЩІ СУБД MICROSOFT SQL SERVER

## 7.1 Теоретичні відомості

### 7.1.1 Основні поняття цілісності даних

Виконання операторів модифікації даних (INSERT, DELETE, UPDATE) в таблицях бази даних може привести до порушення узгодженості даних і їх коректності, тобто до втрати їх достовірності і несуперечності.

Щоб інформація, що зберігається у базі даних, була однозначною і несуперечливою, в реляційній моделі встановлюються деякі обмежувальні умови - правила, що визначають можливі значення даних і таких, що забезпечують логічну основу для підтримки коректних значень. Обмеження цілісності дозволяють звести до мінімуму помилки, що виникають при оновленні і обробці даних.

У базі даних, побудованій на реляційній моделі, задається ряд правил цілісності, які, по суті, є обмеженнями для усіх допустимих станів бази даних і гарантують коректність даних.

Основні типи обмежень цілісності даних:

1. Обов'язкові дані.
2. Обмеження для доменів та атрибутів.
3. Цілісність сутностей.
4. Посилальна цілісність.
5. Вимоги замовника.

**Обов'язкові дані** передбачають, що деякі атрибути завжди повинні містити одно з допустимих значень, тобто, ці атрибути не можуть мати порожнього значення.

**Обмеження для доменів та атрибутів** передбачають, що кожен атрибут належить до певного домену, який визначає множину допустимих значень атрибуту.

**Цілісність сутностей** передбачає виконання низки умов, зокрема:

а) у кожній сутності повинен бути визначений первинний ключ;

б) повинна існувати можливість гарантувати унікальність значень і для будь-яких альтернативних (тобто, UNIQUE) ключів;

в) повинна існувати можливість визначення атрибутів, що є обов'язковими для заповнення (NOT NULL);

г) усі сутності в реляційній базі даних повинні бути зв'язаними. Видаленню (або зміні первинного ключа) не підлягають сутності, якщо існують об'єкти, що посилаються на них.

**Посилальна цілісність** передбачає існування набору правил, що забезпечують відповідність ключових значень у сутностях, зв'язаних за допомогою первинних та зовнішніх ключів.

Вказане обмеження цілісності торкається зовнішніх ключів. Зовнішній ключ – це поле (чи множина полів) однієї таблиці, що є первинним ключем іншої (чи тій же самій) таблиці. Зовнішні ключі використовуються для встановлення логічних зв'язків між таблицями. Зв'язок встановлюється шляхом привласнення значень зовнішнього ключа однієї таблиці значенням ключа інший.

Між двома або більше таблицями бази даних можуть існувати стосунки підлеглості, які визначають, що для кожного запису головної таблиці (званою ще батьківською ) може існувати одна або декілька записів в підпорядкованій таблиці (званою так же дочірньої ).

Існує три різновиди зв'язку між таблицями бази даних :

- «один-до-багатьох»;
- «один-до-одного»;
- «багато-до-багатьох».

Відношення «один-до-багатьох» має місце, коли одному запису батьківської таблиці може відповідати декілька записів дочірньої. Зв'язок

«один-до-багатьох» іноді називають зв'язком «багато-до-одного». І у тому, і в іншому випадку суть зв'язку між таблицями залишається незмінною.

Зв'язок «один-до-багатьох» найбільш поширений для реляційних баз даних. Вона дозволяє моделювати також ієрархічні структури даних.

Відношення «один-до-одного» має місце, коли одному запису у батьківській таблиці відповідає один запис в дочірній. Це відношення зустрічається набагато рідше, ніж відношення «один-до-багатьох». Його використовують, якщо не хочуть, щоб таблиця у базі даних «розпухала» від другорядної інформації. Використання зв'язку «один-до-одного» призводить до того, що для читання пов'язаної інформації в декількох таблицях доводиться робити декілька операцій читання замість однієї, коли дані зберігаються в одній таблиці.

Відношення «багато-до-багатьох» має місце в наступних випадках:

- одному запису у батьківській таблиці відповідає більше за один запис в дочірній таблиці;
- одному запису в дочірній таблиці відповідає більше за один запис у батьківській таблиці.

Вважається, що всякий зв'язок «багато-до-багатьох» може бути замінений на зв'язок «один-до-багатьох» (один або декілька).

Часто зв'язок між таблицями встановлюється по первинному ключу, тобто значення первинного ключа однієї таблиці привласнюються значенню зовнішнього ключа іншої. Поля зовнішнього ключа не зобов'язані мати тих же імен, що і імена ключів, яким вони відповідають. Зовнішній ключ може посилатися на свою власну таблицю – у такому разі зовнішній ключ називається рекурсивним.

Посилальна цілісність визначає: якщо в таблиці існує зовнішній ключ, то його значення повинне або відповідати значенню первинного ключа деякого запису у базовій таблиці, або задаватися визначником NULL.

Існує декілька важливих моментів, пов'язаних із зовнішніми ключами. По-перше, слід проаналізувати, чи допустиме використання в

зовнішніх ключах порожніх значень. У загальному випадку, якщо участь дочірньої таблиці в зв'язку є обов'язковою, то рекомендується забороняти застосування порожніх значень у відповідному зовнішньому ключі. В той же час, якщо має місце часткова участь дочірньої таблиці в зв'язку, то приміщення порожніх значень в поле зовнішнього ключа має бути дозволене.

Наступна проблема пов'язана з організацією підтримки посилальної цілісності при виконанні операцій модифікації даних у базі. Тут можливі наступні ситуації:

1. Вставка нового рядка в дочірню таблицю. Для забезпечення посилальної цілісності необхідно переконатися, що значення зовнішнього ключа нового рядка дочірньої таблиці рівно порожньому значенню або деякому конкретному значенню, присутньому в поле первісного ключа одного з рядків батьківської таблиці.

2. Видалення рядка з дочірньої таблиці. Ніяких порушень посилальної цілісності не відбувається.

3. Оновлення зовнішнього ключа в рядку дочірньої таблиці. Цей випадок подібний до описаної вище першої ситуації. Для збереження цілісності посилання необхідно переконатися, що значення зовнішнього ключа в оновленому рядку дочірньої таблиці рівно порожньому значенню або деякому конкретному значенню, присутньому в полі первинного ключа одного з рядків батьківської таблиці.

4. Вставка рядка у батьківську таблицю. Така вставка не може викликати порушення посилальної цілісності. Доданий рядок просто стає батьківським об'єктом, що не має дочірніх об'єктів.

5. Видалення рядка з батьківської таблиці. Посилальна цілісність виявиться порушеною, якщо в дочірній таблиці існуватимуть рядки, що посилаються на видалений рядок батьківської таблиці. В цьому випадку може використовуватися одна з наступних стратегій:

NO ACTION. Видалення рядка з батьківської таблиці забороняється, якщо в дочірній таблиці існує хоч би один рядок, що посилається на неї.

CASCADE. При видаленні рядка з батьківської таблиці автоматично видаляються усі рядки дочірньої таблиці, що посилаються на неї. Якщо будь-який з рядків дочірньої таблиці, що видаляються, виступає батьківською стороною в якому-небудь іншому зв'язку, то операція видалення застосовується до усіх рядків дочірньої таблиці цього зв'язку і так далі. Іншими словами, видалення рядка батьківської таблиці автоматично поширюється на будь-які дочірні таблиці.

SET NULL. При видаленні рядка з батьківської таблиці в усіх рядках дочірнього відношення, що посилаються на неї, в поле зовнішнього ключа, що відповідає первинному ключу видаленого рядка, записується порожнє значення. Отже, видалення рядків з батьківської таблиці викличе занесення порожнього значення у відповідне поле дочірньої таблиці. Ця стратегія може використовуватися, тільки коли в полі зовнішнього ключа дочірньої таблиці дозволяється поміщати порожні значення.

SET DEFAULT. При видаленні рядка з батьківської таблиці в поле зовнішнього ключа усіх рядків дочірньої таблиці, що посилаються на неї, автоматично поміщається значення, вказане для цього поля як значення за умовчанням. Таким чином, видалення рядка з батьківської таблиці викликає приміщення значення, що набуває за умовчанням, в поле зовнішнього ключа усіх рядків дочірньої таблиці, що посилаються на видалений рядок. Ця стратегія застосовна лише в тих випадках, коли полю зовнішнього ключа дочірньої таблиці призначено деяке значення, що приймається за умовчанням.

NO CHECK. При видаленні рядка з батьківської таблиці ніяких дій зі збереження посилальної цілісності даних не робиться.

6. Оновлення первинного ключа в рядку батьківської таблиці. Якщо значення первинного ключа деякого рядка батьківської таблиці буде оновлено, порушення посилальної цілісності станеться за тієї умови, що в

дочірньому відношенні існують рядки, що посилаються на початкове значення первинного ключа. Для збереження посилальної цілісності може застосовуватися будь-яка з описаних вище стратегій. При використанні стратегії CASCADE оновлення значення первинного ключа в рядку батьківської таблиці буде відображене у будь-якому рядку дочірньої таблиці, що посилається на цей рядок.

Існує і інший вид цілісності – смислова (семантична) цілісність бази даних. Вимога смислової цілісності визначає, що дані у базі даних повинні змінюватися так, щоб не порушувався смисловий зв'язок, що склався між ними.

Рівень підтримки цілісності даних в різних системах істотно варіюється.

Ідеологія архітектури клієнт-сервер вимагає перенесення максимально можливого числа правил цілісності даних на сервер. До переваг такого підходу відносяться:

• гарантія цілісності бази даних, оскільки усі правила зосереджені в одному місці (у базі даних);

• автоматичне застосування визначених на сервері обмежень цілісності для будь-яких застосувань;

• відсутність різних реалізацій обмежень в різних клієнтських застосуваннях, працюючих з базою даних;

• швидке спрацьовування обмежень, оскільки вони реалізовані на сервері і, отже, немає необхідності посилати дані клієнтові, збільшуючи при цьому мережевий трафік;

• доступність внесених в обмеження на сервері змін для усіх клієнтських застосувань, працюючих з базою даних, і відсутність необхідності повторного поширення змінених додатків клієнтів серед користувачів.

До недоліків зберігання обмежень цілісності на сервері можна віднести:

• відсутність у клієнтського додатка можливості реагувати на деякі помилкові ситуації, що виникають на сервері при реалізації тих або інших правил (наприклад, помилок при виконанні процедур, що зберігаються, на сервері);

• обмеженість можливостей мови SQL і мови процедур, що зберігаються, і тригерів для реалізації усіх виникаючих потреб визначення цілісності даних.

На практиці в клієнтських застосунках реалізують лише такі правила, які важко або неможливо реалізувати із застосуванням засобів сервера. Усі інші обмеження цілісності даних переносяться на сервер.

**Вимоги замовника** (або корпоративні обмеження цілісності, або вимоги конкретного підприємства) передбачають існування додаткових правил підтримки цілісності даних, що визначаються користувачами, зовнішнім середовищем, прийняті на підприємстві або адміністраторами баз даних. Можуть вимагати застосування специфічних засобів контролю, зокрема тригерів.

### 7.1.2 Визначення обмежень цілісності

При створенні баз даних велика увага має бути приділена засобам підтримки даних в цілісному стані. Розглянемо передбачені стандартом мови SQL функції, які призначені для підтримки цілісності даних. Ця підтримка включає засоби завдання обмежень, вони вводяться з метою захисту бази даних від порушення узгодженості що зберігаються в ній даних. Велика частина цих обмежень задається в операторах CREATE TABLE і ALTER TABLE.

У стандарті SQL дано декілька варіантів визначення оператора створення таблиці, проте його базовий формат має наступний вигляд:

<визначення_таблиці> ::=
CREATE TABLE ім'я_таблиці
{(ім'я_стовпця тип_даних [ NOT NULL ][ UNIQUE]

```
[DEFAULT <значення>]
[ CHECK (<умова_вибору>)][,..n]}
[CONSTRAINT ім'я_обмеження]
[PRIMARY KEY (ім'я_стовпця [,..n])
{[UNIQUE (ім'я_стовпця [,..n])}
[FOREIGN KEY (ім'я_стовпця_зовнішнього_ключа
      [,..n])
REFERENCES ім'я_рід_таблиці
      [(ім'я_стовпця_рід_таблиці [,..n])],
[MATCH {PARTIAL | FULL}]
[ON UPDATE {CASCADE| SET NULL |SET DEFAULT
      |NO ACTION}]
[ON DELETE {CASCADE| SET NULL |SET DEFAULT
            |NO ACTION}]
{[CHECK(<умова_вибору>)][,..n]})
```

Представлена версія оператора створення таблиці включає засоби визначення вимог цілісності даних, а також інші конструкції. Є дуже великі варіації в наборі функціональних можливостей цього оператора, реалізованих в різних діалектах мови SQL. Розглянемо призначення параметрів команди, використовуваних для підтримки цілісності даних.

Обов'язкові дані. Для деяких стовпців потрібно наявність в кожному рядку таблиці конкретного і допустимого значення, відмінного від опущеного значення або значення NULL. Для завдань обмежень подібного типу стандарт SQL передбачає використання специфікації NOT NULL.

Вимоги конкретного підприємства. Оновлення даних в таблицях можуть бути обмежені існуючими в організації вимогами (бізнес-правилами). Стандарт SQL дозволяє реалізувати бізнес-правила підприємств за допомогою пропозиції CHECK і ключового слова UNIQUE.

Обмеження для доменів полів. Кожен стовпець має власний домен – деякий набір допустимих значень. Стандарт SQL передбачає два різні

механізми визначення доменів. Перший полягає у використанні пропозиції CHECK, що дозволяє задати необхідні обмеження для стовпця або таблиці в цілому, а другою припускає застосування оператора CREATE DOMAIN.

Цілісність сутностей. Первинний ключ таблиці повинен мати унікальне не порожнє значення в кожному рядку. Стандарт SQL дозволяє задавати подібні вимоги підтримки цілісності даних за допомогою фрази PRIMARY KEY. В межах таблиці вона може вказуватися тільки один раз. Проте існує можливість гарантувати унікальність значень і для будь-яких альтернативних ключів таблиці, що забезпечує ключове слово UNIQUE. Крім того, при визначенні альтернативних ключів рекомендується використовувати і специфікатори NOT NULL.

Посилальна цілісність. Зовнішні ключі є стовпцями або наборами стовпців, призначеними для зв'язування кожної з рядків дочірньої таблиці, що містить цей зовнішній ключ, з рядком батьківської таблиці, що містить відповідне значення потенційного ключа. Стандарт SQL передбачає механізм визначення зовнішніх ключів за допомогою пропозиції FOREIGN KEY, а фраза REFERENCES визначає ім'я батьківської таблиці, тобто таблиці, де знаходиться відповідний потенційний ключ. При використанні цієї пропозиції система відхилить виконання будь-яких операторів INSERT або UPDATE, за допомогою яких буде зроблена спроба створити в дочірній таблиці значення зовнішнього ключа, що не відповідає одному із вже існуючих значень потенційного ключа батьківської таблиці. Коли дії системи виконуються при вступі операторів UPDATE і DELETE, що містять спробу відновити або видалити значення потенційного ключа у батьківській таблиці, якому відповідає одна або більше за рядки дочірньої таблиці, то вони залежать від правил підтримки посилальної цілісності, вказаних у фразах ON UPDATE і ON DELETE пропозиції FOREIGN KEY. Якщо користувач робить спробу видалити з батьківської таблиці рядок, на який посилається одна або більше за рядки дочірньої таблиці, мова SQL надає наступні можливості:

- CASCADE – виконується видалення рядка з батьківської таблиці, що супроводжується автоматичним видаленням усіх рядків дочірньої таблиці, що посилаються на неї;
- SET NULL – виконується видалення рядка з батьківської таблиці, а в зовнішні ключі усіх рядків дочірньої таблиці, що посилаються на неї, записується значення NULL;
- SET DEFAULT – виконується видалення рядка з батьківської таблиці, а в зовнішні ключі усіх рядків дочірньої таблиці, що посилаються на неї, заноситься значення, що приймається за умовчанням;
- NO ACTION – операція видалення рядка з батьківської таблиці відміняється. Саме це значення використовується за умовчанням в тих випадках, коли в описі зовнішнього ключа фраза ON DELETE опущена.

Ті ж самі правила застосовуються в мові SQL і тоді, коли значення потенційного ключа батьківської таблиці оновлюється.

Визначник MATCH дозволяє уточнити спосіб обробки значення NULL в зовнішньому ключі.

При визначенні таблиці пропозиція FOREIGN KEY може вказуватися довільна кількість разів.

У операторові CREATE TABLE використовується необов'язкова фраза DEFAULT, яка призначена для задання по замовчуванні значення, коли в операторі INSERT значення в цьому стовпці буде відсутнє.

Фраза CONSTRAINT дозволяє задати ім'я обмеженню, що дозволить згодом відмінити те або інше обмеження за допомогою оператора ALTER TABLE.

Для внесення змін до вже створених таблиць стандартом SQL передбачений оператор ALTER TABLE, призначений для виконання наступних дій:

- додавання в таблицю нового стовпця;
- видалення стовпця з таблиці;
- додавання у визначення таблиці нового обмеження;

- видалення з визначення таблиці існуючого обмеження;
- завдання для стовпця значення за умовчанням;
- відміна для стовпця значення за умовчанням.

Оператор зміни таблиці має наступний узагальнений формат:

```
<зміна_таблиці> ::=
ALTER TABLE ім'я_таблиці
[ADD [COLUMN]ім'я_стовпця тип_даних
            [ NOT NULL ][UNIQUE]
[DEFAULT <значення>][ CHECK (<умова_вибору>)]]
[DROP [COLUMN] ім'я_стовпця [RESTRICT | CASCADE ]]
[ADD [CONSTRAINT [ім'я_обмеження]]
[{PRIMARY KEY (ім'я_стовпця [,..n])
    |[UNIQUE (ім'я_стовпця [,..n])}
|[FOREIGN KEY (ім'я_стовпця_зовнішнього_ключа [,..n])
    REFERENCES ім'я_рід_таблиці
        [(ім'я_стовпця_рід_таблиці [,..n])],
[ MATCH {PARTIAL | FULL}
    [ON UPDATE {CASCADE| SET NULL |
        SET DEFAULT | NO ACTION}]
    [ON DELETE {CASCADE| SET NULL |
        SET DEFAULT | NO ACTION}]
    |[CHECK(<умова_вибору>)][,..n]}]
[DROP CONSTRAINT ім'я_обмеження
    [RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT <значення>]
[ALTER [COLUMN] DROP DEFAULT]
```

Тут параметри мають те ж саме призначення, що і у визначенні оператора CREATE TABLE.

Оператор ALTER TABLE реалізований не в усіх діалектах мови SQL. У деяких діалектах він підтримується, проте не дозволяє видаляти з таблиці вже існуючі стовпці.

Для видалення таблиці використовується команда DROP TABLE.

### 7.1.3 Визначення обмежень цілісності в середовищі MS SQL Server

В процесі проектування бази даних приймається рішення про те, які таблиці повинні входити у базу даних, які у них будуть імена (ідентифікатори), які типи даних знадобляться для побудови таблиць і які користувачі отримають доступ до кожної з них. Крім того, для ефективного створення таблиць необхідно відповісти на наступні питання:

- Стовпці якого типу і розміру складатимуть кожну з таблиць, які вимагається вибрати імена для стовпців таблиць?
- Які стовпці можуть містити значення NULL?
- Чи будуть використані обмеження цілісності, значення за умовчанням і правила для стовпців?
- Чи потрібне індексування стовпців, які типи індексів будуть застосовані для конкретних стовпців?
- Які стовпці входитимуть в первинні і зовнішні ключі.

Для створення таблиць в середовищі MS SQL Server використовується команда:

<визначення_таблиці> ::=

CREATE TABLE [ ім'я_бази_даних.[власник].

| власник. ]ім'я_таблиці

( (<елемент_таблиці>[,..n])

де

<елемент_таблиці> ::=

{<визначення_стовпця>}

| <ім'я_стовпця> AS <вираження>

| <обмеження_таблиці>

Зазвичай власником таблиці (dbo) є той, хто її створив.

<Вираження> задає значення для обчислюваного стовпця. Обчислювані стовпці - це віртуальні стовпці, т. е. фізично в таблиці вони не зберігаються і обчислюються з використанням значень стовпців тієї ж таблиці. У вираженні для обчислюваного стовпця можуть бути присутніми імена звичайних стовпців, константи і функції, пов'язані одним або декількома операторами. Підзапити в такому вираженні брати участь не можуть. Обчислювані стовпці можуть бути включені в розділ SELECT при вказівці списку стовпців, які мають бути повернені в результаті виконання запиту. Обчислювані стовпці не можуть входити в зовнішній ключ, для них не використовуються значення за умовчанням. Крім того, обчислювані стовпці не можуть брати участь в операціях INSERT і DELETE.

<визначення_стовпця> ::=
{ ім'я_стовпця <тип_даних>}
[ [ DEFAULT <вираження> ]
| [ IDENTITY (початок, крок)[NOT FOR REPLICATION]]]]
[ROWGUIDCOL][<обмеження_стовпця>][..n]]

У визначенні стовпця звернемо увагу на параметр IDENTITY, який вказує, що відповідний стовпець буде стовпцем-лічильником. Для таблиці може бути визначений тільки один стовпець з такою властивістю. Можна додатково вказати початкове значення і крок приросту. Якщо ці значення не вказуються, то за умовчанням вони обоє рівні 1. Якщо з ключовим словом IDENTITY вказане NOT FOR REPLICATION, то сервер не виконуватиме автоматичного генерування значень для цього стовпця, а дозволить вставку в стовпець довільних значень.

В якості обмежень використовуються обмеження стовпця і обмеження таблиці. Відмінність між ними в тому, що обмеження стовпця застосовується тільки до певного поля, а обмеження таблиці - до груп з одного або більше за поля.

<обмеження_стовпця>::=
[ CONSTRAINT ім'я_обмеження ]
{ [ NULL | NOT NULL ]
| [ {PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ]
[ WITH FILLFACTOR=чинник_заповнення ]
[ ON {ім'я_групи_файлів | DEFAULT } ] ] ]
| [ [ FOREIGN KEY ]
REFERENCES ім'я_рід_таблиці
[(ім'я_стовпця_рід_таблиці) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
[ NOT FOR REPLICATION ]]
| CHECK [ NOT FOR REPLICATION](<балка_вираження>) }

<обмеження_таблиці>::=
[CONSTRAINT ім'я_обмеження ]
{ [ {PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ]
{(ім'я_стовпця [ASC | DESC][,..n])}
[WITH FILLFACTOR=чинник_заповнення ]
[ON {ім'я_групи_файлів | DEFAULT } ]]
|FOREIGN KEY[(ім'я_стовпця [,..n])]
REFERENCES ім'я_рід_таблиці
[(ім'я_стовпця_рід_таблиці [,..n])]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
| NOT  FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ] (балка_вираження)}

Розглянемо окремі параметри представлених конструкцій, пов'язані з обмеженнями цілісності даних. Обмеження цілісності мають пріоритет над тригерами, правилами і значеннями за умовчанням. До обмежень цілісності відносяться обмеження первинного ключа PRIMARY KEY, обмеження зовнішнього ключа FOREIGN KEY, обмеження унікальності UNIQUE, обмеження значення NULL, обмеження на перевірку CHECK .

Обмеження первинного ключа (PRIMARY KEY). Таблиця зазвичай має стовпець або комбінацію стовпців, значення яких унікально ідентифікують кожен рядок в таблиці. Цей стовпець (чи стовпці) називається первісним ключем таблиці і потрібний для забезпечення її цілісності. Якщо в первинний ключ входить більше за один стовпець, то значення в межах одного стовпця можуть дублюватися, але будь-яка комбінація значень усіх стовпців первинного ключа має бути унікальна.

При створенні первинного ключа SQL Server автоматично створює унікальний індекс для стовпців, що входять в первинний ключ. Він прискорює доступ до даних цих стовпців при використанні первинного ключа в запитах.

Таблиця може мати тільки одно обмеження PRIMARY KEY, причому жоден з включених в первинний ключ стовпців не може набувати значення NULL. При спробі використовувати як первинний ключ стовпець (чи групу стовпців), для якого обмеження первинного ключа не виконуються, первинний ключ створений не буде, а система видасть повідомлення про помилку.

Оскільки обмеження PRIMARY KEY гарантує унікальність даних, воно часто визначається для стовпців-лічильників. Створення обмеження цілісності PRIMARY KEY можливо як при створенні, так і при зміні таблиці. Одним з призначень первинного ключа є забезпечення посилальної цілісності даних декількох таблиць. Природно, це може бути реалізовано тільки при визначенні відповідних зовнішніх ключів в інших таблицях.

Обмеження зовнішнього ключа (FOREIGN KEY). Обмеження зовнішнього ключа – це основний механізм для підтримки посилальної цілісності між таблицями реляційної бази даних. Стовпець дочірньої таблиці, визначений в якості зовнішнього ключа в параметрі FOREIGN KEY, застосовується для посилання на стовпець батьківської таблиці, що є в ній первинним ключем. Ім'я батьківської таблиці і стовпці її первинного ключа вказуються в пропозиції REFERENCES. Дані в стовпцях, визначених в якості зовнішнього ключа, можуть приймати тільки такі ж значення, які знаходяться в пов'язаних з ним стовпцях первинного ключа батьківської таблиці. Збіг імен стовпців для зв'язку дочірньої і батьківської таблиць необов'язково. Первинний ключ може бути визначений для стовпця з одним ім'ям, тоді як стовпець, на який накладено обмеження FOREIGN KEY, може мати абсолютно інше ім'я. Єдиною вимогою залишається відповідність стовпців за типом і розміром даних.

На первинний ключ можуть посилатися не лише стовпці інших таблиць, але і стовпці, розташовані в тій же таблиці, що і власне первинний ключ ; це дозволяє створювати рекурсивні структури.

Зовнішній ключ може бути пов'язаний не лише з первинним ключем іншої таблиці. Він може бути визначений і для стовпців з обмеженням UNIQUE другої таблиці або будь-яких інших стовпців, але таблиці повинні знаходитися в одній базі даних.

Стовпці зовнішнього ключа можуть містити значення NULL, проте перевірка на обмеження FOREIGN KEY ігнорується. Зовнішній ключ може бути проіндексований, тоді сервер швидше відшукуватиме потрібні дані. Зовнішній ключ визначається як при створенні, так і при зміні таблиць.

Обмеження посилальної цілісності задає вимогу, згідно з якою для кожного запису в дочірній таблиці має бути запис у батьківській таблиці. При цьому зміна значення стовпця зв'язку в записі батьківської таблиці за наявності дочірнього запису блокується, так само як і видалення батьківського запису (заборона каскадної зміни і видалення), що гарантується параметрами ON DELETE NO ACTION і ON UPDATE NO

ACTION, прийнятими за умовчанням. Для дозволу каскадної дії слід використовувати параметри ON DELETE CASCADE і ON UPDATE CASCADE.

Обмеження унікального ключа (UNIQUE). Це обмеження задає вимогу унікальності значення поля (стовпця) або групи полів (стовпців), що входять в унікальний ключ, по відношенню до інших записів. Обмеження UNIQUE для стовпця таблиці схоже на первинний ключ: для кожного рядка даних в нім повинні міститися унікальні значення. Встановивши для деякого стовпця обмеження первинного ключа, можна одночасно встановити для іншого стовпця обмеження UNIQUE. Відмінність в обмеженні первинного і унікального ключа полягає в тому, що первинний ключ служить як для впорядкування даних в таблиці, так і для з'єднання пов'язаних між собою таблиць. Крім того, при використанні обмеження UNIQUE допускається існування значення NULL, але лише єдиний раз.

Обмеження на порожнє значення (NOT NULL). Для кожного стовпця таблиці можна встановити обмеження NOT NULL, що забороняє введення в цей стовпець нульового значення.

Обмеження перевірочне (CHECK) і правила. Це обмеження використовується для перевірки допустимості даних, що вводяться в конкретний стовпець таблиці, тобто обмеження CHECK забезпечує ще один рівень захисту даних.

Обмеження цілісності CHECK задають діапазон можливих значень для стовпця або стовпців. У основі обмежень цілісності CHECK лежить використання логічних виразів.

Допускається застосування декількох обмежень CHECK до одного і тому ж стовпця. В цьому випадку вони будуть застосовні в тій послідовності, в якій відбувалося їх створення. Можливе застосування одного і того ж обмеження до різних стовпців і використання в логічних виразах значень інших стовпців. Вказівка параметра NOT FOR

REPLICATION наказує не виконувати перевірочних дій, якщо вони виконуються підсистемою реплікації.

Перевірочні обмеження можуть бути реалізовані і за допомогою правил. Правило є самостійним об'єктом бази даних, який зв'язується із стовпцем таблиці або призначеним для користувача типом даних. Причому одно і те ж правило може бути одночасно пов'язане з декількома стовпцями і призначеними для користувача типами даних, що є безперечною перевагою. Проте істотний недолік полягає в тому, що з кожним стовпцем або типом даних може бути пов'язано тільки одне правило. Дозволяється комбінування обмежень цілісності CHECK з правилами. В цьому випадку виконується перевірка відповідності значення, що вводиться, як обмеженням цілісності, так і правилам.

Правило може бути створене командою:

CREATE RULE ім'я_правила AS вираження

Щоб зв'язати правило з тим або іншим стовпцем якої-небудь таблиці, необхідно використовувати системну процедуру, що зберігається :

sp_bindrule [@rulename=] 'rule'
[@objname=] 'object_name'
[,[@futureonly=['futureonly_flag']
Щоб відмінити правила, слід виконати наступну процедуру:
sp_unbindrule [@objname=] 'object_name'
[,[@futureonly=['futureonly_flag']
Видалення правила робиться командою
DROP RULE {ім'я_правила} [,..n].

Обмеження за умовчанням (DEFAULT). Стовпцю може бути присвоєне значення за умовчанням. Воно буде актуальним у тому випадку, якщо користувач не введе в стовпець ніякого іншого значення.

Окремо необхідно відмітити користь від використання значень за умовчанням при додаванні нового стовпця в таблицю. Якщо для стовпця, що додається, не дозволено зберігання значень NULL і не визначено значення за умовчанням, то операція додавання стовпця закінчиться невдачею.

При визначенні в таблиці стовпця з параметром ROWGUIDCOL сервер автоматично визначає для нього значення за умовчанням у вигляді функції NEWID(). Таким чином, для кожного нового рядка автоматично генеруватиметься глобальний унікальний ідентифікатор.

Додатковим механізмом використання значень за умовчанням є об'єкти бази даних, створені командою:

CREATE DEFAULT ім'я_умовчання AS константа

Умовчання зв'язується з тим або іншим стовпцем якої-небудь таблиці за допомогою процедури:

sp_bindefault [@defname=] 'default
[@objname=]  'object_name'
[,[@futureonly=] 'futureonly_flag']
де
"object_name'
може бути представлений як
"ім'я_таблиці.ім'я_стовпця'

Видалення обмеження за умовчанням виконується командою

DROP DEFAULT {ім'я_умовчання} [,..n]

якщо заздалегідь це обмеження було видалене з усіх таблиць процедурою

sp_unbindefault [@objname=] 'object_name'
[,[@futureonly=] 'futureonly_flag']

При створенні таблиці, окрім розглянутих прийомів, можна вказати необов'язкове ключове слово CONSTRAINT, щоб присвоїти обмеженню ім'я, унікальне в межах бази даних.

Ключові слова CLUSTERED і NONCLUSTERED дозволяють створити для стовпця кластерний або некластерний індекс. Для обмеження PRIMARY KEY за умовчанням створюється кластерний індекс, а для обмеження UNIQUE - некластерний. У кожній таблиці може бути створений лише один кластерний індекс, відмітною особливістю якого є те, що відповідно до нього змінюється фізичний порядок рядків в таблиці. ASC і DESC визначають метод впорядкування даних в індексі.

За допомогою параметра WITH FILLFACTOR=чинник_заповнення задається міра заповнення індексних сторінок при створенні індексу. Значення чинника заповнення вказується у відсотках і може змінюватися в проміжку від 0 до 100.

Параметр ON ім'я_групи_файлів означає групу, в якій передбачається зберігати таблицю.

Зміни в таблицю можна внести командою:

<зміна_таблиці> ::=
ALTER TABLE ім'я_таблиці
{[ALTER COLUMN ім'я_стовпця
{ тип_даних [(точність[,масштаб])]
[ NULL | NOT NULL ]
| {ADD | DROP } ROWGUIDCOL }]
| ADD { [<визначення_стовпця>]
| ім'я_стовпця AS вираження } [,..n]

| [WITH CHECK | WITH NOCHECK ]
ADD { <обмеження-таблиці> } [,..n]
| DROP
{ [CONSTRAINT ] ім'я_обмеження
| COLUMN ім'я_стовпця}[,..n]
| {CHECK | NOCHECK } CONSTRAINT
{ALL | ім'я_обмеження[,..n]}
| {ENABLE | DISABLE } TRIGGER
{ALL | ім'я_тригера [,..n]}}

На додаток до вже названих параметрів визначимо параметр {ENABLE | DISABLE } TRIGGER ALL, який дозволяє задіяти або відключити конкретний тригер або усі тригери, пов'язані з таблицею.

Видалення таблиці виконується командою:

DROP TABLE ім'я_таблиці

Видалити можна будь-яку таблицю, навіть системну. До цього питання треба підходити дуже обережно. Проте видаленню не підлягають таблиці, якщо існують об'єкти, що посилаються на них. До таких об'єктів відносяться таблиці, пов'язані з таблицею, що видаляється, за допомогою зовнішнього ключа. Тому, перш ніж видаляти батьківську таблицю, необхідно видалити або обмеження зовнішнього ключа, або дочірні таблиці. Якщо з таблицею пов'язано хоч би одно представлення, то таблицю також видалити не вдасться. Крім того, зв'язок з таблицею може бути встановлений з боку функцій і процедур. Отже, перед видаленням таблиці необхідно видалити усі об'єкти бази даних, які на неї посилаються, або змінити їх так, щоб посилань на таблицю, що видаляється, не було.

## 7.2 Практичне опанування теми 7

### 7.2.1 Передумови виконання завдань теми 7

При виконанні практичних завдань теми 7 передбачається використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи Microsoft SQL Server. При цьому передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Увага! При виконанні практичних завдань цієї теми рекомендується використовувати окрему базу даних. Її можна спеціально створити, використовуючи запити, створені при виконанні практичних завдань теми «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

Також треба звернути увагу на те, що далі будуть розглянуті питання, пов'язані із використанням засобів контролю посилальної цілісності. Це обумовлено тим, що інші види обмежень цілісності даних та засоби їх підтримки (наприклад, обов'язкові дані, тригери ті інші), вже розглядалися у практичних завданнях попередніх тем.

### 7.2.2 Вивчення особливостей роботи механізму контролю посилальної цілісності No Action

Розглянемо особливості роботи механізму посилальної цілісності No Action на прикладі відносин між таблицями «Suppliers» та «Contracts», «Suppliers» та «IndividualEntrepreneurs», «Suppliers» та «LegalEntities». Ці таблиці пов'язані між собою по полю SupplierID. У цих зв'язках таблиця «Suppliers» є батьківською, а таблиці «Contracts», «LegalEntities», «IndividualEntrepreneurs» – дочірніми. Для вивчення особливостей роботи механізму цілісності посилань виконаємо наступну послідовність дій.

1. У списку таблиць вибрати таблицю «Suppliers», клацнувши по ній правою кнопкою миші. У меню вибрати пункт Modify. В результаті буде отримано доступ до редагування структури таблиці.

2. Клацнути правою кнопкою миші по будь-якому полю таблиці і в меню вибрати пункт Relationships ... В результаті на екрані з'явиться вікно Foreign Key Relationships (рисунок 7.1).

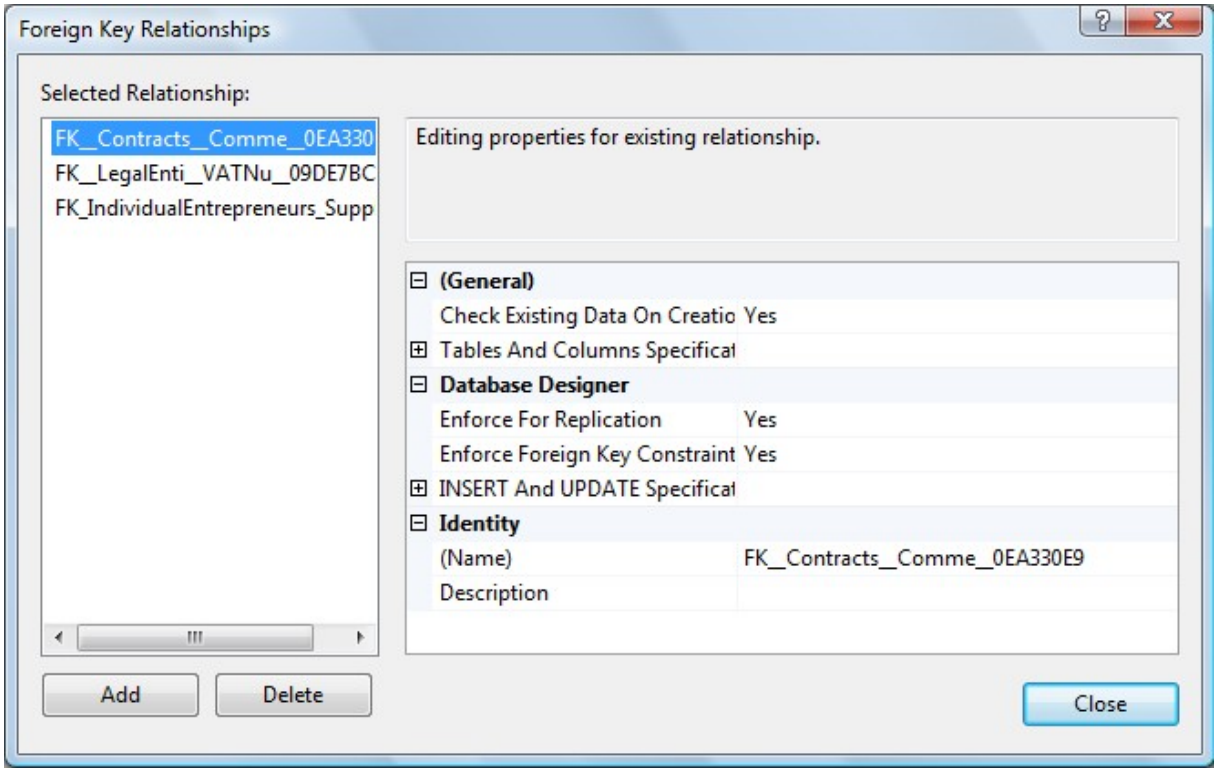



Рисунок 7.1

3. Вибрати зв'язок, що відповідає зв'язку між таблицями «Suppliers» та «Contracts» (рисунок 7.1). Розкривши пункт Tables And Columns Specification, можна побачити інформацію, що показує, які таблиці пов'язані і які ключі при цьому були використані.

4. Далі перейдемо до розгляду механізмів контролю посилальної цілісності, що використовуються при видаленні записів у таблиці «Suppliers» або зміни ключового значення (тобто значення поля SupplierID). Розкриємо пункт INSERT And UPDATE Specification (рисунок 7.2). Як видно, механізм цілісності посилання No Action

встановлений за умовчанням. Так само потрібно перевірити механізм посилальної цілісності для зв'язків між таблицями «Suppliers» та «IndividualEntrepreneurs» та «Suppliers» та «LegalEntities». Вікно Foreign Key Relationships необхідно закрити. Також необхідно закрити вікно, що забезпечує доступ до структури таблиці «Suppliers».

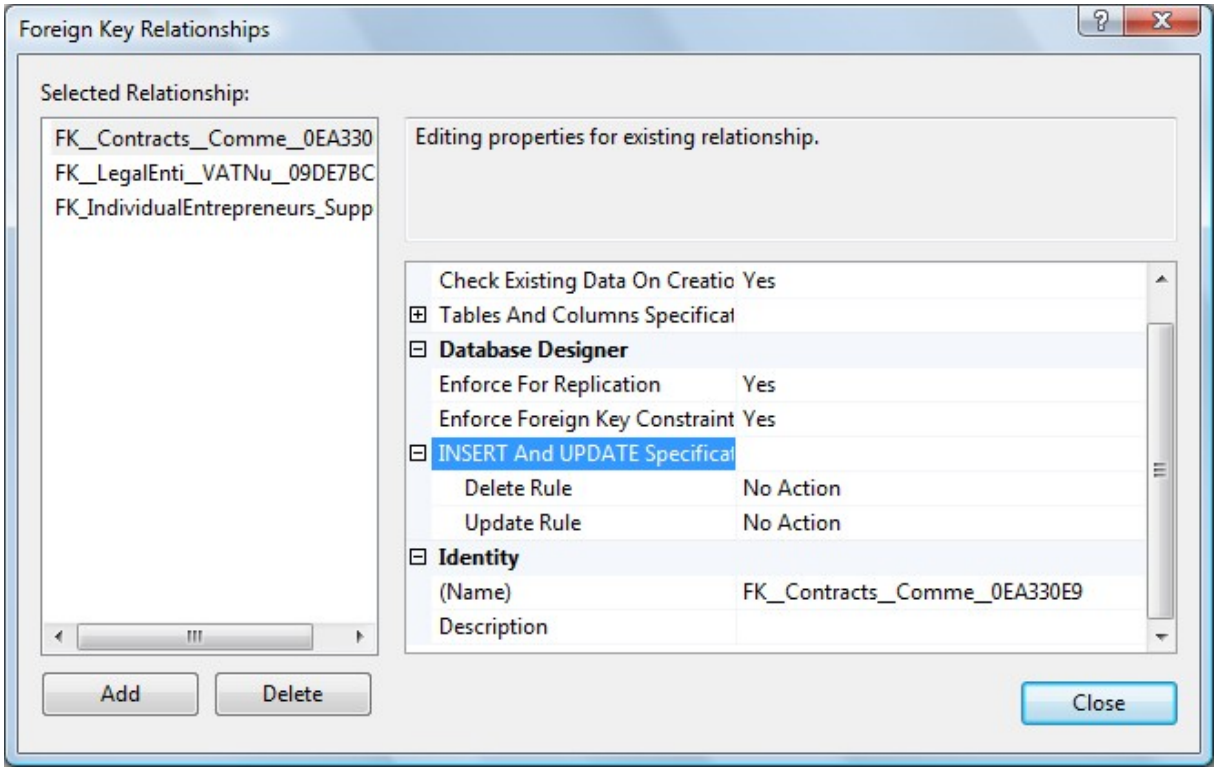



Рисунок 7.2

5. Відкрити таблицю «Suppliers» в режимі перегляду/редагування даних. Аналогічно потрібно відкрити таблиці «Contracts», «IndividualEntrepreneurs» та «LegalEntities».

6. Припустимо, що через якісь причини необхідно видалити постачальника з кодом 4. Вибравши відповідний запис у таблиці Постачальники, натисніть праву кнопку миші і в меню виберіть Delete. Потім підтвердьте видалення запису. Після цього на екран буде виведено вікно (рисунок 7.3), що інформує користувача про те, що видалення запису неможливе, тому що на цей запис посилаються записи у зв'язаних таблицях.

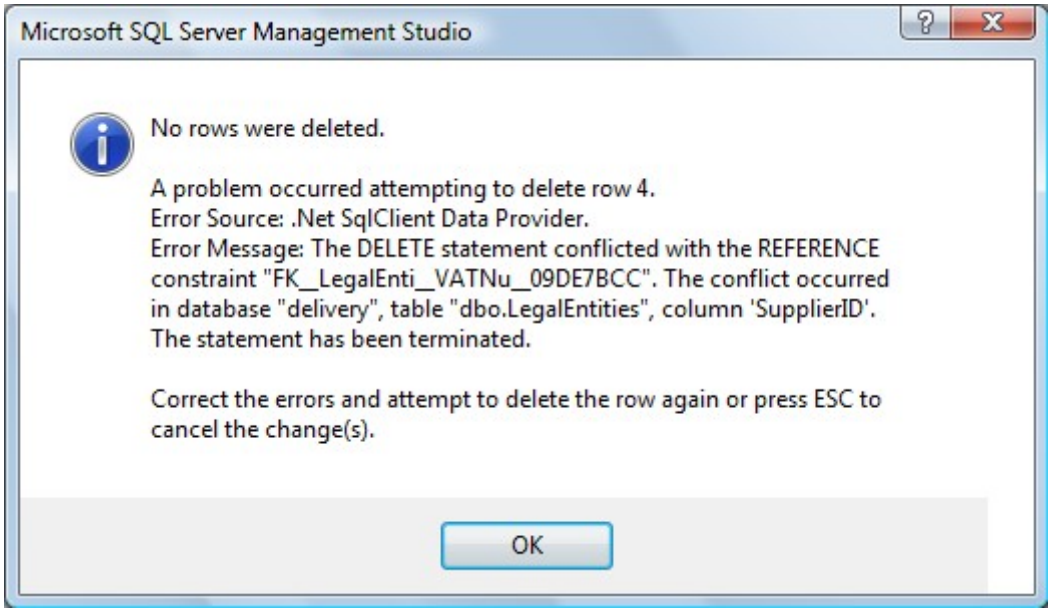



Рисунок 7.3

7. Таким чином, щоб видалити даного постачальника, потрібно попередньо видалити всі пов'язані з ним дані. Для цього потрібно видалити відповідний запис з таблиці «LegalEntities» та перевірити наявність договорів із цим постачальником у таблиці «Contracts». Якщо такі договори є, їх теж потрібно видалити (при цьому потрібно мати на увазі, що може виникнути потреба у видаленні та вмісті цих договорів). Після цього потрібно спробувати повторити спробу видалення постачальника з кодом 4. Якщо зв'язаних із ним даних немає, постачальника буде видалено.

8. Припустимо, що з якихось причин виникла потреба для постачальника з кодом 5 змінити код на 7. Вибравши відповідний запис у таблиці «Suppliers», змініть код постачальника з 5 на 7. Потім спробуйте перейти на попередній запис. Після цього на екран буде виведено вікно (рисунок 7.4), що інформує користувача про те, що зміна даних неможлива, тому що на цей код посилаються записи у зв'язаних таблицях. Оскільки договори з цим постачальником відсутні, посилання на нього є лише у таблиці «IndividualEntrepreneurs». Видаливши цей запис, повторіть спробу зміни коду постачальника з 5 на 7. Тепер ця операція повинна пройти успішно. Після цього потрібно перевірити вміст таблиць та таблиці закрити.

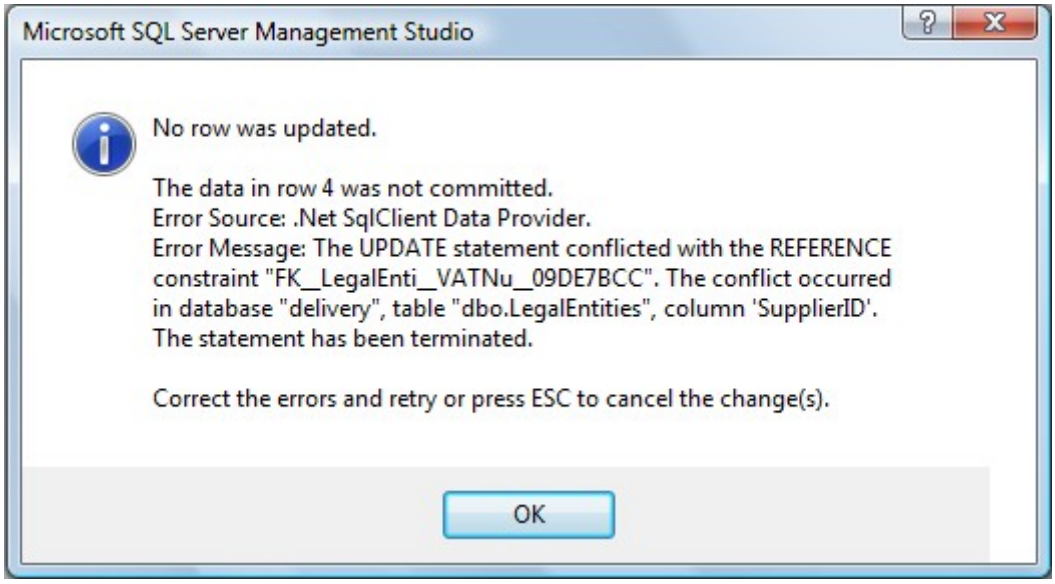



Рисунок 7.4

### 7.2.3 Вивчення особливостей роботи механізму контролю посилальної цілісності Cascade

Розглянемо особливості роботи механізму контролю посилальної цілісності Cascade на прикладі відносин між таблицями «Suppliers» та «Contracts», «Suppliers» та «IndividualEntrepreneurs», «Suppliers» та «LegalEntities», «Contracts» та «Supplied». Доступ до зв'язків виконується так, як описано вище. Для вивчення особливостей роботи механізму цілісності посилань виконаємо наступну послідовність дій.

1. Змінимо механізми цілісності посилань для зв'язків між усіма таблицями на Cascade. Приклад результату зміни наведений на рисунку 7.5.

2. Припустимо, що з якихось причин виникла потреба для постачальника з кодом 2 змінити код на 8. Вибравши відповідний запис у таблиці Постачальники, змініть код постачальника з 2 на 8. Потім спробуйте перейти на попередній запис. Перевірте зміни значення коду постачальника у безпосередньо пов'язаних з цим постачальником таблицях («LegalEntities», «Contracts»). Якщо зміни не з'явилися відразу, то таблицю

потрібно закрити і потім знову відкрити або у вікні вмісту таблиці клацнути правою кнопкою миші і в меню вибрати пункт Execute SQL.

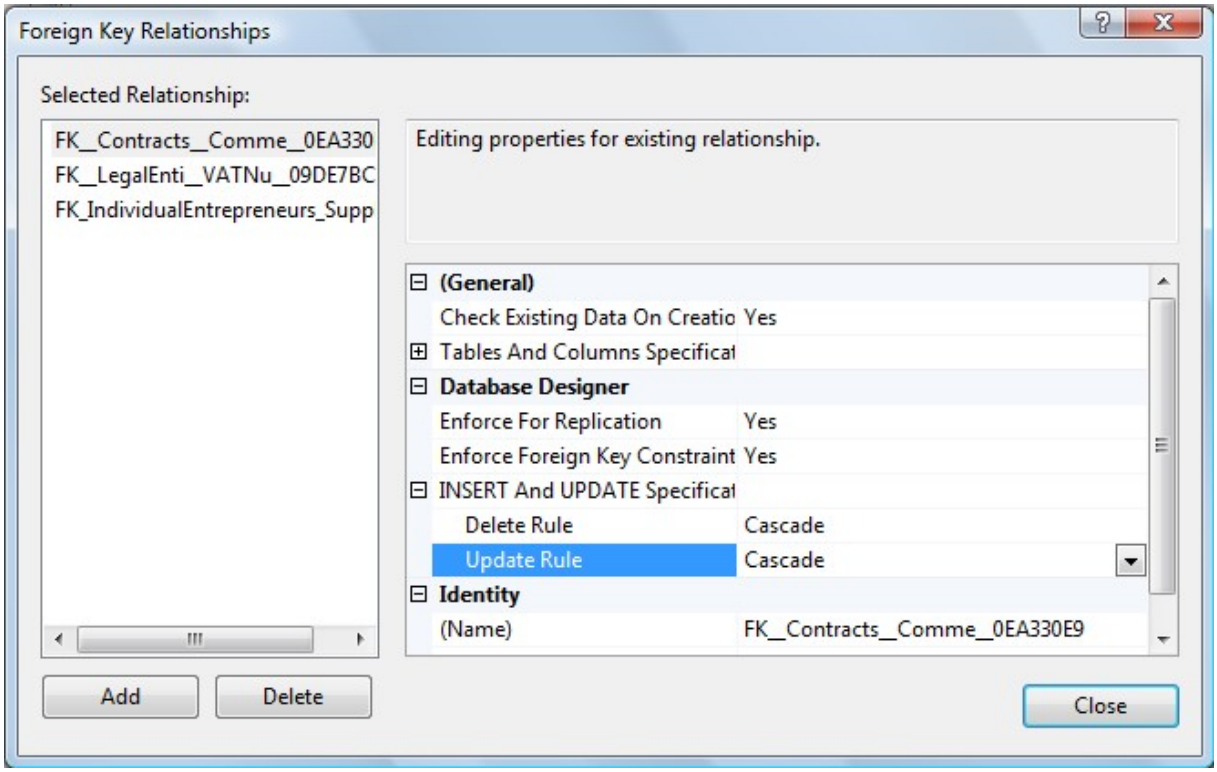



Рисунок 7.5

3. Тепер припустимо, що цього постачальника (який має код 8), необхідно видалити. Вибравши відповідний запис у таблиці «Suppliers», натисніть праву кнопку миші та в меню виберіть Delete. Потім підтвердьте видалення запису. Після цього перевірте стан даних у таблицях, які безпосередньо чи опосередковано пов'язані з цим постачальником («LegalEntities», «Contracts», «Supplied»). Переконайтеся, що дані видалено. Після цього потрібно таблиці закрити.

### 7.2.4 Вивчення особливостей роботи механізму контролю посилальної цілісності Set Null

Розглянемо особливості роботи механізму контролю посилальної цілісності Set Null на прикладі відносин між таблицями «Suppliers» та «Contracts». Доступ до зв'язків виконується так, як описано вище. Для

вивчення особливостей роботи механізму цілісності посилань виконаємо наступну послідовність дій.

1. У списку таблиць вибрати таблицю «Contracts», клацнувши по ній правою кнопкою миші. У меню вибрати пункт Modify. В результаті буде отримано доступ до редагування структури таблиці. Для поля SupplierID встановити властивість Allow Nulls (рисунок 7.6).

| Column Name | Data Type | Allow Nulls |
|---|---|---|
| ContractNumber | int | ☐ |
| ContractDate | datetime | ☑ |
| SupplierID | int | ☑ |
| ContractName | text | ☑ |
| Comment | text | ☑ |
| | | ☐ |

Рисунок 7.6

2. Клацнути правою кнопкою миші по будь-якому полю таблиці і в меню вибрати пункт Relationships ... В результаті на екрані з'явиться вікно Foreign Key Relationships. Змінити механізми цілісності посилання для зв'язку між таблицями «Suppliers» та «Contracts» на Set Null (рисунок 7.7). Зберегти зміни у таблиці.

3. Відкрити в режимі перегляду/модифікації даних таблиці «Suppliers» та «Contracts». Для договору 6 змінити код постачальника з 1 на 7. Потім у таблиці «Suppliers» змінити код постачальника 7 на 10. Перевірити дані у таблиці «Contracts». Код постачальника у договорі 6 повинен прийняти значення Null. Приклад таблиці із зміненими даними наведено рисунку 7.8.

4. У таблиці «Contracts» для договору 6 змініть код постачальника з Null на 10. Після цього в таблиці «Suppliers» видаліть постачальника з кодом 10. Перевірте стан даних у таблиці «Contracts». Для договору 6 значення коду постачальника знову має прийняти значення Null.

5. Відкрити для перегляду даних таблиці закрити.

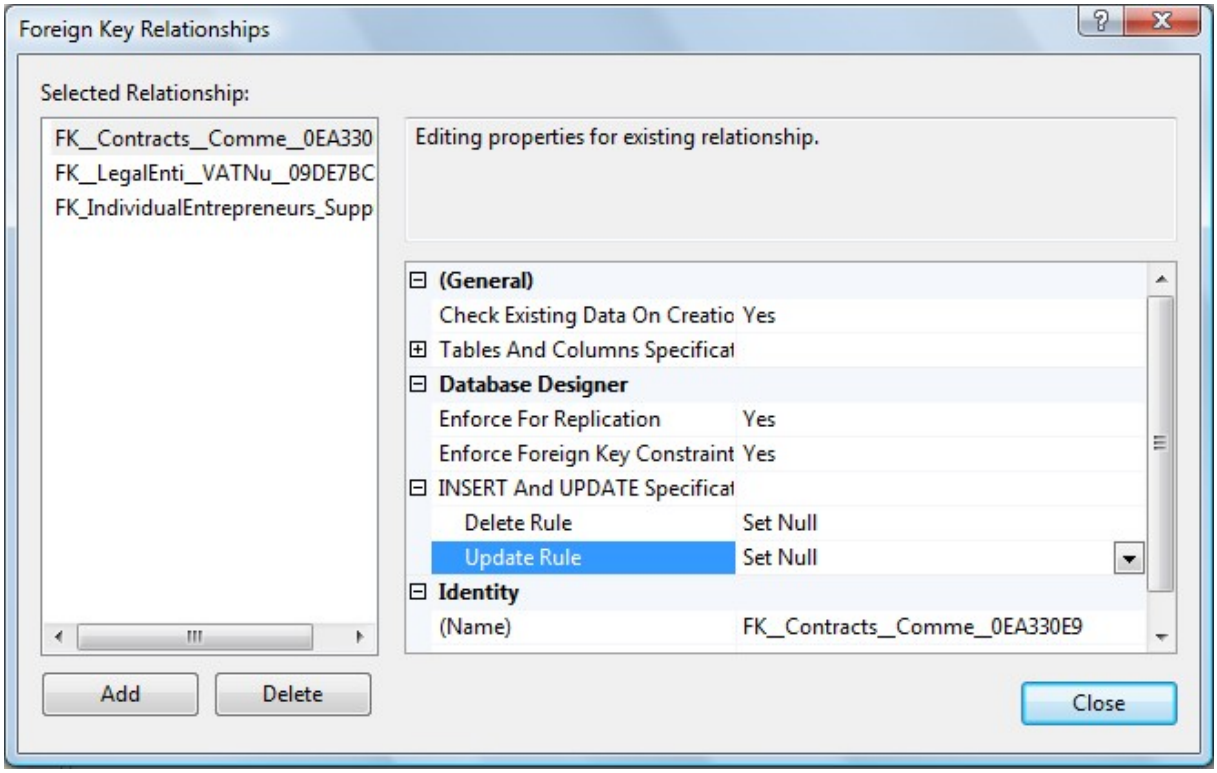



Рисунок 7.7

| | ContractNumber | ContractDate | SupplierID | ContractName | Comment |
|---|---|---|---|---|---|
| ▶ | 1 | 1999-09-01 00:... | 1 | Договір № 1 | Підстава - накл... |
| | 2 | 1999-09-10 00:... | 1 | Договір № 2 | Підстава - раху... |
| | 3 | 1999-09-10 00:... | 3 | Договір № 3 | Підстава - раху... |
| | 4 | 1999-09-23 00:... | 3 | Договір № 4 | Підстава - замо... |
| | 5 | 1999-09-24 00:... | 2 | Договір № 5 | Підстава - накл... |
| | 6 | 1999-10-01 00:... | *NULL* | Договір № 6 | Підстава - раху... |
| | 7 | 1999-10-02 00:... | 2 | Договір № 7 | Підстава - накл... |
| * | *NULL* | *NULL* | *NULL* | *NULL* | *NULL* |

Рисунок 7.8

### 7.2.5 Вивчення особливостей роботи механізму контролю посилальної цілісності Set Default

Розглянемо особливості роботи механізму контролю посилальної цілісності Set Default на прикладі відносин між таблицями «Suppliers» та «Contracts». Доступ до зв'язків виконується так, як описано вище. Для вивчення особливостей роботи механізму цілісності посилань виконаємо наступну послідовність дій.

1. Відкрити таблицю «Contracts» у режимі перегляду/модифікації даних. Для договору 6 змінити значення коду постачальника Null на 3. Закрити таблицю.

2. Відкрити таблицю «Contracts» у режимі редагування структури. Для поля SupplierID вимкнути властивість Allow Nulls (рисунок 7.9).

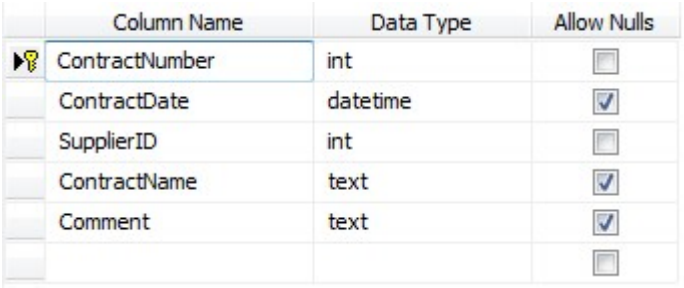



Рисунок 7.9

3. Вибрати поле SupplierID та встановити значення за умовчанням для цього поля. Для цього встановити для властивості Default Value or Binding значення 1 (рисунок 7.10).

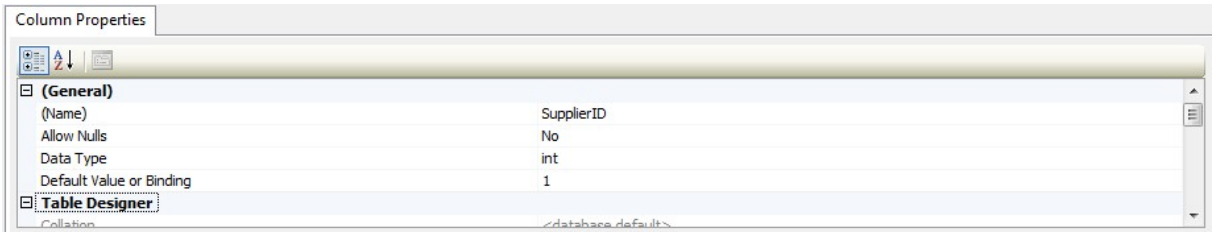



Рисунок 7.10

4. Клацнути правою кнопкою миші по будь-якому полю таблиці і в меню вибрати пункт Relationships…. На екрані з'явиться вікно Foreign Key Relationships. Змінити механізми цілісності посилання для зв'язку між таблицями Постачальники та Договори на Set Default (рисунок 7.11). Закрити вікно Foreign Key Relationships та зберегти зміни в таблиці.

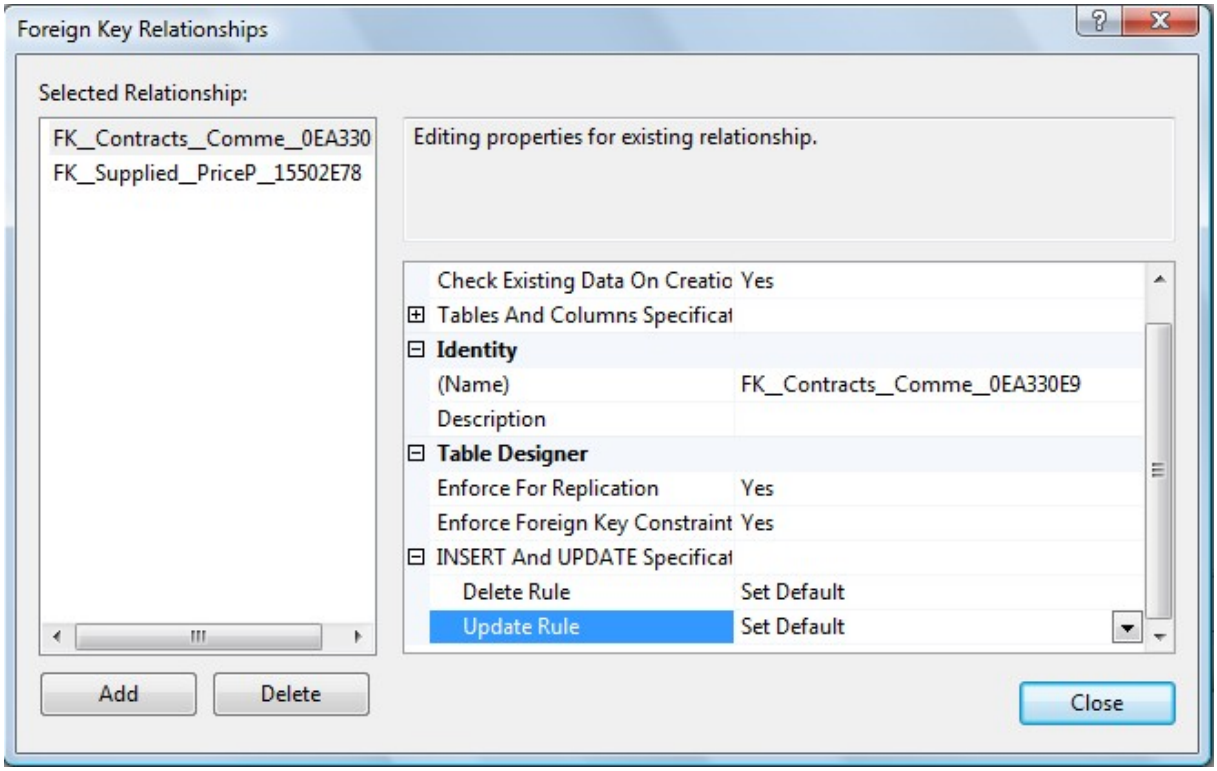



Рисунок 7.11

5. Відкрити в режимі перегляду даних таблиці «Suppliers» та «Contracts». У таблиці «Contracts» визначити список договорів, для яких код постачальника дорівнює 3. У таблиці «Suppliers» змінити код постачальника 3 на 12. Перевірити дані у таблиці «Contracts». Для договорів, для яких код постачальника дорівнював 3, код постачальника повинен змінитися на 1.

6. У таблиці «Contracts» змінити для деяких договорів (наприклад, для договорів 3, 4, 6) код постачальника з 1 до 12.

7. У таблиці «Suppliers» видалити запис, що відповідає постачальнику з кодом 12.

8. Перевірити дані у таблиці «Contracts». Для договорів, для яких код постачальника дорівнював 12, код постачальника повинен змінитися на 1.

Наведені вище приклади застосування механізмів контролю посилальної цілісності є досить простими. Більш детальну інформацію

щодо контролю посилальної цілісності засобами мови Transact-SQL, можна, наприклад, отримати за посиланнями:

https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver16

https://www.mullinsconsulting.com/sql_ref.htm

Після закінчення роботи всі таблиці слід закрити, а потім базу даних відключити. Базу даних можна не зберігати.

### 7.2.6 Звітність про виконання практичних завдань теми 7

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) описати особливості розглянутих механізмів контролю посилальної цілісності та результати їх використання;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 7.3 Питання для самоперевірки по темі 7

1. Наведіть основні типи обмежень цілісності даних.
2. Що таке обов'язкові дані?
3. Що таке обмеження для доменів полів?
4. Що таке корпоративні обмеження цілісності даних? Наведіть приклади.
5. Які засоби забезпечують контроль корпоративних обмежень цілісності даних?
6. Що таке цілісність сутностей?
7. Що таке посилальна цілісність?
8. За допомогою яких ключів реалізується посилальна цілісність?
9. Що таке первинний ключ?
10. Що таке зовнішній ключ?
11. Що таке альтернативний ключ? Яким чином можна реалізувати альтернативний ключ?
12. Які різновиди зв'язку можуть існувати між таблицями бази даних?

13. Організація підтримки посилальної цілісності при виконанні операцій модифікації даних у базі. Які при цьому можливі ситуації?

14. Наведіть основні типи стратегій (або механізмів) контролю посилальної цілісності.

15. Стратегія (або механізм) контролю посилальної цілісності NO ACTION. Призначення та особливості використання.

16. Стратегія (або механізм) контролю посилальної цілісності CASCADE. Призначення та особливості використання.

17. Стратегія (або механізм) контролю посилальної цілісності SET NULL. Призначення та особливості використання.

18. Стратегія (або механізм) контролю посилальної цілісності SET DEFAULT. Призначення та особливості використання.

19. Переваги зберігання обмежень цілісності даних на сервері баз даних.

20. Недоліки зберігання обмежень цілісності даних на сервері баз даних.

21. Які обмеження цілісності даних задаються в операторі CREATE TABLE?

22. Які обмеження цілісності даних задаються в операторі ALTER TABLE?

23. Яким чином обов'язкові дані задаються в операторі CREATE TABLE?

24. Яким чином обов'язкові дані задаються в операторі ALTER TABLE?

25. Яким чином обмеження посилальної цілісності задаються в операторі CREATE TABLE?

26. Яким чином обмеження посилальної цілісності задаються в операторі ALTER TABLE?

27. Що таке правило? Як його можна реалізувати?

# ТЕМА 8
# ВИКОРИСТАННЯ ТРАНЗАКЦІЙ НА ПРИКЛАДІ СУБД MICROSOFT SQL SERVER

## 8.1 Теоретичні відомості

### 8.1.1 Поняття транзакції

Під транзакцією розуміється неділима з точки зору дії на базу даних послідовність операторів маніпулювання даними (читання, видалення, вставки, модифікації), що призводить до одного з двох можливих результатів : або послідовність виконується, якщо усі оператори правильні, або уся транзакція відкочується, якщо хоч би один оператор не може бути успішно виконаний. Обробка транзакцій гарантує цілісність інформації у базі даних. Таким чином, транзакція переводить базу даних з одного цілісного стану в інше.

Підтримка механізму транзакцій – показник рівня розвиненості СУБД. Коректна підтримка транзакцій одночасно є основою забезпечення цілісності БД. Транзакції також складають основу ізольованості в розрахованих на багато користувачів системах, де з однією БД паралельно можуть працювати декілька користувачів або прикладних програм. Одне з основних завдань СУБД – забезпечення ізольованості, тобто створення такого режиму функціонування, при якому кожному користувачеві здавалося б, що база даних доступна тільки йому. Таке завдання СУБД прийнято називати паралелізмом транзакцій.

Більшість виконуваних дій робляться в тілі транзакцій. За умовчанням кожна команда виконується як самостійна транзакція. При необхідності користувач може явно вказати її початок і кінець, щоб мати можливість включити в неї декілька команд.

При виконанні транзакції система управління базами даних повинна дотримуватися певних правил обробки набору команд, що входять в транзакцію. Зокрема, розроблено чотири правила, відомі як вимоги ACID, вони гарантують правильність і надійність роботи системи.

### 8.1.2 Властивості транзакцій

Характеристики (або властивості) транзакцій описуються в термінах ACID (Atomicity, Consistency, Isolation, Durability – неподільність, узгодженість, ізольованість, стійкість). Розглянемо визначення цих характеристик.

Транзакція неділима в тому сенсі, що є єдиним цілим. Усі її компоненти або виконуються, або ні. Не буває часткової транзакції. Якщо може бути виконана лише частина транзакції, вона відхиляється.

Транзакція є погодженою, тому що не порушує бізнес-логіку і стосунки між елементами даних. Ця властивість дуже важлива при розробці клієнт-серверних систем, оскільки до бази даних надходить велика кількість транзакцій від різних систем і об'єктів. Якщо хоч би одна з них порушить цілісність даних, то усі інші можуть видати невірні результати.

Транзакція завжди ізольована, оскільки її результати самодостатні. Вони не залежать від попередніх або наступних транзакцій. Ця властивість називається серіалізуємістю і означає, що транзакції в послідовності незалежні.

Транзакція стійка. Після завершення транзакції результати її виконання зберігаються в системі, яку ніщо не може повернути в початковий (до початку транзакції) стан, тобто відбувається фіксація транзакції, що означає, що її дія є постійною навіть при збою системи. При цьому мається на увазі деяка форма зберігання інформації в постійній пам'яті як частина транзакції .

Вказані вище правила виконує сервер. Програміст лише вибирає потрібний рівень ізоляції, піклується про дотримання логічної цілісності даних і бізнес-правил. На нього покладаються обов'язки по створенню ефективних і логічно вірних алгоритмів обробки даних. Він вирішує, які команди повинні виконуватися як одна транзакція, а які можуть бути розбиті на декілька послідовно виконуваних транзакцій. Слід по можливості використовувати невеликі транзакції, тобто такі, що

включають якомога менше команд і змінюють мінімум даних. Дотримання цієї вимоги дозволить найбільш ефективним чином забезпечити одночасну роботу з даними багатьом користувачам.

### 8.1.3 Блокування

Підвищення ефективності роботи при використанні невеликих транзакцій пов'язане з тим, що при виконанні транзакції сервер накладає на ці дані блокування.

Блокуванням називається тимчасове обмеження на виконання деяких операцій обробки даних. Блокування може бути накладене як на окремий рядок таблиці, так і на усю базу даних. Управлінням блокуваннями на сервері займається менеджер блокувань, контролюючий їх застосування і вирішення конфліктів. Транзакції і блокування тісно пов'язані один з одним. Транзакції накладають блокування на дані, щоб забезпечити виконання вимог ACID. Без використання блокувань декілька транзакцій могли б змінювати одні і ті ж дані.

Блокування є методом управління паралельними процесами, при якому об'єкт БД не може бути модифікований без відома транзакції, тобто відбувається блокування доступу до об'єкту з боку інших транзакцій, чим виключається непередбачувана зміна об'єкту. Розрізняють два види блокування:

- блокування запису – транзакція блокує рядки в таблицях таким чином, що запит іншої транзакції до цих рядків буде скасований;
- блокування читання – транзакція блокує рядки так, що запит з боку іншої транзакції на блокування запису цих рядків буде знехтуваний, а на блокування читання – прийнятий.

У СУБД використовують протокол доступу до даних, що дозволяє уникнути проблеми паралелізму. Його суть полягає в наступному:

- транзакція, результатом дії якої на рядок даних в таблиці являється її витягання, зобов'язана накласти блокування читання на цей рядок;

- транзакція, призначена для модифікації рядка даних, накладає на неї блокування запису;
- якщо прошене блокування на рядок відкидається із-за вже наявного блокування, то транзакція переводиться в режим очікування до тих пір, поки блокування не буде знято;
- блокування запису зберігається аж до кінця виконання транзакції.

Вирішення проблеми паралельної обробки даних в базі даних полягає в тому, що рядки таблиць блокуються, а наступні транзакції, які модифікують ці рядки, відкидаються і переводяться в режим очікування. У зв'язку з властивістю збереження цілісності бази даних, транзакції є відповідними одиницями ізольованості користувачів. Дійсно, якщо кожен сеанс взаємодії з базою даних реалізується транзакцією, то користувач починає з того, що звертається до погодженого стану бази даних, тобто стану, в якому вона могла б знаходитися, навіть якщо б користувач працював з нею самостійно, і в якому гарантовано виконуються усі вимоги цілісності даних.

Якщо в системі управління базами даних не реалізовані механізми блокування, то при одночасному читанні і зміні одних і тих же даних декількома користувачами можуть виникнути наступні проблеми одночасного доступу:

- проблема останньої зміни виникає, коли декілька користувачів змінюють один і той же рядок, ґрунтуючись на її початковому значенні; тоді частина даних буде втрачена, оскільки кожна наступна транзакція перезапише зміни, зроблені попередньою. Вихід з цієї ситуації полягає в послідовному внесенні змін;
- проблема «брудного» читання можлива у тому випадку, якщо користувач виконує складні операції обробки даних, що вимагають множинної зміни даних перед тим, як вони набудуть логічно вірного стану. Якщо під час зміни даних інший користувач прочитуватиме їх, то може виявитися, що він отримає логічно невірну інформацію. Для виключення

подібних проблем необхідно робити зчитування даних після закінчення усіх змін;

- проблема неповторюваного читання є наслідком неодноразового зчитування транзакцією одних і тих же даних. Під час виконання першої транзакції інша може внести в ці зміни, тому при повторному читанні перша транзакція отримає вже інший набір даних, що призводить до порушення їх цілісності або логічної неузгодженості;
- проблема читання фантомів з'являється після того, як одна транзакція вибирає дані з таблиці, а інша вставляє або видаляє рядки до завершення першої. Вибрані з таблиці значення будуть некоректні.

Для вирішення перерахованих проблем в спеціально розробленому стандарті визначено чотири рівні блокування. Рівень ізоляції транзакції визначає, чи можуть інші (що конкурують) транзакції вносити зміни в дані, змінені поточною транзакцією, а також чи може поточна транзакція бачити зміни, зроблені конкуруючими транзакціями, і навпаки. Кожен наступний рівень підтримує вимоги попереднього і накладає додаткові обмеження:

- рівень 0 – заборона «забруднення» даних. Цей рівень вимагає, щоб змінювати дані могла тільки одна транзакція ; якщо іншій транзакції необхідно змінити ті ж дані, вона повинна чекати завершення першої транзакції ;
- рівень 1 – заборона «брудного» читання. Якщо транзакція почала зміну даних, то ніяка інша транзакція не зможе прочитати їх до завершення першої;
- рівень 2 – заборона неповторюваного читання. Якщо транзакція прочитує дані, то ніяка інша транзакція не зможе їх змінити. Таким чином, при повторному читанні вони знаходитимуться в первинному стані;
- рівень 3 – заборона фантомів. Якщо транзакція звертається до даних, то ніяка інша транзакція не зможе додати нові або видалити рядки, які можуть бути враховані при виконанні транзакції. Реалізація цього рівня блокування виконується шляхом використання блокувань діапазону

ключів. Схоже блокування накладається не на конкретні рядки таблиці, а на рядки, що задовольняють певній логічній умові.

### 8.1.4 Управління транзакціями

Під управлінням транзакціями розуміється здатність управляти різними операціями над даними, які виконуються усередині реляційної СУБД. Передусім, мається на увазі виконання операторів INSERT, UPDATE і DELETE. Наприклад, після створення таблиці (виконання оператора CREATE TABLE) не треба фіксувати результат: створення таблиці фіксується у базі даних автоматично. Так само за допомогою відміни транзакції не вдасться відновити тільки що видалену оператором DROP TABLE таблицю.

Після успішного виконання команд, поміщених в тіло однієї транзакції, негайної зміни даних не відбувається. Для остаточного завершення транзакції існують так звані команди управління транзакціями, за допомогою яких можна або зберегти у базі даних усі зміни, що сталися в ході її виконання, або повністю їх відмінити.

Існують три команди, які використовуються для управління транзакціями:

- COMMIT – для збереження змін;
- ROLLBACK – для відміни змін;
- SAVEPOINT – для установки особливих точок повернення.

Після завершення транзакції уся інформація про зроблені зміни зберігається або в спеціально виділеній оперативній пам'яті, або в тимчасовій області відкату в самій базі даних до тих пір, поки не буде виконана одна з команд управління транзакціями. Потім усі зміни або фіксуються у базі даних, або відкидаються, а тимчасова область відкату звільняється.

Команда COMMIT призначена для збереження у базі даних усіх змін, що сталися в ході виконання транзакції. Вона зберігає результати усіх

операцій, які мали місце після виконання останньої команди COMMIT або ROLLBACK .

Команда ROLLBACK призначена для відміни транзакцій, ще не збережених у базі даних. Вона відміняє тільки ті транзакції, які були виконані з моменту видачі останньої команди COMMIT або ROLLBACK .

Команда SAVEPOINT (точка збереження) призначена для установки в транзакції особливих точок, куди надалі може бути зроблений відкат (при цьому відкату всієї транзакції не відбувається). Команда має наступний вигляд:

SAVEPOINT ім'я_точки_збереження

Вона служить виключно для створення точок збереження серед операторів, призначених для зміни даних. Ім'я точки збереження в пов'язаній з нею групі транзакцій має бути унікальним.

Для відміни дії групи транзакцій, обмежених точками збереження, використовується команда ROLLBACK з наступним синтаксисом:

ROLLBACK TO ім'я_точки_збереження

Оскільки за допомогою команди SAVEPOINT велике число транзакцій може бути розбите на менші і тому більше керовані групи, її застосування є одним із способів управління транзакціями.

### 8.1.5 Управління транзакціями в середовищі Microsoft SQL Server

#### 8.1.5.1 Визначення транзакцій

Microsoft SQL Server пропонує багато засобів управління поведінкою транзакцій. Користувачі в основному повинні вказувати тільки почало і кінець транзакції, використовуючи команди SQL або API (прикладного інтерфейсу програмування). Транзакція визначається на рівні з'єднання з базою даних і при закритті з'єднання автоматично закривається. Якщо користувач спробує встановити з'єднання знову і продовжити виконання транзакції, то це йому не вдасться. Коли транзакція починається, усі

команди, виконані в з'єднанні, вважаються тілом однієї транзакції, поки не буде досягнутий її кінець.

SQL Server підтримує три види визначення транзакцій:

- явне;
- автоматичне;
- неявний (що мається на увазі).

За умовчанням SQL Server працює в режимі автоматичного початку транзакцій, коли кожна команда розглядається як окрема транзакція. Якщо команда виконана успішно, то її зміни фіксуються. Якщо при виконанні команди сталася помилка, то зроблені зміни відміняються і система повертається в первинний стан.

Коли користувачеві знадобиться створити транзакцію, що включає декілька команд, він повинен явно вказати транзакцію.

Сервер працює тільки в одному з двох режимів визначення транзакцій: автоматичному або такому, що мається на увазі. Він не може знаходитися в режимі виключно явного визначення транзакцій. Цей режим працює поверх двох інших.

Для установки режиму автоматичного визначення транзакцій використовується команда:

```
SET IMPLICIT_TRANSACTIONS  OFF
```

При роботі в режимі неявного (що мається на увазі) початку транзакцій SQL Server автоматично починає нову транзакцію, як тільки завершена попередня. Установка режиму визначення транзакцій, що мається на увазі, виконується за допомогою іншої команди:

```
SET IMPLICIT_TRANSACTIONS  ON
```

**8.1.5.2 Явні транзакції**

Явні транзакції вимагають, щоб користувач вказав початок і кінець транзакції, використовуючи наступні команди:

- початок транзакції: в журналі транзакцій фіксуються первинні значення змінюваних даних і момент початку транзакції;

```
BEGIN TRAN[SACTION]
        [ім'я_транзакції |
        @ім'я_змінної_транзакції
 [WITH MARK ['опис_транзакції']]]
```

- кінець транзакції: якщо в тілі транзакції не було помилок, то ця команда наказує серверу зафіксувати усі зміни, зроблені в транзакції, після чого в журналі транзакцій позначається, що зміни зафіксовані і транзакція завершена;

```
COMMIT [TRAN[SACTION]
        [ім'я_транзакції |
    @ім'я_змінної_транзакції]]
```

- створення усередині транзакції точки збереження : СУБД зберігає стан БД в поточній точці і привласнює збереженому стану ім'я точки збереження;

```
SAVE TRAN[SACTION]
   {ім'я_точки_сохранеия |
   @ім'я_змінної_точки_збереження}
```

переривання транзакції: коли сервер зустрічає цю команду, відбувається відкат транзакції, відновлюється первинний стан системи і в журналі транзакцій відмічається, що транзакція була скасована. Приведена нижче команда відміняє усі зміни, зроблені у базі даних після оператора BEGIN TRANSACTION або відміняє зміни, зроблені у базі даних після точки збереження, повертаючи транзакцію до місця, де був виконаний оператор SAVE TRANSACTION.

```
ROLLBACK [TRAN[SACTION]
  [ім'я_транзакції |
  @ім'я_змінної_транзакції
  | ім'я_точки_збереження
  |@ім'я_змінної_точки_збереження]]
```

При цьому часто застосовують функцію @@TRANCOUNT, що повертає кількість активних транзакцій та функцію @@NESTLEVEL, що повертає рівень вкладеності транзакцій.

**8.1.5.3 Вкладені транзакції**

Вкладеними називаються транзакції, виконання яких ініціюється з тіла вже активній транзакції.

Для створення вкладеної транзакції користувачеві не потрібні які-небудь додаткові команди. Він просто починає нову транзакцію, не закривши попередню. Завершення транзакції верхнього рівня відкладається до завершення вкладених транзакцій. Якщо транзакція самого нижнього (вкладеного) рівня завершена невдало і скасована, то всі транзакції верхнього рівня, включаючи транзакцію першого рівня, будуть скасовані. Крім того, якщо декілька транзакцій нижнього рівня було завершено успішно (але не зафіксовані), проте на середньому рівні (не сама верхня транзакція) невдало завершилася інша транзакція, то відповідно до вимог ACID станеться відкат всіх транзакцій усіх рівнів, включаючи успішно завершені. Тільки коли усі транзакції на усіх рівнях завершені успішно, відбувається фіксація усіх зроблених змін в результаті успішного завершення транзакції верхнього рівня.

Кожна команда COMMIT TRANSACTION працює тільки з останньою початою транзакцією. При завершенні вкладеної транзакції команда COMMIT застосовується до найбільш «глибокої» вкладеної транзакції. Навіть якщо в команді COMMIT TRANSACTION вказано ім'я

транзакції більше високого рівня, буде завершена транзакція, почата останньою.

Якщо команда ROLLBACK TRANSACTION використовується на будь-якому рівні вкладеності без вказівки імені транзакції, то відкочуються усі вкладені транзакції, включаючи транзакцію самого високого рівня. У команді ROLLBACK TRANSACTION дозволяється вказувати тільки ім'я самої верхньої транзакції. Імена будь-яких вкладених транзакцій ігноруються, і спроба їх вказівки приведе до помилки. Таким чином, при відкаті транзакції будь-якого рівня вкладеності завжди відбувається відкат усіх транзакцій. Якщо ж вимагається відкотити лише частину транзакцій, можна використовувати команду SAVE TRANSACTION, за допомогою якої створюється точка збереження.

### 8.1.6 Блокування в середовищі Microsoft SQL Server

#### 8.1.6.1 Управління блокуваннями

Користувачеві найчастіше не треба робити ніяких дій з управління блокуваннями. Усю роботу по установці, зняттю і вирішенню конфліктів виконує спеціальний компонент сервера, званий менеджером блокувань. MS SQL Server підтримує різні рівні блокування об'єктів (чи деталізацію блокувань), починаючи з окремого рядка таблиці і закінчуючи базою даних в цілому. Менеджер блокувань автоматично оцінює, яку кількість даних необхідно блокувати, і встановлює відповідний тип блокування. Це дозволяє підтримувати рівновагу між продуктивністю роботи системи блокування і можливістю користувачів діставати доступ до даних. Блокування на рівні рядка дозволяє найточніше управляти таким доступом, оскільки блокуються тільки дійсно змінювані рядки. Безліч користувачів можуть одночасно працювати з даними з мінімальними затримками. Платою за це є збільшення числа операцій установки і зняття блокувань, а також велика кількість службової інформації, яку доводиться зберігати для відстежування встановлених блокувань. При блокуванні на рівні таблиці продуктивність системи блокування різко збільшується,

оскільки необхідно встановити лише одне блокування і зняти її тільки потім завершення транзакції. Користувач при цьому має максимальну швидкість доступу до даних. В той же час вони не доступні нікому іншому, тому що уся таблиця заблокована. Доводиться чекати, доки поточний користувач завершить роботу.

Дії, що виконуються користувачами при роботі з даними, зводяться до операцій двох типів: їх читанню і зміні. У операції по зміні включаються дії з додавання, видалення і власне зміни даних. Залежно від виконуваних дій сервер накладає певний тип блокування з наступного переліку:

- Колективні блокування. Вони накладаються при виконанні операцій читання даних (наприклад, SELECT). Якщо сервер встановив на ресурс колективне блокування, то користувач може бути упевнений, що вже ніхто не зможе змінити ці дані.
- Блокування оновлення. Якщо на ресурс встановлено колективне блокування і для цього ресурсу встановлюється блокування оновлення, то ніяка транзакція не зможе накласти колективне блокування або блокування оновлення .
- Монопольне блокування. Цей тип блокувань використовується, якщо транзакція змінює дані. Коли сервер встановлює монопольне блокування на ресурс, то ніяка інша транзакція не може прочитати або змінити заблоковані дані. Монопольне блокування не сумісне ні з якими іншими блокуваннями, і ні одне блокування, включаючи монопольне, не може бути накладене на ресурс.
- Блокування масивного оновлення. Накладається сервером при виконанні операцій масивного копіювання в таблицю і забороняє звернення до таблиці будь-яким іншим процесам. В той же час декілька процесів, що виконують масивне копіювання, можуть одночасно вставляти рядки в таблицю.

Окрім зазначених основних типів блокувань SQL Server підтримує ряд спеціальних блокувань, призначених для підвищення продуктивності і

функціональності обробки даних. Вони називаються блокуваннями намірів і використовуються сервером у тому випадку, якщо транзакція має намір отримати доступ до даних вниз за ієрархією і для інших транзакцій необхідно встановити заборону на накладення блокувань, які конфліктуватимуть з блокуванням, першої транзакції, що накладається .

Раніше розглянуті блокування відносяться до даних. Окрім перерахованих в середовищі SQL Server існує два інші типи блокувань: блокування діапазону ключів і блокування схеми (метаданих, що описують структуру об'єкту).

Блокування діапазону ключів вирішує проблему виникнення фантомів і забезпечує вимоги серіалізуємості транзакції. Блокування цього типу встановлюються на діапазон рядків, що відповідають певній логічній умові, за допомогою якої здійснюється вибірка даних з таблиці.

Блокування схеми використовується при виконанні команд модифікації структури таблиць для забезпечення цілісності даних.

**8.1.6.2 «Мертві» блокування**

«Мертві», або тупикові, блокування характерні для розрахованих на багато користувачів систем. «Мертве» блокування виникає, коли дві транзакції блокують два блоки даних і для завершення будь-якої з них потрібний доступ до даних, заблокованих раніше іншою транзакцією. Для завершення кожної транзакції необхідно дочекатися, поки блокована іншою транзакцією частина даних буде розблокована. Але це неможливо, оскільки друга транзакція чекає розблокування ресурсів, використовуваних першою.

Без застосування спеціальних механізмів виявлення і зняття «мертвих» блокувань нормальна робота транзакцій буде порушена. Якщо в системі встановлений нескінченний період очікування завершення транзакції (а це задано за умовчанням), то при виникненні «мертвого» блокування для двох транзакцій цілком можливо, що, чекаючи звільнення заблокованих ресурсів, у безвиході виявляться і нові транзакції. Щоб

уникнути подібних проблем, в середовищі MS SQL Server реалізований спеціальний механізм вирішення конфліктів тупикового блокування.

Для цих цілей сервер знімає одне з блокувань, що викликали конфлікт, і відкочує транзакцію, що ініціалізувала її. При виборі блокування, якому необхідно пожертвувати, сервер виходить з міркувань мінімальної вартості.

Повністю уникнути виникнення «мертвих» блокувань не можна. Хоча сервер і має ефективні механізми зняття таких блокувань, все ж при написанні додатків слід враховувати вірогідність їх виникнення і робити усі можливі дії для попередження цього. «Мертві» блокування можуть істотно понизити продуктивність, оскільки системі потрібно досить багато часу для їх виявлення, відкату транзакції і повторного її виконання.

Для мінімізації можливості утворення «мертвих» блокувань при розробці коду транзакції слід дотримуватися наступних правил:

- виконувати дії з обробки даних в постійному порядку, щоб не створювати умови для захоплення одних і тих же даних;
- уникати взаємодії з користувачем в тілі транзакції ;
- мінімізувати тривалість транзакції і виконувати її по можливості в одному пакеті;
- застосовувати як можна нижчий рівень ізоляції.

### 8.1.6 Рівні ізоляції в середовищі Microsoft SQL Server

Рівень ізоляції визначає міру незалежності транзакцій один від одного. Найвищим рівнем ізоляції є серіалізуємість, що забезпечує повну незалежність транзакцій один від одного. Кожен наступний рівень відповідає вимогам усіх попередніх і забезпечує додатковий захист транзакцій. SQL Server підтримує усі чотири рівні ізоляції, визначені стандартом ANSI. Рівень ізоляції встановлюється командою:

```
SET TRANSACTION ISOLATION LEVEL
        { READ UNCOMMITTED | READ COMMITTED
          | REPEATABLE READ | SERIALIZABLE }
```

- READ UNCOMMITED – незавершене читання, або допустиме чорнове читання. Нижчий рівень ізоляції, що відповідає рівню 0. Він гарантує тільки фізичну цілісність даних : якщо декілька користувачів одночасно змінюють один і той же рядок, то в остаточному варіанті рядок матиме значення, визначене користувачем, що останнім змінив запис. По суті, для транзакції не встановлюється ніякого блокування, яке гарантувало б цілісність даних. Для установки цього рівня використовується команда:

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

- READ COMMITTED – завершене читання, при якому відсутнє чорнове, «брудне» читання. Проте в процесі роботи однієї транзакції інша може бути успішно завершена і зроблені нею зміни зафіксовані. У результаті перша транзакція працюватиме з іншим набором даних. Це проблема неповторюваного читання . Цей рівень ізоляції встановлений в SQL Server за умовчанням і встановлюється за допомогою команди:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED

- REPEATABLE READ – читання, що повторюється. Повторне читання рядка поверне спочатку лічені дані, незважаючи на будь-які оновлення, зроблені іншими користувачами до завершення транзакції. Проте на цьому рівні ізоляції можливе виникнення фантомів. Його установка реалізується командою:

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

- SERIALIZABLE – серіалізуємість. Читання заборонене до завершення транзакції. Це максимальний рівень ізоляції, який забезпечує повну ізоляцію транзакцій один від одного. Він встановлюється командою:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

У кожен момент часу можливий тільки один рівень ізоляції.

## 8.2 Практичне опанування теми 8

### 8.2.1 Передумови виконання завдань теми 8

При виконанні практичних завдань теми 8 передбачається використання застосунку SQL Server Management Studio, який входить до складу інтегрованої платформи Microsoft SQL Server. При цьому передбачається використання СУБД Microsoft SQL Server версії 2008 або вищої. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Увага! При виконанні практичних завдань цієї теми рекомендується використовувати окрему базу даних. Її можна спеціально створити, використовуючи запити, створені при виконанні практичних завдань теми «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

### 8.2.2 Основи використання транзакцій

#### 8.2.2.1 Додавання даних в одну таблицю

Розглянемо послідовність дій під час створення та використання запиту, за допомогою якого запускається транзакція, до таблиці «Supplied» додається новий запис, а потім імітується ситуація некоректного чи коректного завершення транзакції. Стан таблиці контролюється до початку транзакції, у процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати таку послідовність дій.

1. Натиснути кнопку New Query на панелі інструментів.

2. Ввести текст запиту, наведений рисунку 8.1.

3. Виконати запит. У разі успішного виконання запиту на екран будуть виведені дані, що ілюструють стан таблиці до початку транзакції, у процесі виконання транзакції та після завершення транзакції (рисунок 8.2).

Як видно з наведених даних, новий запис у таблиці з'являється, а потім зникає.

```
USE dlvr
SELECT Supplied.ContractNumber, Supplied.Product, Supplied.Amount, Supplied.PricePerItem,
       Suppliers.SupplierName, Contracts.ContractDate
    FROM Suppliers INNER JOIN Contracts ON Suppliers.SupplierID = Contracts.SupplierID
                   INNER JOIN Supplied ON Contracts.ContractNumber = Supplied.ContractNumber
    WHERE Supplied.ContractNumber=1
BEGIN TRANSACTION
   INSERT INTO Supplied VALUES (1,'кавоварка',22,389.75)
   SELECT Supplied.ContractNumber, Supplied.Product, Supplied.Amount, Supplied.PricePerItem,
          Suppliers.SupplierName, Contracts.ContractDate
       FROM Suppliers INNER JOIN Contracts ON Suppliers.SupplierID = Contracts.SupplierID
                      INNER JOIN Supplied ON Contracts.ContractNumber = Supplied.ContractNumber
       WHERE Supplied.ContractNumber=1
   ROLLBACK
SELECT Supplied.ContractNumber, Supplied.Product, Supplied.Amount, Supplied.PricePerItem,
       Suppliers.SupplierName, Contracts.ContractDate
    FROM Suppliers INNER JOIN Contracts ON Suppliers.SupplierID = Contracts.SupplierID
                   INNER JOIN Supplied ON Contracts.ContractNumber = Supplied.ContractNumber
    WHERE Supplied.ContractNumber=1
```

Рисунок 8.1

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | кавоварка | 22 | 389.75 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 6 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

Рисунок 8.2

4. Тепер розглянемо ситуацію коректного завершення транзакції. Для цього в наведеному на рисунку 8.1 тексті запиту змінимо оператор ROLLBACK на COMMIT. Виконаємо запит. Результат наведено на рисунку 8.3. Запит можна зберегти під назвою SQLQuery_trans.sql

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | кавоварка | 22 | 389.75 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 6 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

| | ContractNumber | Product | Amount | PricePerItem | SupplierName | ContractDate |
|---|---|---|---|---|---|---|
| 1 | 1 | відеомагнітофон | 10 | 100.00 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 2 | 1 | кавоварка | 22 | 389.75 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 3 | 1 | комп'ютер | 24 | 1554.22 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 4 | 1 | магнітофон | 35 | 655.12 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 5 | 1 | стереосистема | 12 | 220.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |
| 6 | 1 | телевізор | 10 | 1253.45 | ПП Іваненко І.І | 1999-09-01 00:00:00.000 |

Рисунок 8.3

### 8.2.2.2 Додавання даних до декількох таблиць

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, потім створюється новий постачальник, з цим постачальником укладається договір на поставку, за цим договором поставляється продукція. Імітується ситуація некоректного чи коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, у процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати таку послідовність дій.

1. Натиснути кнопку New Query на панелі інструментів.

2. Ввести текст запиту, наведений на рисунку 8.4.

```
USE dlvr
SELECT * FROM Suppliers
SELECT * FROM Contracts
SELECT * FROM Supplied

BEGIN TRANSACTION
   INSERT INTO Suppliers VALUES (6,'ПП Шевченко Г.С.','м.Суми, вул. Шевченко, 56')
   INSERT INTO Contracts (ContractDate, SupplierID, ContractName, Comment) VALUES ('20021212',6,'','')
   DECLARE @newContractID int
   SET @newContractID=@@IDENTITY
   INSERT INTO Supplied VALUES (@newContractID,'кавоварка',22, 389.75)
   INSERT INTO Supplied VALUES (@newContractID,'пилосос',33, 189.44)
   SELECT * FROM Suppliers
   SELECT * FROM Contracts
   SELECT * FROM Supplied
   ROLLBACK

SELECT * FROM Suppliers
SELECT * FROM Contracts
SELECT * FROM Supplied
```

Рисунок 8.4

3. Виконати запит.

4. У разі успішного виконання запиту на екран будуть виведені дані, що ілюструють стан таблиць до початку транзакції, у процесі виконання транзакції та після завершення транзакції (аналогічно попередньому запиту). Як видно з цих даних, нові записи в таблицях з'являються, а потім зникають. Приклади відображення даних тут не наведено тому, що вони досить громіздкі.

5. Тепер розглянемо ситуацію коректного завершення транзакції. Для цього в наведеному тексті запиту змінимо оператора ROLLBACK на COMMIT. Виконаємо запит. В результаті виконання запиту дані мають бути внесені до таблиць та збережені. У цьому необхідно переконатися, відкривши відповідні таблиці як перегляду даних. Запит можна зберегти під назвою SQLQuery_trans1.sql.

### 8.2.2.3 Зміна даних у декількох таблицях

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, потім змінюються дані, введені в таблиці під час попереднього запиту, тобто ціна усіх товарів, які надійшли від нового постачальника, збільшується на 10%. Імітується ситуація некоректного чи коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, у процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати таку послідовність дій.

1. Для відносин посилальної цілісності між усіма таблицями бази даних встановити механізм Cascade.

2. Натиснути кнопку New Query на панелі інструментів та вести текст запиту, наведений рисунку 8.5.

```
USE dlvr
SELECT * FROM Suppliers
SELECT * FROM Contracts WHERE SupplierID=6
SELECT * FROM Supplied WHERE ContractNumber IN (SELECT ContractNumber FROM Contracts WHERE SupplierID=6)

BEGIN TRANSACTION
   UPDATE Suppliers SET SupplierID=22 WHERE SupplierID=6
   UPDATE Supplied SET PricePerItem=PricePerItem*1.1
                  WHERE ContractNumber IN (SELECT ContractNumber FROM Contracts WHERE SupplierID=22)
   SELECT * FROM Suppliers
   SELECT * FROM Contracts WHERE SupplierID=22
   SELECT * FROM Supplied WHERE ContractNumber IN (SELECT ContractNumber FROM Contracts WHERE SupplierID=22)
   ROLLBACK

SELECT * FROM Suppliers
SELECT * FROM Contracts WHERE SupplierID=6
SELECT * FROM Supplied WHERE ContractNumber IN (SELECT ContractNumber FROM Contracts WHERE SupplierID=6)
```

Рисунок 8.5

3. Виконати запит.

4. У разі успішного виконання запиту на екран будуть виведені дані, що ілюструють стан таблиць до початку транзакції, у процесі виконання транзакції та після завершення транзакції (аналогічно попереднім запитам). Як видно з цих даних, зміни даних у таблицях з'являються, а потім зникають. Приклади відображення даних тут не наведено тому, що вони досить громіздкі.

5. Тепер розглянемо ситуацію коректного завершення транзакції. Для цього в наведеному тексті запиту змінимо оператор ROLLBACK на COMMIT. Виконаємо запит. В результаті виконання запиту дані мають бути внесені до таблиць та збережені. У цьому необхідно переконатися, відкривши відповідні таблиці як перегляду даних. Запит можна зберегти під назвою SQLQuery_trans2.sql.

**8.2.2.3 Видалення даних у декількох таблицях**

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, в рамках якої видаляється постачальник, створений при виконанні запиту з розділу 8.2.2.2 і дані якого були змінені при виконанні запиту з розділу 8.2.2.3. З урахуванням використовуваного механізму контролю цілісності (Cascade) дані будуть видалені в декількох таблицях. Імітується ситуація некоректного чи коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, у процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати таку послідовність дій.

1. Натиснути кнопку New Query на панелі інструментів.

2. Ввести текст запиту, наведений на рисунку 8.6.

```
USE dlvr
SELECT * FROM Suppliers
SELECT * FROM Contracts
SELECT * FROM Supplied

BEGIN TRANSACTION
    DELETE FROM Suppliers WHERE SupplierID=6

    SELECT * FROM Suppliers
    SELECT * FROM Contracts
    SELECT * FROM Supplied
    ROLLBACK

SELECT * FROM Suppliers
SELECT * FROM Contracts
SELECT * FROM Supplied
```

Рисунок 8.6

3. Виконати запит.

4. У разі успішного виконання запиту на екран будуть виведені дані, що ілюструють стан таблиць до початку транзакції, у процесі виконання транзакції та після завершення транзакції (аналогічно попереднім запитам). Як видно з цих даних, зміни даних у таблицях з'являються, а потім зникають. Приклади відображення даних тут не наведено тому, що вони досить громіздкі.

5. Тепер розглянемо ситуацію коректного завершення транзакції. Для цього в наведеному тексті запиту змінимо оператор ROLLBACK на COMMIT. Виконаємо запит. В результаті виконання запиту дані мають бути внесені до таблиць та збережені. У цьому необхідно переконатися, відкривши відповідні таблиці як перегляду даних. Запит можна зберегти під назвою SQLQuery_trans3.sql.

### 8.2.3 Рівні ізоляції транзакцій та їх особливості

Оскільки далі будуть розглядатися ситуації, які можуть виникнути лише при роботі з базою даних декількох користувачів одночасно, виникає потреба виконувати ці дії або у мережевому оточенні, або у режимі імітації роботи декількох користувачів на одному комп'ютері. Таку імітацію зробити досить легко. Для цього треба запустити два або більше (рисунок 8.7) екземплярів застосунку SQL Server Management Studio (SSMS), кожен з яких буде працювати з тією ж самою базою даних. При цьому кожен екземпляр SSMS буде імітувати роботу з базою даних окремого користувача.

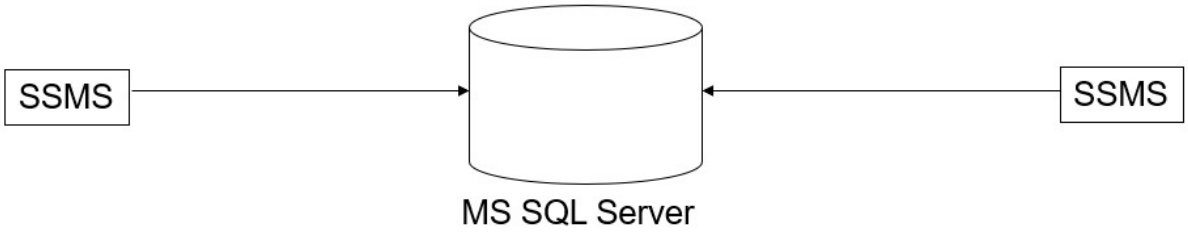



Рисунок 8.7

Далі ці екземпляри SSMS будемо умовно називати першим та другим користувачем.

### 8.2.3.1 Використання рівнів ізоляції READ UNCOMMITTED та READ COMMITTED

Нехай першим користувачем створено запит, який наведено на рисунку 8.8. Другим користувачем створено запит, який наведено на рисунку 8.9. Основною метою при цьому є створення ситуації, коли в процесі виконання транзакції, яку було ініційовано одним користувачем, до тих же самих даних звертається транзакція, яку було ініційовано іншим користувачем.

```
USE delivery

BEGIN TRANSACTION
  UPDATE Contracts SET Comment='Підстава
         WHERE ContractNumber=5
  SELECT * FROM Contracts

  WAITFOR DELAY '00:00:15'

  ROLLBACK TRAN
```

Рисунок 8.8

```
USE delivery

SET TRANSACTION ISOLATION LEVEL READ COMM
--SET TRANSACTION ISOLATION LEVEL READ UN
BEGIN TRANSACTION
  SELECT * FROM Contracts
COMMIT TRAN
```

Рисунок 8.9

Отже, перший користувач ініціює виконання запиту, в рамках якого здійснюється модифікація даних, яку потім скасовують. Між модифікацією даних та скасуванням результатів модифікації робиться пауза, тобто виконання запиту буде призупинено на 15 секунд. Цього часу досить для того, щоб встигнути переключитися на другого користувача та запустити

на виконання другий запит. Залежно від того, який рівень ізоляції було вказано (READ UNCOMMITTED або READ COMMITTED) можна спостерігати наявність або відсутність такого явища, як «брудне» читання.

Запити можна зберегти під назвою SQLQuery_trans4.sql та SQLQuery_trans5.sql відповідно.

### 8.2.3.2 Використання рівня ізоляції REPEATABLE READ

Розглянемо приклад, у якому в рамках транзакції треба забезпечити виконання умови повторюваності читання. Нехай першим користувачем створено запит, який наведено на рисунку 8.10. У цьому запиті є два оператори SELECT, які виконуються через певний час, але результати їх виконання повинні бути однаковими. У той же час другий користувач намагається змінити ті ж самі дані. Запит, який створено другим користувачем, наведено на рисунку 8.11.

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION
   SELECT ContractNumber, SUM(Amount) AS Amount, SUM(Amount*PriceP
       FROM Supplied
       GROUP BY ContractNumber

   WAITFOR DELAY '00:00:15'

   SELECT ContractNumber, SUM(Amount) AS Amount, SUM(Amount*PriceP
       FROM Supplied
       GROUP BY ContractNumber
```

Рисунок 8.10

```
BEGIN TRANSACTION
   SELECT ContractNumber, SUM(Amount) AS Amount, SUM(Amount*PriceP
       FROM Supplied
       GROUP BY ContractNumber
   UPDATE Supplied SET Amount=10,PricePerItem=100
          WHERE ContractNumber=1 AND Product='відеомагнітофон'
   SELECT ContractNumber, SUM(Amount) AS Amount, SUM(Amount*PriceP
       FROM Supplied
```

Рисунок 8.11

На рисунку 8.12 показано, яким чином будуть бачити агреговані дані перший та другий користувачі. У разі застосування рівня ізоляції REPEATABLE READ другий користувач буде чекати, коли закінчить працювати перший користувач. При цьому перший користувач отримає два однакові результати при виконанні двох операторів SELECT, тобто умова повторюваності читання буде виконана. Другий користувач отримає два різних результати агрегування даних (до та після модифікації даних відповідно), але при цьому від буде змушений дочекатися закінчення транзакції, яку ініціював перший користувач.

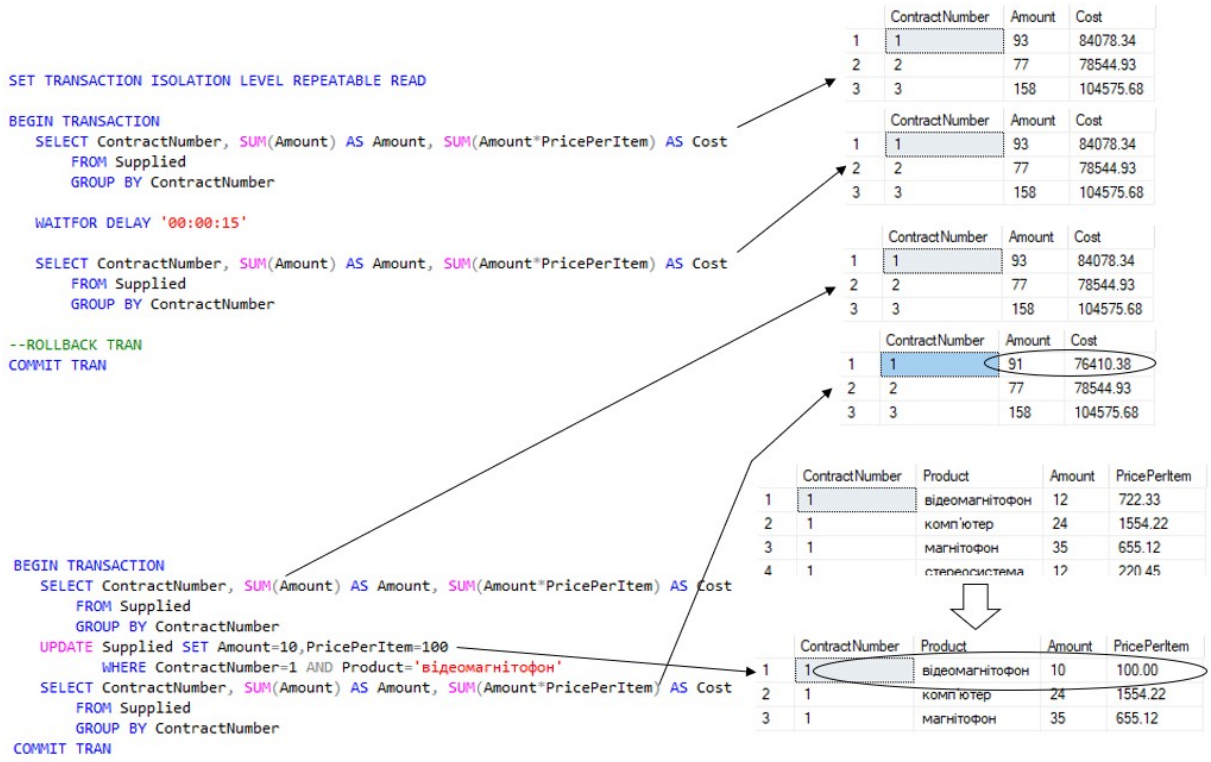



Рисунок 8.12

Бажано також проаналізувати, що зміниться, якщо замість рівня ізоляції REPEATABLE READ будуть використані рівні ізоляції READ UNCOMMITTED та READ COMMITTED.

Запити можна зберегти під назвою SQLQuery_trans6.sql та SQLQuery_trans7.sql відповідно.

Наведені вище приклади застосування транзакцій є досить простими. Більш детальну інформацію щодо побудови та використання збережених процедур засобами мови Transact-SQL, можна, наприклад, отримати за посиланнями:

https://learn.microsoft.com/ru-ru/sql/t-sql/language-elements/begin-transaction-transact-sql?view=sql-server-ver16

https://www.sqlservertutorial.net/sql-server-basics/sql-server-transaction/

### 8.2.4 Звітність про виконання практичних завдань теми 8

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) для кожного з реалізованих запитів навести призначення, текст запиту та результат;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка - 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 8.3 Питання для самоперевірки по темі 8

1. Наведіть визначення транзакції.
2. Властивості транзакцій.
3. Як розшифрувати абревіатуру ACID?
4. Що таке блокування?
5. Які існують види блокувань?
6. Яким чином уникнути проблеми паралелізму?
7. Що таке проблема останньої зміни? Коли вона виникає?
8. Що таке проблема «брудного» читання? Коли вона виникає?
9. Що таке проблема неповторюваного читання? Коли вона виникає?
10. Що таке проблема читання фантомів? Коли вона виникає?
11. Команди SQL для управління транзакціями.
12. Команда COMMIT, призначення та використання.
13. Команда ROLLBACK, призначення та використання.
14. Команда SAVEPOINT, призначення та використання.
15. Які види визначення транзакцій підтримує SQL Server?
16. Яким чином визначаються явні транзакції?
17. Яким чином визначаються неявні транзакції?

18. Які команди мови Transact-SQL автоматично запускають транзакції?

19. Що таке журнали транзакцій?

20. Які функції підтримує журнал транзакцій?

21. Для чого та яким чином здійснюється іменування транзакцій?

22. Які особливості є характерними для вкладених транзакцій?

23. Які особливості блокувань в середовищі MS SQL Server?

24. Що таке «мертві» блокування?

25. Що таке ізоляція транзакцій?

26. Назвіть основні рівні ізоляції транзакцій.

27. Рівень ізоляції транзакцій Read uncommitted. Призначення та особливості застосування.

28. Рівень ізоляції транзакцій Read committed. Призначення та особливості застосування.

29. Рівень ізоляції транзакцій Repeatable read. Призначення та особливості застосування.

30. Рівень ізоляції транзакцій Serializable. Призначення та особливості застосування.

31. Що значить термін OLTP?

32. Для чого використовуються інформаційні системи класу OLTP?

33. Які переваги та недоліки притаманні інформаційним системам класу OLTP?

# ТЕМА 9
# РЕАЛІЗАЦІЯ ПРОСТОГО КЛІЄНТСЬКОГО ЗАСТОСУНКУ ДЛЯ РОБОТИ З БАЗОЮ ДАНИХ MICROSOFT SQL SERVER

## 9.1 Теоретичні відомості

### 9.1.1 SQL і прикладне програмне забезпечення

Мову SQL можна використовувати як в інтерактивному режимі, так і шляхом впровадження його операторів в програми, написані на процедурних мовах високого рівня. Прикладом інтерактивного використання SQL-операторів є вікно Query Analyzer в середовищі MS SQL Server. Застосування ж мови SQL в прикладних програмах на практиці реалізоване двома різними способами:

1. Впроваджені SQL-оператори. Окремі SQL-оператори впроваджуються прямо в початковий текст програми і змішуються з операторами базової мови. Цей підхід дозволяє створювати програми, що звертаються безпосередньо до бази даних. Спеціальні програми-передкомпілятори перетворять початковий текст з метою заміни SQL-операторів відповідними викликами підпрограм СУБД, потім він компілюється і збирається звичайним способом.

2. Використання прикладного інтерфейсу програмування (Application Programming Interface, API). Альтернативний варіант полягає в наданні програмісту стандартного набору функцій, до яких можна звертатися із створюваних ним програм. Конкретний варіант API може надавати той же набір функціональних можливостей, який існує при підключенні вбудованих операторів, проте при цьому усувається необхідність передкомпіляції початкового тексту. Крім того, деякі розробники вказують, що в цьому випадку використовується зрозуміліший інтерфейс і створений програмний текст зручніший з точки зору його супроводу.

Обидва способи припускають використання операторів як статичного SQL, так і динамічного SQL.

Що стосується операторів статичного SQL, то якого-небудь зміни після їх одноразового написання не передбачається. Вони можуть зберігатися як у файлах, призначених для подальшого використання, так і у вигляді процедур бази даних, що зберігаються, проте програмісти не отримують усієї тієї гнучкості, яку пропонує їм динамічний SQL. Незважаючи на наявність великого числа запитів, доступних кінцевому користувачеві, може статися так, що жоден з цих "законсервованих" запитів не зможе задовольнити його поточним потребам.

Динамічний SQL дає можливість програмісту або кінцевому користувачу створювати оператори під час виконання додатка і передавати їх базі даних, яка після виконання цих операторів поміщає вихідні дані в змінні програми. Динамічний SQL часто використовується інструментальними засобами, призначеними для побудови заздалегідь незапланованих запитів, що дозволяють оперативно формувати той або інший оператор SQL залежно від особливих вимог, що виникли в конкретній ситуації. Після налаштування оператора SQL відповідно до потреб користувача він прямує серверу баз даних для перевірки на наявність синтаксичних помилок і необхідних для його виконання привілеїв, після чого відбувається його компіляція і виконання.

Розглянемо застосування прикладного інтерфейсу програмування для виконання операторів SQL.

Прикладний API включає набір бібліотечних функцій, що надають програмістові різноманітні типи доступу до бази даних, а саме: підключення, виконання різних SQL-операторів, вибірка окремих рядків даних з результуючих наборів даних і т. д.

Щоб не розробляти окремі версії призначеного для користувача застосування для кожної з цільових СУБД, з якими це застосування планується використовувати, Microsoft розробила стандарт, що отримав назву Open Database Connectivity (ODBC). Технологія ODBC передбачає застосування єдиного інтерфейсу для доступу до різних баз даних SQL, причому мова SQL розглядається як основний стандартний засіб доступу.

Цей інтерфейс забезпечує високу міру універсальності, в результаті одно і те ж застосування може діставати доступ до даних, що зберігаються у базах різних цільових СУБД, без необхідності внесення змін до його програмного тексту. Таким чином, розробники отримали інструмент для створення і поширення додатків архітектури "клієнт-сервер", здатних працювати з широким спектром різних цільових СУБД, а зв'язати додатки з будь-якою вибраною цільовою СУБД можна за допомогою відповідного ODBC -драйвера.

Нині технологія ODBC фактично набула значення галузевого стандарту. Головною причиною її популярності є властива їй гнучкість, що надає розробникам наступні переваги:

- застосунки більше не пов'язані з прикладним API якоїсь однієї СУБД;
- SQL-оператори можуть явно включатися в початковий текст додатка або динамічно створюватися безпосередньо під час виконання програм;
- застосунок здатний ігнорувати особливості використовуваних протоколів передачі даних;
- дані посилаються і доставляються в тому форматі, який найбільш зручний для конкретного застосування;
- засоби підтримки ODBC розроблені з урахуванням вимог стандартів X/Open і CLI (Call Level Interface);
- нині існують драйвери ODBC для різних типів найпоширеніших СУБД.

Джерела даних комп'ютера зберігають відомості про підключення в реєстрі Windows на певному комп'ютері (https://support.microsoft.com/uk-ua/office/%D0%B0%D0%B4%D0%BC%D1%96%D0%BD%D1%96%D1%81%D1%82%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B4%D0%B6%D0%B5%D1%80%D0%B5%D0%BB-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-odbc-b19f856b-5b9b-48c9-8b93-07484bfab5a7). Джерела даних комп'ютера можна використовувати

лише на тому комп'ютері, на якому їх визначено. Є два типи джерел даних комп'ютера: користувацькі та системні. Користувацькі джерела даних може використовувати лише поточний користувач, і вони видимі лише для цього користувача. Системні джерела даних можуть використовувати всі користувачі на комп'ютері, і вони видимі для всіх користувачів на комп'ютері та в системних службах. Джерело даних комп'ютера особливо корисне, коли потрібно посилити безпеку, оскільки його можуть переглядати лише користувачі, які ввійшли в систему, і віддалений користувач не зможе скопіювати це джерело даних на інший комп'ютер.

Файлові джерела даних (так звані файли DSN) зберігають відомості про підключення в текстовому файлі, а не в реєстрі Windows, і зазвичай гнучкіші у використанні ніж джерела даних комп'ютера. Наприклад, ви можете скопіювати файлове джерело даних на будь-який комп'ютер із правильним драйвером ODBC, щоб програма посилалася на узгоджені та точні відомості про підключення на всіх комп'ютерах, які вона використовує. Або можна розмістити файлове джерело даних на одному сервері, надати до нього спільний доступ на багатьох комп'ютерах у мережі та легко зберігати відомості про підключення в одному розташуванні.

Крім того, для файлового джерела даних можна заборонити спільний доступ. Файлове джерело даних із забороненим спільним доступом розташовується на одному комп'ютері та вказує на джерело даних комп'ютера. Ви можете використовувати файлові джерела даних із забороненим спільним доступом, щоб отримати доступ до наявних джерел даних комп'ютера із файлових джерел даних.

### 9.1.2 Технологія ODBC та її особливості

У інтерфейс ODBC включені наступні елементи:

- бібліотека функцій, виклик яких дозволяє застосунку підключатися до бази даних, виконувати SQL-оператори і витягати інформацію з результуючих наборів даних;

- стандартний метод підключення і реєстрації в СУБД;
- стандартне представлення для цих різних типів;
- стандартний набір кодів помилок;
- типовий синтаксис SQL-операторів, побудований на використанні специфікації X/Open і ISO CGI.

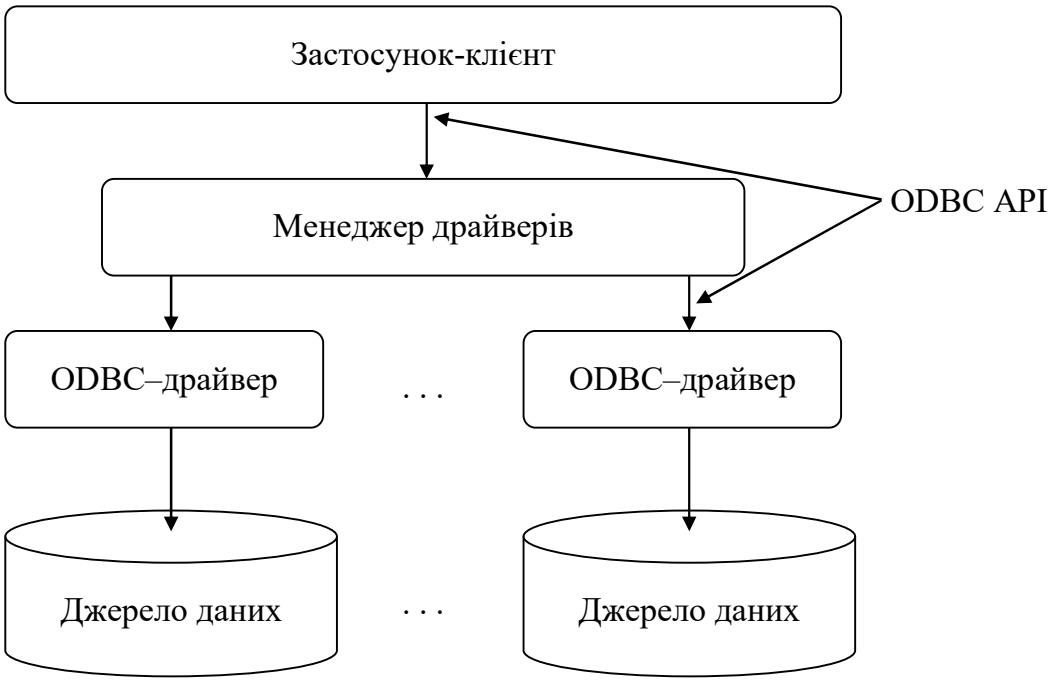



Рисунок 9.1

Загальна архітектура ODBC, яку наведено на рисунку 9.1, включає чотири елементи:

1. Застосунок. Цей компонент виконує обробку даних і виклик функцій бібліотеки ODBC для відправки SQL-операторів СУБД і вибірки поверненої СУБД інформації.

2. Менеджер драйверів. Він виконує завантаження драйверів на вимогу додатка.

3. Драйвери і агенти баз даних. Ці компоненти обробляють виклики функцій ODBC і направляють SQL-запити до конкретних джерел даних, а також повертають отримані результати застосунку. При необхідності драйвери виконують модифікацію початкового запиту

застосунка з метою приведення його у відповідність синтаксичним вимогам цільової СУБД.

4. Джерела даних. Містять ті дані, доступ до яких потрібний користувачеві застосунка. Дані зберігаються у БД, контрольованій цільовою СУБД, операційною системою, а також мережевою операційною системою, якщо така використовується.

Виходячи з вищесказаного, можна відмітити, що технологія ODBC пропонує єдиний інтерфейс доступу до різноманітних баз даних SQL. Мова SQL використовується в ній як основний стандарт доступу до даних. Інтерфейс ODBC (вбудований в мову С та інші) забезпечує високу міру універсальності: одно застосування може звертатися до різних SQL-сумісних СУБД за допомогою загального коду. Це дозволяє розробникові створювати і поширювати застосунки "клієнт/сервер" без урахування особливостей конкретної СУБД, в результаті одно і те ж застосування дістає можливість доступу до баз цих різних СУБД, що підтримують мову SQL. Подібні функціональні можливості технології ODBC дозволяють розробляти застосунки для роботи із СУБД різного типу. Для зв'язку застосунків з різнотипними СУБД використовуються відповідні ODBC–драйвери.

Технологія ODBC передбачає створення додаткового рівня між застосунком і використовуваною СУБД. Служби ODBC забезпечують отримання від застосунка запитів на вибірку інформації і переклад їх на мову ядра бази даних, що адресується, для доступу до інформації, що зберігається в ній.

Основне призначення ODBC полягає в абстрагуванні застосунка від особливостей ядра серверної бази даних, з якою воно здійснює взаємодію, тому серверна база даних стає як би прозорою для будь-якого клієнтського застосування.

Взаємодія застосунка з даними робиться за допомогою менеджера (диспетчера) драйверів, він підключає необхідний драйвер відповідно до формату цих СУБД. Драйвер СУБД, використовуючи мережеві засоби, як

правило, комунікаційні модулі конкретної СУБД, передає SQL-оператори серверу СУБД. Результати виконання запитів на сервері пересилаються назад до застосунку. Один із можливих варіантів підключення до бази даних клієнтського застосунку із використанням технології ODBC, наведений на рисунку 9.2 (джерело: https://www.sqlshack.com/how-to-configure-a-linked-server-using-the-odbc-driver/ ).

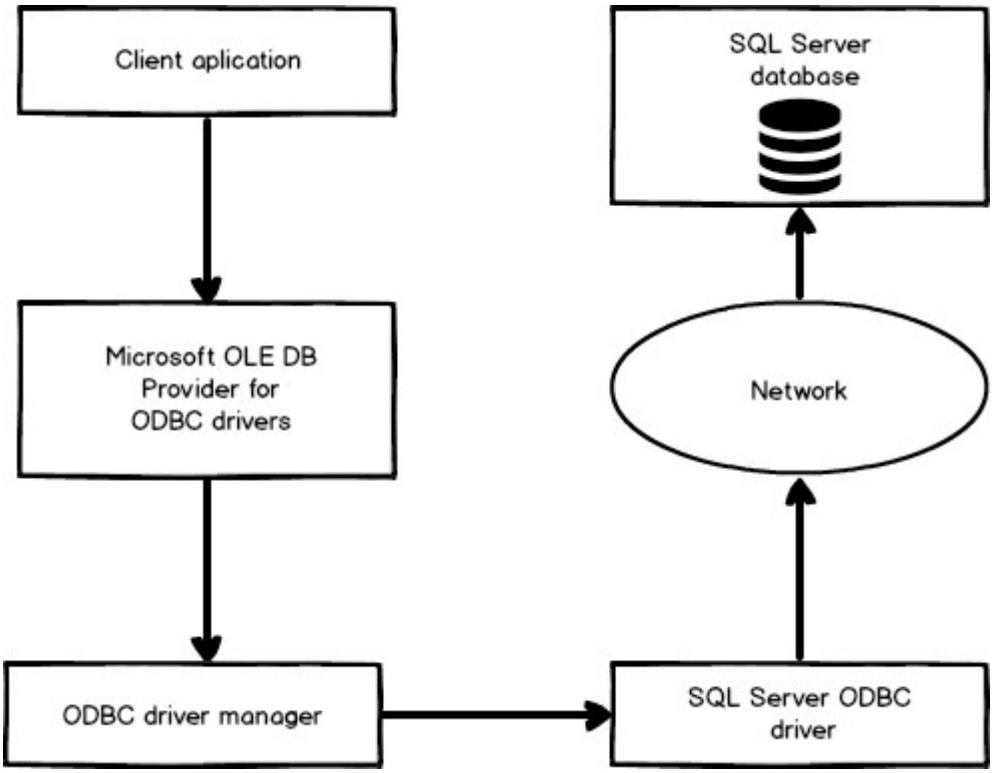



Рисунок 9.2

Перевага технології ODBC полягає в простоті розробки застосунків, що обумовлено високим рівнем абстрактності інтерфейсу доступу до даних практично будь-яких існуючих типів СУБД. При цьому можливе створення джерела даних, пов'язаного з будь-яким типом бази даних. За допомогою цієї технології можна створювати клієнт-серверні застосунки, причому засобами персональних СУБД доцільно розробляти клієнтську частину застосунка, а засобами SQL Server – серверну.

Основний недолік технології ODBC пов'язаний з необхідністю трансляції запитів, що знижує швидкість доступу до даних. У системах

клієнт-сервер він усувається шляхом переміщення обробки запиту з комп'ютера-клієнта на комп'ютер-сервер.

При використанні в клієнтському застосунку засобів ODBC здійснюється звернення до певного джерела даних, а через нього – до СУБД, що надає це джерело. Крім того, встановлюється загальна підсистема ODBC і визначаються пари "драйвер - база даних", яким задаються імена, вживані при установці з'єднання з базою даних. Відповідні пари називаються іменами джерел даних, або пойменованими джерелами даних ( Data Source Names, DSN ).

Створення джерела даних виконується за допомогою утиліти ODBC Data Source Administrator, що викликається з вікна панелі управління. До складу параметрів джерела даних входять: його ім'я і опис; сервер, з яким встановлюється з'єднання ; метод аутентифікації; ім'я бази даних.

Ім'я DSN дозволяє звернутися до джерела даних ODBC з прикладного застосунку.

Використання ODBC значно спрощується за допомогою Microsoft Foundation Classes Library (Бібліотека основних класів Microsoft). Прості застосування, що отримують доступ к таблицям через ODBC, можуть бути створені всього лише декількома натисненнями кнопки миші з використанням майстрів AppWizard та ClassWizard. Існує декілька класів MFS, які підтримують доступ до баз даних і наборам записів.

Докладні відомості про інтерфейс ODBC можна також подивитися, наприклад, в розділі MSDN «Довідкові матеріали для програміста ODBC» за посиланням:

https://learn.microsoft.com/en-us/sql/odbc/reference/odbc-programmer-s-reference?view=sql-server-ver16

## 9.2 Практичне опанування теми 9

### 9.2.1 Передумови виконання завдань теми 9

При виконанні практичних завдань теми 9 передбачається використання утиліти ODBC Data Source Administrator та СУБД MS Access. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, які зазначено далі (особливо у випадку використання версії програмного забезпечення, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Для початку виконання роботи треба підключити базу даних, яку було створено при виконанні лабораторної роботи за темою «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

### 9.2.2 Створення джерела даних ODBC

Для доступу до джерел даних ODBC необхідно виконати таку послідовність дій.

1. Відкрити Панель управління (Control Panel) Windows.
2. Відкрити папку Адміністрування (Administrative Tools).
3. Вибрати пункт Джерела даних (ODBC) (Data Sources (ODBC)).
4. Відкрити список ODBC-джерел подвійним клацанням миші.

Для створення нового джерела даних ODBC необхідно виконати таку послідовність дій.

1. Вибрати вкладку Користувацький DSN (User DSN) і натиснути кнопку Додати (Add).
2. Вибрати драйвер SQL Server і натиснути кнопку Готово (Finish).
3. Ввести ім'я джерела ODBC. Нехай це буде delivery і вибрати сервер, до якого потрібно підключитися. При виборі сервера потрібно вибрати сервер, який відповідає комп'ютеру, на якому виконується робота.

Це ім'я також можна побачити при підключенні до Microsoft SQL Server із використанням застосунку SQL Server Management Studio.

4. Натиснути кнопку Далі (Next).

5. У вікні автентифікації користувача залишити дані без змін і натиснути кнопку Далі (Next).

6. У наступному вікні вибрати зі списку баз даних базу даних delivery для використання за замовчуванням.

7. У наступному вікні можна усе залишити без змін і натиснути кнопку Готово (Finish).

8. Перевірити підключення до сервера та у разі успішного підключення натиснути кнопку ОК. У цьому випадку джерело ODBC буде збережено і з'явиться в списку джерел ODBC.

### 9.2.3 Використання СУБД Microsoft Access як клієнтського застосунку

У даному випадку СУБД MS Access буде використано не як СУБД, а як програмний засіб, за допомогою якого буде створено дуже простий клієнтський застосунок для роботи із базою даних Microsoft SQL Server Для цього необхідно виконати таку послідовність дій.

1. Запустити СУБД Microsoft Access.

2. Створити нову базу даних з ім'ям client_mssql.

3. У підменю пункту File вибрати пункт Зовнішні дані (Get External Data) та підпункт Зв'язок із таблицями (Link Tables…) (рисунок 9.3).

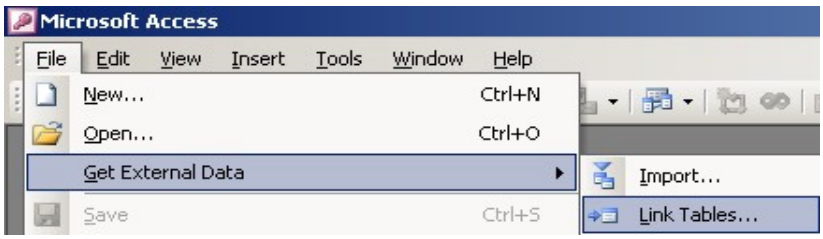



Рисунок 9.3

4. У вікні Зв'язок (Link), у полі зі списком Тип файлів (Files of type) вибрати пункт Бази даних ODBC (ODBC Databases).

5. У вікні Вибір джерела даних (Select Data Source) відкрити вкладку Джерело даних комп'ютера (Machine Data Source), вибрати джерело ODBC з ім'ям delivery.

6. У вікні Зв'язок із таблицями (Link Tables) вибрати таблицю dbo.Suppliers. В результаті буде створено таблицю dbo_Suppliers.

7. Аналогічно створити таблицю dbo_ Contracts, яка буде пов'язана з таблицею Contracts.

8. Перевірити можливість роботи з базою даних, використовуючи як інтерфейс користувача СУБД Microsoft Access. Для цього відкрити створені таблиці dbo_Suppliers (рисунок 9.4) та dbo_Contracts (рисунок 9.5). Використовуючи інтерфейс Microsoft Access, виконати перевірки операцій маніпулювання даними (додавання, видалення, зміна даних). Перевірити збереження результатів маніпулювання за допомогою SQL Server Management Studio.

dbo_Suppliers

| SupplierID | SupplierName | Note |
|---|---|---|
| 1 | ПП Іваненко І.І | м. Харків, вул. Пушкінська, 77 (тел.33-33-44, 12-34-56, факс 22-12-33) |
| 2 | ТОВ «Інтерфрут» | м. Київ, пр. Перемоги, 154, к. 3 |
| 3 | ПП Петренко П.П | м. Харків, пр. Науки, 55, к. 108, тел.32-18-44 |
| 4 | ЗАТ «Транссервіс» | м. Одеса, вул. Дерибасівська, 75 |
| 5 | ПП Сидорчук М.С. | м. Полтава, вул. Свободи, 15, кв. 43 |
| 6 | ПП Шевченко Г.С. | м.Суми, вул. Шевченко, 56 |
| * | | |

Рисунок 9.4

dbo_Contracts

| ContractNumber | ContractDate | ContractName | Comment | SupplierID |
|---|---|---|---|---|
| 1 | 01.09.1999 | Договір № 1 | Підстава - накладна №34 від 30/08/99 | 1 |
| 2 | 10.09.1999 | Договір № 2 | Підстава - рахунок-фактура № 08-78 від 28/08/99 | 1 |
| 3 | 10.09.1999 | Договір № 3 | Підстава - рахунок-фактура № 08-78 від 28/08/99 | 3 |
| 4 | 23.09.1999 | Договір № 4 | Підстава - замовлення № 56 від 28/08/99 | 3 |
| 5 | 24.09.1999 | Договір № 5 | Підстава - накладна № 74 від 11/09/99 | 2 |
| 6 | 01.10.1999 | Договір № 6 | Підстава - рахунок-фактура № 09-12 від 28/09/99 | 1 |
| 7 | 02.10.1999 | Договір № 7 | Підстава - накладна № 85 від 21/09/99 | 2 |
| 12 | 12.12.2002 | | | 6 |
| * | (№) | | | |

Рисунок 9.5

9. Розробити засобами Microsoft Access екранну форму, яка дозволить для кожного постачальника бачити список укладених із ним договорів. Можливий варіант реалізації такої форми наведено на рисунку 9.6. Як видно з рисунка, форма складається з головної та підлеглої форм, дані у яких зв'язані між собою.

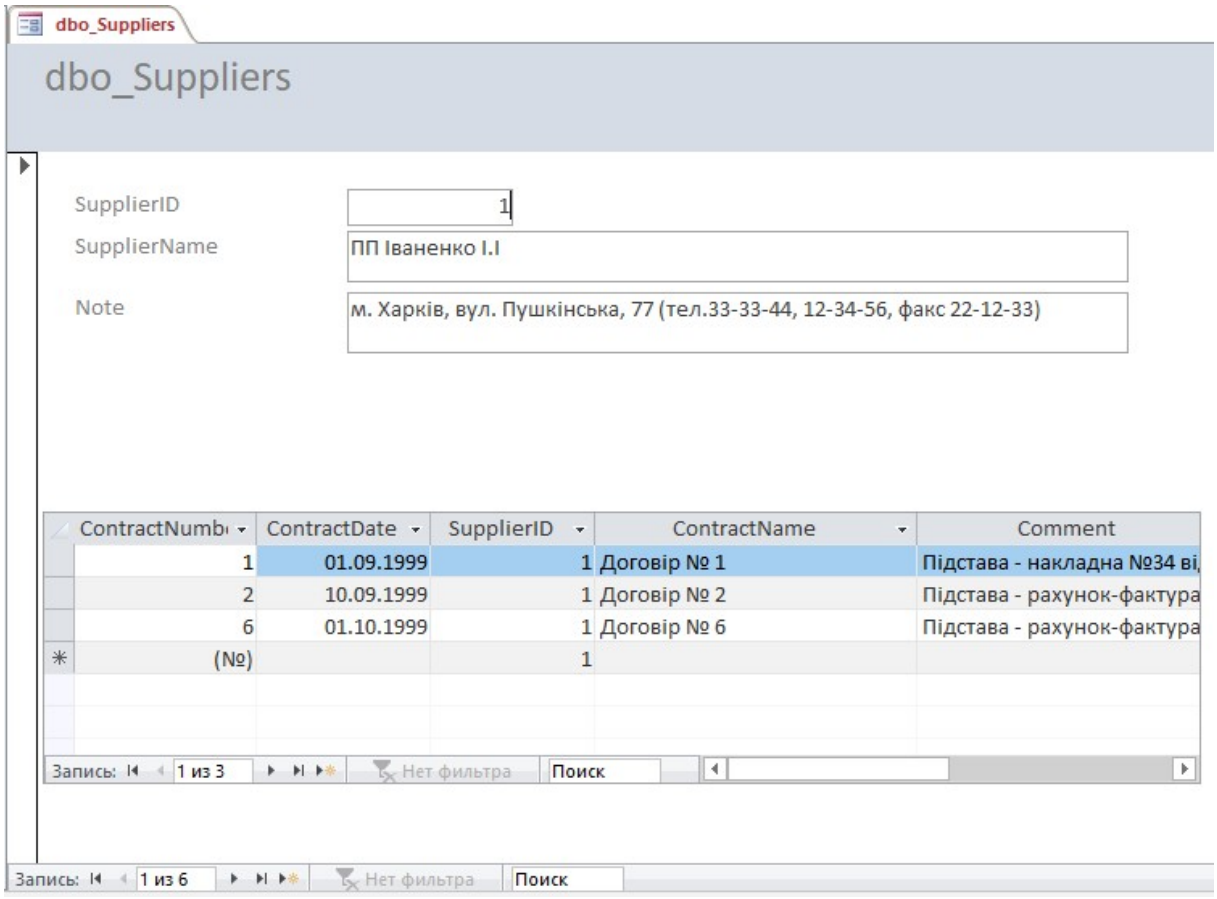



Рисунок 9.6

10. Після створення форми перевірити можливість роботи з базою даних, використовуючи як інтерфейс користувача створену форму.

11. Створити запит, за допомогою якого можна переглядати список договорів та основні відомості про постачальника, зокрема, назву. Текст такого запиту наведений на рисунку 9.7. Перевірити правильність роботи цього запиту та зберегти його із довільною назвою, наприклад, Запит1.

```
SELECT dbo_Contracts.ContractNumber, dbo_Contracts.ContractDate, dbo_Contracts.ContractName, dbo_Suppliers.SupplierName
FROM dbo_Contracts INNER JOIN dbo_Suppliers ON dbo_Contracts.SupplierID = dbo_Suppliers.SupplierID;
```

Рисунок 9.7

12. Припустимо, що список договорів, який формується за допомогою зазначеного вище запиту, треба мати можливість роздрукувати у певному вигляді. Для цього треба створити відповідний звіт та призначити для цього звіту джерелом даних Запит1. У режимі попереднього перегляду такий звіт може мати вигляд, наведений на рисунку 9.8. Перевірити правильність роботи цього звіту та зберегти його із довільною назвою, наприклад, Report1.

Report 1

| ContractNumber | ContractDate | ContractName | SupplierName |
|---|---|---|---|
| 1 | 01.09.1999 | Договір № 1 | ПП Іваненко І.І |
| 2 | 10.09.1999 | Договір № 2 | ПП Іваненко І.І |
| 3 | 10.09.1999 | Договір № 3 | ПП Петренко П.П |
| 4 | 23.09.1999 | Договір № 4 | ПП Петренко П.П |
| 5 | 24.09.1999 | Договір № 5 | ТОВ «Інтерфрут» |
| 6 | 01.10.1999 | Договір № 6 | ПП Іваненко І.І |
| 7 | 02.10.1999 | Договір № 7 | ТОВ «Інтерфрут» |

Рисунок 9.8

13. Підключити інші таблиці базі даних Microsoft SQL Server.

14. Використовуючи раніше отримані навички роботи з СУБД Microsoft Access створити інші форми, що дозволяють працювати з базою даних, а також засоби обробки даних (запити, звіти), що дозволяють обробляти інформацію, що зберігається в базі даних Microsoft SQL Server засобами Microsoft Access, виводити її на друк тощо.

Наведений вище приклад створення джерела даних ODBC та використання його у клієнтському застосунку є досить простим. Більш детальну інформацію щодо побудови та використання джерел даних ODBC при роботі з базами даних Microsoft SQL Server, можна, наприклад, отримати за посиланнями:

https://learn.microsoft.com/en-us/sql/odbc/admin/odbc-data-source-administrator?view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/odbc/admin/managing-data-sources?view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/odbc/odbc-glossary?view=sql-server-ver16

https://docs.devart.com/odbc/sqlserver/access.htm

Для збереження результатів виконання практичних завдань треба зберегти файл client_mssql.accdb або файл client_mssql.mdb залежно від того, яку версію СУБД Microsoft Access було використано.

### 9.2.4 Звітність про виконання практичних завдань теми 9

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;
2) описати створений клієнтський застосунок та особливості роботи з ним;
3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка – 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 9.3 Питання для самоперевірки по темі 9

1. Якими способами реалізоване на практиці застосування мови SQL в прикладних програмах?
2. Що таке впроваджені SQL-оператори?
3. Що таке API?
4. Що таке ODBC?
5. Які переваги надає використання технології ODBC?
6. Які недоліки є характерними для технології ODBC?
7. Які складові виділяють у інтерфейсі ODBC?
8. Які складові включено до загальної архітектури ODBC?

9. Загальна архітектура ODBC. Застосунок як архітектурна складова, його призначення.

10. Загальна архітектура ODBC. Менеджер драйверів як архітектурна складова, його призначення.

11. Загальна архітектура ODBC. Драйвери і агенти баз даних як архітектурні складові, їх призначення.

12. Загальна архітектура ODBC. Джерела даних як архітектурні складові, їх призначення.

13. Що таке DSN?

14. Які види DSN реалізовано у Windows?

15. Яким чином запустити утиліту ODBC Data Source Administrator?

16. Яким чином виконується створення джерела даних за допомогою утиліти ODBC Data Source Administrator?

17. Яким чином можна забезпечити автентифікацію користувача при створенні джерела даних ODBC?

18. Як забезпечити підключення із середовища СУБД Microsoft Access до бази даних Microsoft SQL Server?

19. Які особливості роботи із зовнішніми даними у середовищі СУБД Microsoft Access при використанні засобу зв'язку Link Tables?

20. Який засіб зв'язку із зовнішніми даними у середовищі СУБД Microsoft Access є альтернативним засобу зв'язку Link Tables?

21. Яким чином можна реалізувати роботу із зовнішніми даними у середовищі СУБД Microsoft Access для кінцевого користувача?

22. Які недоліки є характерними при використанні СУБД Microsoft Access як засобу розробки клієнтського застосунку?

# ТЕМА 10
# РОЗРОБКА ЗАСОБАМИ MICROSOFT VISUAL STUDIO ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОБОТИ З БАЗОЮ ДАНИХ MICROSOFT SQL SERVER

## 10.1 Теоретичні відомості

Користувач автоматизованої обчислювальної системи має право очікувати не тільки точних результатів обробки, але і зручності в використанні системи. Тіло людини (користувача) - це механізм, який працює в рамках визначених обмежень і допусків. Очам людини необхідно, щоб образи мали визначений розмір, рівень яскравості, контрастності і розміщувались на зручній відстані. Деякі кольори сприймаються краще інших. Необхідно відмітити і обмеження мозку людини. У людини велика довготривала пам'ять і дуже обмежена короткочасна пам'ять. Подібно буферам обчислювальної системи ця пам'ять може легко перевантажуватись. Люди можуть розширювати цю пам'ять за допомогою записів на листках паперу, застосовуючи нові моделі роботи або пристосовуючись до нового способу роботи. Однак така адаптація може приводити до стресів, а як результат до неприйнятного рівня помилок.

Критерій зручності став визначальним в останні роки в зв'язку з широким застосування персональних комп'ютерів з діалоговими режимами роботи в різних галузях народного господарства, побуті, що привело до зростання кількості користувачів неспеціалістів в області інформатики та обчислювальної техніки. Згідно з класифікацією фірми IBM розрізняють такі типи користувачів:

– системний програміст;
– програміст;
– кінцевий користувач.

Кінцеві користувачі – це люди, які хочуть більш ефективно і економно вирішувати свої завдання, використовуючи комп'ютерні

засоби. Частіше всього вони не є спеціалістами в області інформатики. Досвід показує, що ці користувачі можуть ефективно вирішувати на машині свої завдання лише в умовах наявності активної допомоги зі сторони системи на всіх етапах вирішення завдання і спрощення методів взаємодії з обчислювальною системою.

Зрозуміло, що сучасні обчислювальні системи повинні орієнтуватись на таких користувачів в першу чергу, оскільки більшість користувачів персональних комп'ютерів є кінцевими користувачами. Використовуючи в подальшому термін користувач, ми маємо на увазі кінцевих користувачів.

Таким чином, з точки зору користувача діалогової системи - система є зручною для користувача, якщо інтелектуальні зусилля користувача, необхідні для розуміння дій системи і реакції на них мінімальні. Можливо сформулювати ряд вимог до системи з точки зору зручності:

1. Поведінка системи по відношенню до користувача повинна бути гнучкою, тобто користувач не повинен діяти строго визначеним способом.

2. Система повинна вміти розрізняти користувача і пристосовуватись до нього.

3. Поведінка системи повинна бути зрозумілою користувачеві.

4. Система завжди повинна бути готова допомогти користувачеві.

5. Для використання системи не потрібні спеціальні навики та додаткове навчання.

6. Не потрібно зловживати здатністю людини до навчання під час роботи з системою.

7. Система повинна реагувати на порушення взаємодії з користувачем, обумовлені властивостями людини і приймати запобіжні заходи проти цих порушень.

Інтерфейс користувач-комп'ютер включає всі ті аспекти обчислювальної системи, з якими безпосередньо взаємодіє користувач.

Ефективна робота кінцевих користувачів з базами даних можлива лише за наявності інтерфейсу користувача як обов'язкової складової застосунку, який забезпечує роботу з базою даних.

### 10.1.1 Визначення та суть інтерфейсу користувача

Інтерфейс користувача (ІК) - це сукупність засобів, за допомогою яких користувач взаємодіє з різними пристроями (з комп'ютером або побутовою технікою) або іншим складним інструментарієм (системою). Інтерфейс користувача – це такий різновид інтерфейсів, в якому з одного боку – людина, з іншого – машина (пристрій, програмне забезпечення).

ІК часто розуміють лише як зовнішній вигляд програмного забезпечення (ПЗ), але таке розуміння є надто вузьким, оскільки саме за допомогою інтерфейсу користувач сприймає програму в цілому та використовує її функціональність. ІК забезпечує підтримку прийняття рішень у визначеній предметній галузі та визначає порядок використання ПЗ і документації до нього. В дійсності, ІК об'єднує усі елементи і компоненти ПЗ, які здатні впливати на взаємодію користувача з програмним забезпеченням. До таких елементів належать: набір задач, які користувач розв'язує за допомогою ПЗ; використовувана програмним забезпеченням метафора (наприклад, «робочий стіл» у операційній системі Windows); елементи управління ПЗ; навігація між блоками ПЗ; візуальний (і не тільки) дизайн вікон та екранних форм програми та інші складові (рисунок 10.1).

Стиль інтерфейсу користувача – це набір ознак, методів, прийомів діяльності, які характеризують індивідуальність інтерфейсу користувача, а також сукупність прийомів використання інструментів розроблення ПЗ.

Процес проектування ІК – це складний, нелінійний, недетермінований і неортогональний процес. Складність ІК обумовлюється рядом невизначеностей, які суттєво впливають на процес розроблення. Нелінійність проектування ІК полягає у

відсутності фіксованого, впорядкованого і прямолінійного алгоритму від початку до кінця проектування. Процес проектування є невизначеним, оскільки не існує рівняння, за яким можна було б одержати однаковий результат при заданих однакових початкових умовах, більш того, одержати ідентичний результат практично неможливо. Інтерфейс користувача неортогональний у тому сенсі, що будь-який аспект проектного рішення може впливати на інші аспекти, до того ж результат цього впливу не завжди є позитивним та прийнятним.


Рисунок 10.1 – Складові інтерфейсу користувача

### 10.1.2 Графічний інтерфейс користувача

Графічний інтерфейс користувача (GUI – Graphical User Interface) - тип інтерфейсу користувача, в якому елементи інтерфейсу (меню, кнопки, значки, списки), представлені на екрані, виконані у вигляді графічних зображень. В графічному інтерфейсі користувача (ГІК) забезпечено довільний доступ (за допомогою пристроїв введення) до всіх видимих екранних об'єктів (елементів інтерфейсу) та безпосереднє маніпулювання ними. Найчастіше елементи інтерфейсу в ГІК реалізовані на основі метафор та відображають свої призначення та властивості, що полегшує розуміння та засвоєння ПЗ недосвідченими користувачами.

Основною характеристикою графічного інтерфейсу користувача є використання ряду робочих елементів. ГІК є графічним представленням на екрані комп'ютера способу чи функції для інтерактивної взаємодії з програмами, об'єктами і даними. Він забезпечує користувачів необхідним інструментарієм і застосунками, а не просто списком функцій.

Характеристики ГІК:

1) підтримує ідею сумісності між програмами;

2) користувачі можуть бачити графічні зображення і текст на екрані в тому вигляді, в якому вони будуть роздруковані;

3) слідує концепції інтерактивної взаємодії «об'єкт-дія»;

4) дозволяє переміщувати інформацію між програмами;

5) надає можливість безпосереднього маніпулювання об'єктами та інформацією на екрані;

6) пропонує стандартні елементи інтерфейсу (меню та діалогові вікна);

7) забезпечує візуальне відображення інформації і об'єктів (іконки і вікна);

8) забезпечує візуальний зворотній зв'язок в процесі виконання користувачами дій та задач;

9) надає візуальне відображення дій користувача/системи, а також режимів (меню, палітри);

10) використовує графічні керуючі елементи, які дозволяють користувачам робити вибір та вводити дані;

11) надає користувачам можливість налагодити та персоналізувати інтерфейс та інтерактивні дії;

12) забезпечує гнучкість переходу від клавіатури до іншого пристрою введення даних.

Під час досліджень зручності застосування ГІК було окреслено коло задач, які найбільш часто розв'язуються користувачами різних рівнів підготовки: запуск програми, виклик підказки, відкриття файлу, збереження файлу, копіювання файлу, зміна кольору робочого столу, пошук файлу, запуск ще одного додатку, видалення файлу/папки, перейменування файлу/папки, вибір принтера, створення папки, перегляд черги завдань принтера, розміщення документу на сервері, закриття програми, відхилення видалення/відновлення файлу, перевірка стану ресурсів системи.

Одним з недоліків ГІК є його орієнтація на застосунки. При роботі з ГІК користувачі бачать інформацію на екрані, в них складається враження, що вони дійсно працюють з об'єктами, однак увага користувачів сконцентрована на додатках. Вони обирають застосунок, вказують файл даних, який використовується, організовують додатки і дані на комп'ютерах у вигляді графічних деревовидних структур, використовуючи диски і папки.

ГІК передбачає проблемно-орієнтований підхід – перш ніж виконувати операції з файлами, користувачі повинні запустити програмне забезпечення, яке працює з даним типом файлів. Користувачі працюють з застосунками у вікнах, які мають панель меню з варіантами маршрутів, відображуваними спадними меню. Такі меню містять варіанти дій і маршрутів, пов'язаних з об'єктами у вікні або додатком, який надає саме вікно.

Будь-який вдало розроблений ІК повинен знижувати навантаження на пам'ять користувача. ГІК здатні запропонувати наочні підказки та необхідні відомості, використовуючи комп'ютерні потужності для зберігання та пошуку інформації. ГІК об'єднують три основних стилі інтерфейсу: командний рядок, меню та пряме маніпулювання.

Методи інтерактивної взаємодії ГІК передбачають використання клавіатури або вказівних пристроїв для переміщення, вибору та маніпулювання інформацією на екрані.

До методів інтерактивної взаємодії ГІК належать:

1) можливість використання клавіатури або миші в залежності від переваг користувача;
2) робота з різними типами меню;
3) два режими редагування: вставка або заміна;
4) технологія drag-and-drop;
5) буфер обміну.

### 10.1.3 Принципи побудови «дружнього» інтерфейсу користувача

Поняття «дружнього» інтерфейсу користувача (User-friendly Interface) розшифровується як «інтерактивний програмний інтерфейс, який забезпечує природній для користувача режим взаємодії з обчислювальною машиною». Відтоді, як ІК став одним з найважливіших елементів ПЗ, його якість та «дружність» хвилює кожного розробника і проектувальника.

Думка користувача про те, що деякий ІК якісний, тобто кращий за інші, залежить від невеликої кількості характеристик, які стають очевидними після знайомства з будь-яким додатком незалежно від стилю ІК. Якісний ІК повинен базуватись на 5 головних принципах, які позначають абревіатурою SAPCO (рисунок 10.2).

**Простий**. Програмні об'єкти забезпечують підвищення продуктивності і нарощення можливостей ПЗ без внесення додаткової

складності в ІК. На початкових етапах розробки стилю ІК широко використовується мінімалізм і розбиття ПЗ на множину слабозв'язаних рівнів. Якісний прикладний ІК не потребує довідника або діалогової довідкової системи, щоб почати виконувати задачі за його допомогою. Якісний прикладний ІК зрозумілий на інтуїтивному рівні (користувачу потрібне лише пояснення, як досягти результату).

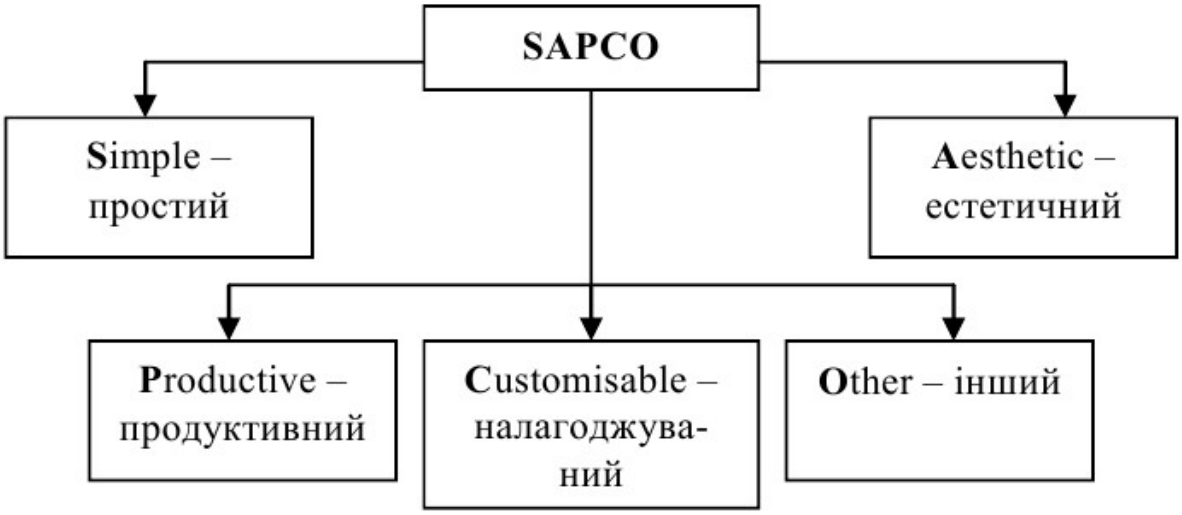



Рисунок 10.2 - Принципи SAPCO проектування якісного ІК

**Естетичний**. Програмні об'єкти повинні мати естетичну та ергономічну привабливість, при цьому широко використовується графічний дизайн та візуалізація. Якісний прикладний ІК повинен не викликати негативних емоцій у користувача, використовувати метафори реального світу користувача та максимально візуалізувати інформацію.

**Продуктивний**. Використання програмних продуктів вимагає мінімальних зусиль для вирішення задач користувача. ІК реалізується таким чином, щоб уникнути складної ієрархічності вікон та/або екранів, а також зайвих дій з клавіатурою та мишею.

**Налагоджуваний**. Програмне забезпечення доступне в різних формах для задоволення індивідуальних потреб. Програмні об'єкти ІК повинні мати можливість налагодження (налагоджуваність). Якісний

прикладний ІК дозволяє користувачу обрати метод взаємодії і методи макетування та доступу для оптимізації потреб користувачів.

### 10.1.4 Правила проектування інтерфейсу користувача

Розглянемо основні правила проектування інтерфейсу користувача.

***Правило 1: «Дати контроль користувачу»***. Основне правило проектування ІК – дати користувачу контроль над системою. Проста аналогія – водій автомобіля та пасажир потягу. Рішення – їхати власним авто або потягом – повинен приймати саме пасажир (в нашому випадку користувач). Розробник інтерфейсу повинен виділити в якості основного такий принцип проектування, при якому продукт буде задовільняти всім вимогам. Досвідчені проектувальники дозволяють користувачам вирішувати деякі задачі на власний розсуд. Так, наприклад, європейські архітектори по завершенні будівництва складного комплексу будівель повинні прокласти між ними доріжки для пішоходів. На майданчиках між будинками ставлять таблички: “Ходіть, будь-ласка, по траві”. Через деякий час будівельники повертаються і лише тоді, згідно волевиявленню населення, асфальтують протоптані доріжки. Це прекрасний приклад надання можливості користувачу контролю над ситуацією і проектування інтерфейсу, який би відповідав потребам та побажанням користувача.

Основними положеннями надання контролю користувачу над інтерфейсом є безрежимність, гнучкість, можливість переривання, корисність, поблажливість, можливість орієнтування, доступність, спрощення користування, пристосовуваність, інтерактивність.

**Безрежимність** – вибір та використання режимів розважливо. Режими (певні умови роботи, діяльності) - атрибут багатьох програмних інтерфейсів, але застосовувати їх потрібно лише при необхідності. Існує ряд інтерфейсів, які занадто часто перемикаються між режимами. На екрані з'являється діалогове вікно режиму, і користувач

стає обмеженим в діях не лише в даній програмі, але й не має можливості переходу в інші програми.

Інтерфейс може працювати в двох режимах, які в багатьох випадках необхідні, однак позбавляють користувача самостійності:

1) режим користувача – обмежує режим роботи і дозволяє користувачу виконувати лише певні дії в певному місці програми. В наш час проектувальники інтерфейсів все частіше обходяться без перемикання режимів. Можливо, оптимальним варіантом є показ даних в форматі, який відповідає рівню доступу користувача;

2) системний режим – використовується достатньо рідко. Якщо система знаходиться в цьому режимі, йому дозволено працювати лише з поточною програмою. При розробленні повідомлень та інформації про допомогу в такому режимі слід пам'ятати про те, наскільки незручний системний режим для користувача.

При виборі режимів важливо слідувати принципу негайного візуального оберненого зв'язку. Користувач повинен бути постійно впевнений в тому, що він знаходиться в потрібному режимі. Режими інтерфейсу повинні бути настільки природніми, щоб користувачу було комфортно працювати з ними.

**Гнучкість** – забезпечує користувачу можливість вибору, наприклад, можливість роботи з клавіатурою передбачає використання клавіатури замість миші. Панелі інструментів створені для прискорення роботи при використанні миші, однак при роботі з клавіатурою до них важко дістатись – для подібних випадків передбачені підменю. Більшість користувачів мають звичку працювати як з клавіатурою, так і з мишею.

**Можливість переривання** - дозвіл користувачу перемикати увагу між різними програмними засобами. Програмні інтерфейси повинні бути спроектовані так, щоб користувач міг в будь-яку хвилину перервати або зберегти результати роботи. Забезпечення контролю користувача над програмою та його підтримка – ось головні принципи розроблення. Не потрібно змушувати користувачів закінчувати

виконання початих послідовностей дій. Вони повинні мати вибір – анулювати або зберегти дані і повернутись туди, де вони перервались.

**Корисність** – демонстрація повідомлень, які допомагатимуть користувачу в роботі. У інтерфейсі потрібно використовувати зрозумілі для користувача терміни. Даний принцип застосовний не лише для повідомлень, але й для усіх текстів на екрані: запрошень, інструкцій, заголовків, написів на кнопках. Усі текстові аспекти інтерфейсу повинні розроблятися спеціалістами, котрі мають знання з орфографії та лексики. При написанні системної та програмної документації, а також повідомлень слід обрати вірний тон. Невдалі термінологія та тон призводять до того, що користувачі звинувачуватимуть себе у помилках, що виникають при використанні ПЗ.

**Поблажливість** – створення умов для негайних та зворотніх дій, а також зворотнього зв'язку. Недостатність зворотнього зв'язку в більшості програмних продуктів змушує користувача витрачати багато сил на виконання поставленої задачі. Кожен програмний продукт повинен мати функції відхилення (скасування) та повторення дії. Необхідно інформувати користувача, якщо дана дія не може бути відхилена, і, по можливості, дозволити йому альтернативну дію.

**Можливість орієнтування** – забезпечення доступу користувачу до будь-якої частини інтерфейсу. Користувач повинен мати можливість вільно орієнтуватись в інтерфейсі, мати доступ до будь-якої його частини, можливість пересуватись вперед і назад по нисхідній та висхідній структурі інтерфейсу, отримувати зручні контекстні підказки там, де вони потрібні. Наприклад, панель задач Windows показує, які програми виконуються, і надає користувачу доступ до них.

Панелі інструментів, меню, палітри та інші елементи інтерфейсу операційних систем, набори програмних продуктів та різні утиліти - все це розроблено, щоб допомогти користувачам орієнтуватись в операційній системі та в просторі жорсткого диску. Системні та

програмні “помічники” та “асистенти” теж пропонують допомогу для орієнтації в програмних функціях або задачах.

**Доступність** – налагодження системи на користувачів з різним рівнем підготовки. Деякі програми пропонують спеціальні інтерфейси, які допомагають обрати рівень складності взаємодії. Наприклад, панель задач та підменю програм можуть бути стандартними або деталізованими в залежності від вибору користувача та типу виконуваної задачі. Для досвідчених користувачів потрібно передбачити швидкий доступ до функцій ПЗ.

**Спрощення користування** – створення зрозумілого, «дружнього» ІК. ІК – «міфічна» частина програмного продукту. При вдалому проектуванні користувачі не помічають інтерфейсу. Якщо інтерфейс розроблений невдало, користувачам доведеться докласти чимало зусиль для ефективного використання програмного продукту. «Прозорість» інтерфейсу забезпечується тим, що користувачу надається можливість користуватись об'єктами, відмінними від системних команд.

**Пристосовуваність** – надання можливості користувачу налагоджувати інтерфейс за своїм смаком. Користувач повинен мати можливість налагодження: способу представлення інформації на свій вибір (кольори, шрифти, розташування елементів), поведінки інтерфейсу (дії за замовчуванням, макроси, кнопки) та інтерфейсних функцій (натискання кнопок чи клавіш, сполучення клавіш для швидкого вибору команд, мнемоніка, розташування кнопок миші для виконання команд).

**Інтерактивність** – дозвіл користувачу маніпулювати об'єктами інтерфейсу. Всюди, де це можливо, потрібно дозволяти користувачу безпосередньо взаємодіяти з об'єктами на екрані в природньому, натуральному стилі. Користувачі повинні комфортно почуватись при виконанні операцій та знати про передбачуваний результат.

***Правило 2: «Зменшити навантаження на пам'ять користувача».***

Положеннями зменшення навантаження на пам'ять користувача є запам'ятовування, розпізнавання, інформування, терпимість, швидкість, інтуїтивність, перенос, контекст, організація.

**Запам'ятовування** – не слід завантажувати короткочасну пам'ять. Короткочасна пам'ять допомагає нам зберігати інформацію на протязі невеликого проміжку часу. Користувачі часто працюють над декількома задачами одночасно, тому не варто ІК завантажувати короткочасну пам'ять користувача в моменти перемикань між ними. Цей принцип проектування часто порушується, що змушує застосовувати зовнішні «засоби» для зберігання інформації (папір, калькулятор).

Функції програм (виділення останньої дії та її повтор) та дії з використанням буферу обміну дозволяють користувачам маніпулювати частинами інформації, необхідними в багатьох місцях, а також всередині задач. Оптимальним є варіант, коли програма в потрібному місці може автоматично зберегти та передати дані, коли користувач зайнятий виконанням інших задач.

**Розпізнавання** – потрібно опиратись на розпізнавання, а не на повторення. Підтримка інтерфейсом довгочасної пам'яті передбачає розпізнавання інформації користувачем, перш ніж він згадає її. Легше обрати якийсь об'єкт зі списку, ніж згадувати його правильну назву для введення в порожній рядок. Онлайнова допомога, повідомлення, поради щодо користування інструментами, система контекстної допомоги тощо, допомагають користувачу обирати інформацію, а не згадувати її. Слід передбачити списки і меню, що містять об'єкти чи документи, які можна обирати, не змушуючи користувачів вводити інформацію в командному рядку без підтримки системи.

**Інформування** – наявність візуальних заставок. Необхідний аспект будь-якого графічного ІК та об'єктно-орієнтованого ІК полягає в тому, що користувачі повинні знати, в якому місці ПЗ вони знаходяться, які

дії виконують наразі і які дії можуть виконати в подальшому. Візуальна інформація служить нагадуванням для користувачів. Коли користувачі працюють в якомусь режимі або з мишею, це повинно відображатись на екрані за допомогою відповідної індикації. Форма курсора може змінюватись для вказання поточного режиму або дії, а індикатор – вмикатись і вимикатись. Візуальна інформативність продукту полягає в наявності візуальних підказок інтерфейсу, які інформують користувача про його поточні дії.

**Терпимість** – треба передбачити параметри за замовчуванням, команди відхилення (скасування) та повторення дії. Інтерфейс повинен надавати користувачу можливість зміни установок та налаштувань ПЗ, а також можливість повернення до установок за замовчуванням. Користувачі за своїми перевагами можуть змінити колір, шрифт, властивості прикладної програми або операційної системи, не задумуючись про їх початковий варіант.

При розробленні інтерфейсу потрібно передбачити багаторівневі системи відхилення та повторення команд. Швидкість - слід передбачити «швидкі» шляхи проходження інтерфейсу. Крім комбінованого використання миші та клавіатури існує ряд інших можливостей прискорити роботу з програмою. «Гарячі» клавіші та ярлики зменшують навантаження на пам'ять користувача та доводять виконання операцій до автоматизму.

Є 2 основних способи встановлення ярликів: прискорюючі та мнемонічні. Мнемонічні (або доступні) ярлики – це одиночні буквенно-цифрові символи, які встановлюють курсор на потрібний об'єкт та дозволяють зробити вибір. Вони використовують різні меню (панель, підменю, контекстні) та списки. Мнемонічні символи повинні бути унікальними для кожного роду дій. Наприклад, типове меню вікна використовує стандартні клавіші: F – для файлу, E – для редагування, V – для перегляду, H – для виклику довідкової системи. Наступний рівень меню (підменю) використовують свої налагодження мнемонічних

клавіш: N – новий документ, O – відкрити, C – закрити, S – зберегти. Мнемонічні символи прискорюють рух і вибір потрібного меню або списку. Прискорюючі ярлики (або клавіші швидкого доступу) – це клавіші або комбінації клавіш, які користувачі повинні натиснути для здійснення якої-небудь дії. Наприклад, Ctrl+P – прискорюючий ярлик для друку.

Щойно користувачі добре засвоять програмний продукт, вони починають відчувати потребу в прискорюючих ярликах. Не слід ігнорувати цю необхідність, однак при проектуванні потрібно слідувати стандартам щодо призначення клавіш та їх комбінацій, інакше користувачі робитимуть помилки, використовуючи звичну комбінацію клавіш, яка виконуватиме непередбачувану дію.

**Інтуїтивність** – потрібно активізувати синтаксис дій з об'єктами. Об'єктно-орієнтований інтерфейс надає ряд переваг від використання об'єктно-орієнтованого синтаксису взаємодії. Користувачам не треба запам'ятовувати, яку дію можна виконати в певний момент часу для об'єкта. При виборі об'єкта користувач бачить у меню дії, які можуть бути виконані над об'єктом. Недопустимі дії виділяються, наприклад, сірим кольором. Об'єктно-орієнтований синтаксис дозволяє користувачу зрозуміти взаємозв'язок між об'єктами та діями в програмному продукті.

**Перенос** – в інтерфейсі слід використовувати метафори (переноси, які використовують назву об'єкту одного класу для опису об'єкту іншого класу) реального світу, які дозволяють користувачу переносити свої знання з реального світу у світ комп'ютерів. Наприклад, для програми Калькулятор не варто розробляти новий інтерфейс, він повинен співпадати з інтерфейсом звичайного калькулятора, який добре засвоїли користувачі. Однак при виборі та використанні метафор для інтерфейсу слід бути обережними. Обираючи метафору, потрібно зафіксувати її та слідувати їй весь час. Якщо виявиться, що метафора не відповідає своєму призначенню в усьому інтерфейсі, з'явиться необхідність вибору нової метафори.

**Контекст** – слід застосовувати розкриття та пояснення понять і дій. Користувачі не повинні вважати можливості ПЗ більшими чи іншими, ніж вони є насправді. На кожному рівні інтерфейсу не варто показувати абсолютно всі функції ПЗ, а лише ті, в яких є потреба саме на даному рівні. Деякі програмні продукти надають різні меню для користувачів. Спочатку можна обрати просте меню для щоденних, звичайних потреб. Пізніше, по мірі засвоєння продукту або при виникненні потреби в більш складних властивостях програми, користувачі зможуть обрати більш складне меню. Це приклад того, як користувач одержує контроль над задачею та програмою. Нові властивості інтерфейсів (Майстри та Порадники) розкривають та пояснюють поняття та дії при вирішенні задач. Майстри допомагають користувачу виконувати задачу крок за кроком, кожен з яких є зрозумілим. Розробник має забезпечити користувачу легкий доступ до часто використовуваних функцій та дій. Можна приховати частину властивостей та функцій і дозволити користувачу викликати їх по мірі необхідності. Не потрібно намагатись відобразити всю інформацію в головному вікні, для цього можна використати вторинні вікна.

**Організація** – потрібно збільшити візуальну ясність інтерфейсу. Принципи візуального проектування дають можливість полегшити сприйняття інформації шляхом використання групувань об'єктів в меню або списки; нумерації об'єктів; заголовків та запрошень. Деякі програми одночасно надають надто багато інформації на екрані, що викликає відчуття хаосу. Інформація має подаватись у порядку, зрозумілому користувачам, тобто “форма повинна відповідати призначенню”.

***Правило 3: «Зробити інтерфейс сумісним»***.

Сумісність – ключовий аспект інтерфейсу. Однією з основних переваг сумісності є те, що користувачі можуть перенести свої знання

та навички зі старої програми, якою вони користувались раніше, у нову.

Наприклад, при створенні інформаційного повідомлення бажано присвоїти йому ідентифікаційний номер, який надалі слід видавати разом з повідомленням. Це дозволить користувачу зрозуміти, що схожі повідомлення з'являються кожен раз в певній ситуації, незалежно від того, в якому місці інтерфейсу він знаходиться.

Положеннями розроблення сумісного інтерфейсу є: послідовність інтерфейсу, досвід, прогнозування, відношення, передбачуваність.

**Послідовність інтерфейсу** – проектування послідовного, узгодженого інтерфейсу. Користувачі повинні мати опорні точки інтерфейсу. Це заголовки вікон, навігаційні карти та деревовидні структури. Користувачу також потрібна можливість завершити поставлену задачу без зміни середовища роботи або перемикання між стилями введення інформації. Візуальна допомога підкаже користувачу, що відбудеться при завершенні задачі.

**Досвід** – спільна сумісність всіх програм. Один з головних аспектів в розробці інтерфейсу – можливість навчання користувача головним концепціям системи та програмного забезпечення, що можуть застосовуватись у нових ситуаціях та інших програмах.

**Сумісність** – одна з головних властивостей ІК, яка проявляється на трьох рівнях: подання інформації, поведінка програми та техніка взаємодії.

Сумісність в поданні інформації передбачає, що користувачі можуть сприймати інформацію та об'єкти в схожому логічному, візуальному та фізичному вигляді в усьому програмному продукті. Стиль представлення інформації не повинен змінюватись без видимих причин.

Сумісність в поведінці програми передбачає, що об'єкт є однаковим всюди. Поведінка контрольних інструментів, таких як кнопки, списки і меню, не змінюється всередині програми або у

різних програмах одного розробника. Користувачі мають бути впевнені в послідовній поведінці об'єктів інтерфейсу.

Сумісність техніки взаємодії також дуже важлива. Однакові сполучення “швидких” клавіш та способи роботи з мишею повинні працювати в схожих за призначенням програмах. Мнемонічні схеми клавіатури теж не повинні змінюватися. Користувачі очікують аналогічних результатів при взаємодії однаковими методами з різними об'єктами. Коли схожі об'єкти поводяться по-різному у схожих ситуаціях, у користувачів відбувається негативний перенос знань. Це гальмує вивчення інтерфейсу програми і призводить до втрати користувачем впевненості у своїх силах.

**Прогнозування** – збереження результатів взаємодії. Якщо користувач, виконуючи одні й ті ж дії, одержує різні результати, він більш схильний звинувачувати у цьому себе, ніж програмне забезпечення. Сумісність кроків і дій має витримуватись в усьому програмному продукті. Процеси реєстрації та навігації мають бути послідовними. Стандартні елементи інтерфейсу повинні поводитись однаково. Якщо результати можуть відрізнятись від того, що очікує користувач, слід інформувати його перед виконанням дії, а також надати йому опції виконання, скасування або здійснення іншої дії.

**Відношення** – естетична привабливість та цілісність. Деякі з сучасних програмних продуктів виглядають так, ніби вони розроблені різними людьми, які жодного разу не спілкувались між собою. Сумніви користувачів щодо цілісності ПЗ виникають, якщо в різних його частинах виявляється непослідовність кольорів, шрифтів, іконок та інших складових інтерфейсу. Приємний для сприйняття інтерфейс не приховує недолік функціональності програмного забезпечення.

**Передбачуваність** – заохочення вивчення. Задача проектувальників ІК – створити «дружній» інтерфейс, який заохочуватиме користувача досліджувати його складові і властивості без страху зробити щось невірно.

## 10.2 Практичне опанування теми 10

### 10.2.1 Передумови виконання завдань теми 10

Для початку виконання практичних завдань теми 10 треба підключити базу даних, яку було створено при виконанні практичних завдань за темою «Ознайомлення з основними особливостями СУБД Microsoft SQL Server. Створення бази даних та об'єктів бази даних».

При виконанні практичних завдань теми 10 передбачається використання інтегрованого середовища розробки Microsoft Visual Studio. У зв'язку із цим елементи інтерфейсу можуть мати відмінності порівняно із тими, що наведено далі у вигляді рисунків (особливо у випадку використання версії, локалізованої для використання певної національної мови). Ці відмінності не є принциповими та не впливають на виконання роботи та отримані результати.

Також передбачається, що виконавець має базові навички роботи у середовищі Microsoft Visual Studio та базові знання мови програмування C#.

### 10.2.2 Створення проекту та головної форми

Для створення проекту та головної форми необхідно виконати таку послідовність дій.

1. Запустити Microsoft Visual Studio.

2. Створити новий проект (для цього, наприклад, вибрати в головному меню пункт File, потім вибрати New, потім Project). Під час створення нового проекту вказати тип проекту (рисунок 10.3) та місце розміщення файлів проекту. В результаті на екрані з'явиться інтерфейс Visual Studio (рисунок 10.4). Як видно, застосунок складається поки що з однієї екранної форми. Цю форму будемо використовувати як головну форму програми.

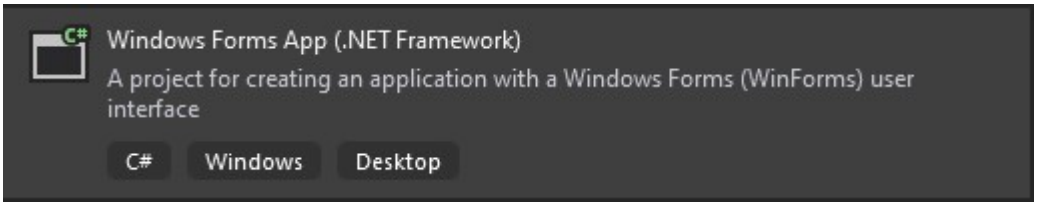



Рисунок 10.3

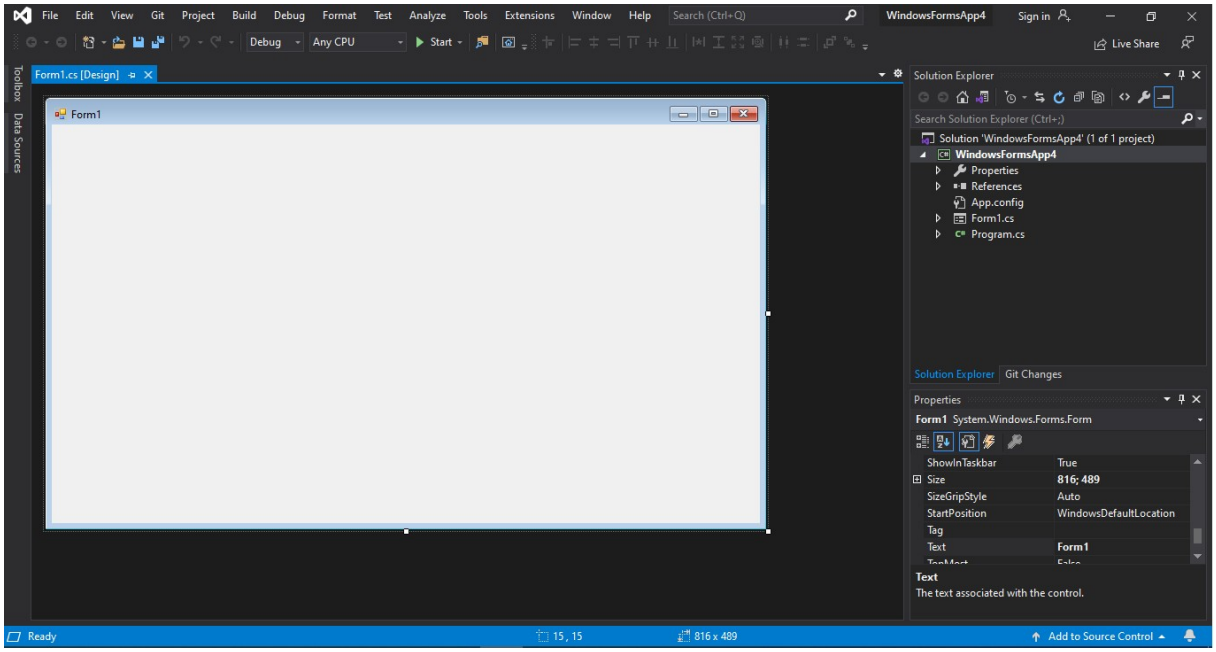



Рисунок 10.4

3. Виконати пробний запуск програми, для чого:

• у головному меню вибрати пункт Debug;

• у вертикальному меню вибрати пункт Start Without Debugging;

• в результаті на екрані з'явиться форма, яка не містить будь-яких елементів керування. Після запуску форму слід закрити.

4. Знайти властивість форми IsMdiContainer та встановити для неї значення True. Для властивості Text встановити значення Supply.

5. Встановити розміри форми. Принципового значення на цьому етапі роботи розміри форми не мають. Для прикладу можна для наступних властивостей форми встановити значення:

Size: 433; 316;  Location: 5; 5

6. Додати у форму головне меню. Для цього в панелі інструментів Toolbox вибрати компонент MenuStrip та перетягнути його на форму. Введіть назви трьох пунктів головного меню: Data, Reports, Exit. Для пункту Data сформувати підменю, що складається з пунктів: Suppliers, Supplied.

7. Запустити форму. Зовнішній вигляд форми може бути приблизно таким, як показано на рисунку 10.5 (геометричні розміри реальної форми будуть більшими).


Рисунок 10.5

8. Для пункту головного меню Exit створити функцію – обробник події, що дозволяє закрити програму. Для введення тексту функції подвійним клацанням миші відкрити вікно програмного коду та ввести текст функції (рисунок 10.6). Після цього запустити програму та перевірити коректність роботи пункту меню Exit.

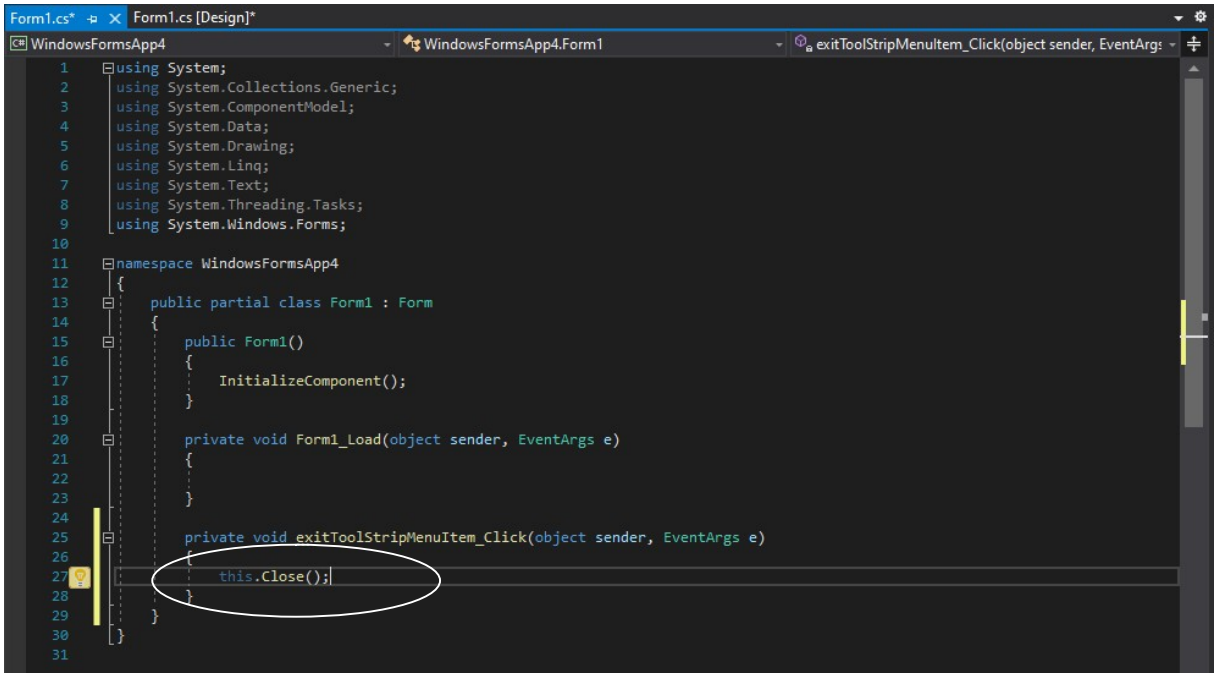



Рисунок 10.6

### 10.2.3 Створення найпростіших форм для роботи з даними

Розглянемо процес створення найпростіших форм для роботи з даними на прикладі форм, що забезпечують роботу з таблицями Suppliers, IndividualEntrepreneurs та LegalEntities. Для цього виконаємо таку послідовність дій.

1. Створити нове джерело даних. Для цього треба відкрити вікно Data Sources (якщо воно не відкрите, у головному меню вибрати пункт View, Other Windows і в підменю вибрати пункт Data Sources). Клацнути мишею по пункту Add New Data Source і у вікні Data Source Configuration Wizard вибрати Database як тип джерела даних. Натиснути кнопку Next.

2. У наступному вікні натиснути кнопку New Connection і в вікні Add Connection встановити параметри з'єднання. При цьому необхідно врахувати, що як Server name слід вводити дані, що відповідають комп'ютеру, на якому виконується робота. У полі Select or enter a database name слід ввести ім'я бази даних, наприклад, delivery. Після введення параметрів слід перевірити правильність підключення, натиснувши кнопку Test Connection. У разі успішного

тестового підключення слід натиснути кнопку ОК. Вікно Add Connection буде закрито і у вікні Data Source Configuration Wizard як активне буде встановлено нове з'єднання. Після цього необхідно натиснути кнопку Next.

3. Створене з'єднання можна зберегти, наприклад, з ім'ям deliveryConnectionString, натиснувши кнопку Next у наступному вікні. Потім потрібно вибрати об'єкти бази даних, які можуть використовуватись під час роботи (рекомендується вибрати усі). Натиснути кнопку Finish.

4. У результаті буде створено джерело даних deliveryDataSet, яке з'явиться у списку джерел даних у вікні Data Sources. У списку об'єктів цього джерела даних повинні бути всі об'єкти бази даних, створені в процесі виконання попередніх лабораторних робіт.

5. Використовуючи створене джерело даних, створимо найпростішу екранну форму, що дозволяє кінцевому користувачеві працювати з таблицею Suppliers. Для цього в головному меню виберемо пункт Project і у вертикальному меню – Add Form (Windows Forms)… . В результаті на екрані з'явиться вікно, що дозволяє визначити тип нової форми та її назву (рисунок 10.7). Для додавання форми до проекту потрібно натиснути кнопку Add.

6. В результаті до проекту буде додано нову форму. Вибравши у списку об'єктів джерела даних таблицю Suppliers, потрібно за допомогою миші перетягнути її на форму. В результаті на формі з'являться об'єкти типу DataGridView та BindingNavigator, за допомогою яких можна переглянути вміст даної таблиці та виконати операції маніпулювання даними (рисунок 10.8).

7. Тепер створену форму потрібно підключити до спільного меню програми. Для цього потрібно перейти на роботу з головною формою (Form1) (в режимі Design) і для пункту меню Suppliers (у вертикальному меню, що відповідає пункту Data головного меню) створити підменю. Перший пункт цього підменю слід назвати General Information (рисунок 10.9). Подвійне натискання миші на цьому пункті меню відкриє вікно коду,

в якому можна ввести текст функції, що забезпечує відкриття форми при виборі цього пункту меню (рисунок 10.10).

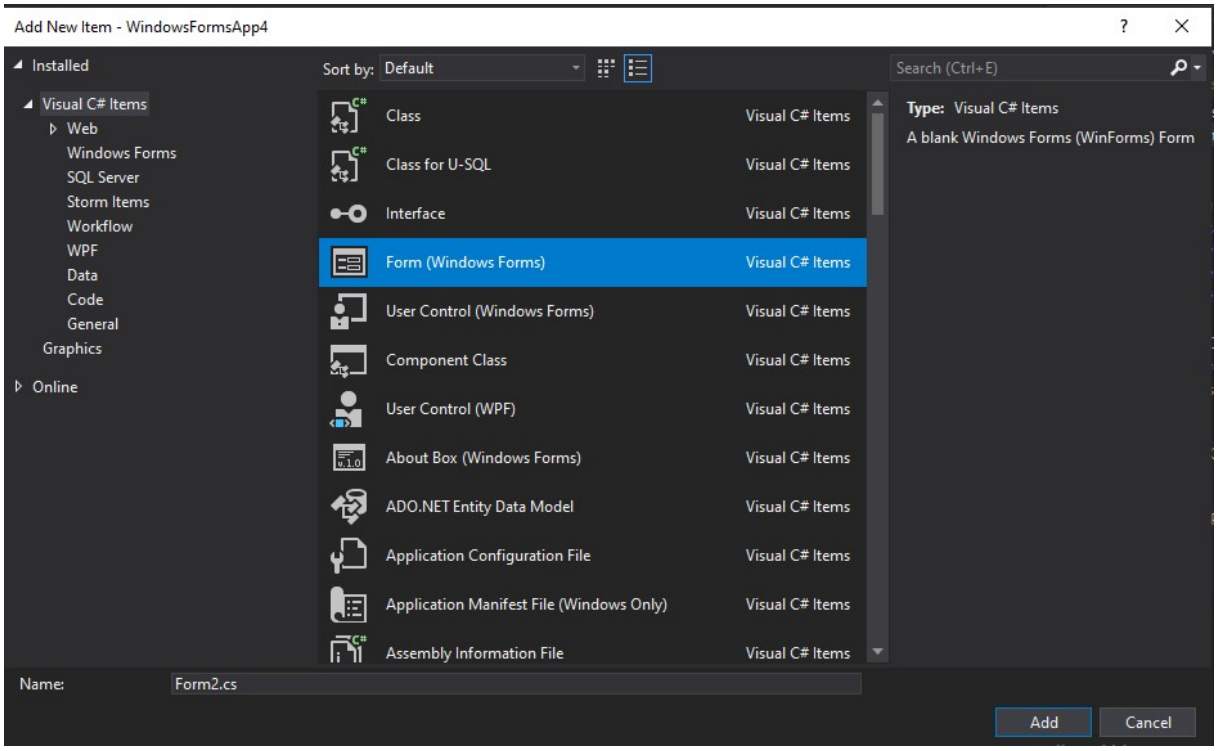



Рисунок 10.7

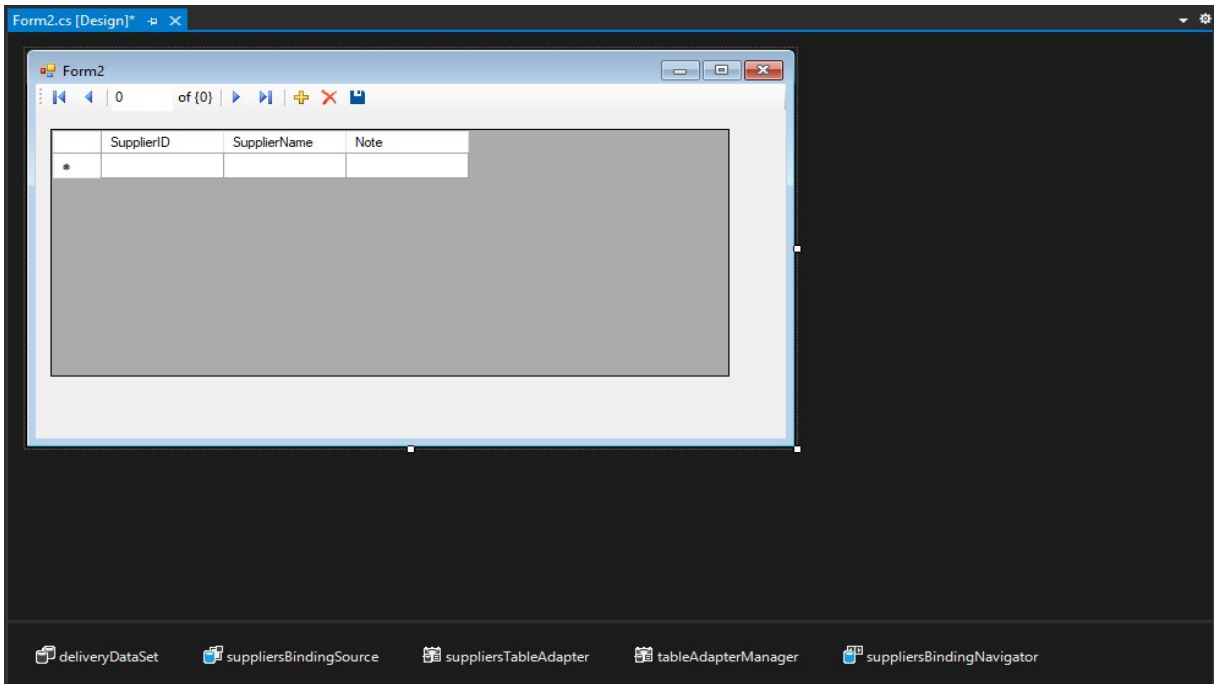



Рисунок 10.8

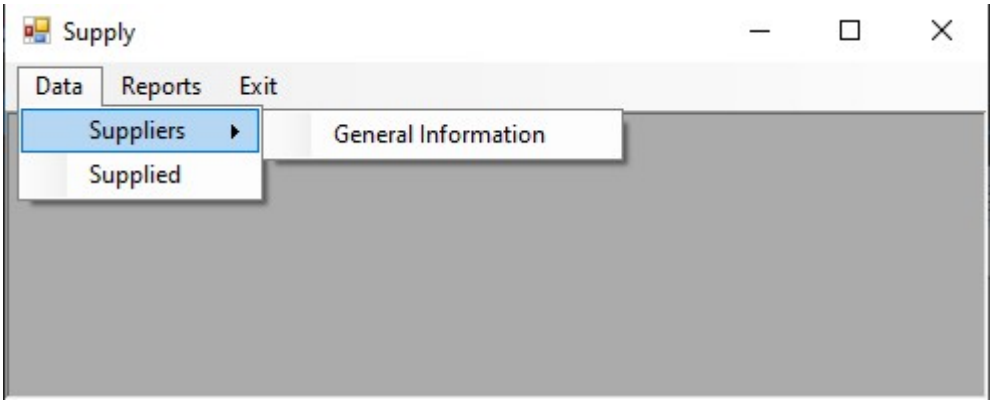



Рисунок 10.9

```
private void generalInformationToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 frm = new Form2();
    frm.Show();
}
```

Рисунок 10.10

8. Потрібно запустити застосунок та, вибравши відповідний пункт меню, вивести на екран форму, що містить загальні дані про постачальників (рисунок 10.11). Перевірити працездатність форми шляхом зміни даних про постачальників (додавання нового постачальника, видалення постачальника, зміна даних постачальника). Зміни обов'язково слід зберігати. Щоб зберегти зміни, потрібно використовувати кнопку Save. Перевірити виконання відповідних операцій можна за допомогою застосунку SQL Server Management Studio, а також шляхом закриття та повторного відкриття цієї форми.

9. Постачальники як суб'єкти підприємницької діяльності можуть бути як фізичними, так і юридичними особами. Для того щоб працювати з такою інформацією, створимо окремі форми. Розглянемо процес створення такої форми з прикладу форми, призначеної для роботи з даними постачальників – фізичних осіб. Насамперед змінимо меню головної форми, доповнивши його пунктами IndividualEntrepreneurs та LegalEntities (рисунок 10.12). Для пункту меню IndividualEntrepreneurs зробимо форму, загалом аналогічну до форми, підключеної до пункту General Information.

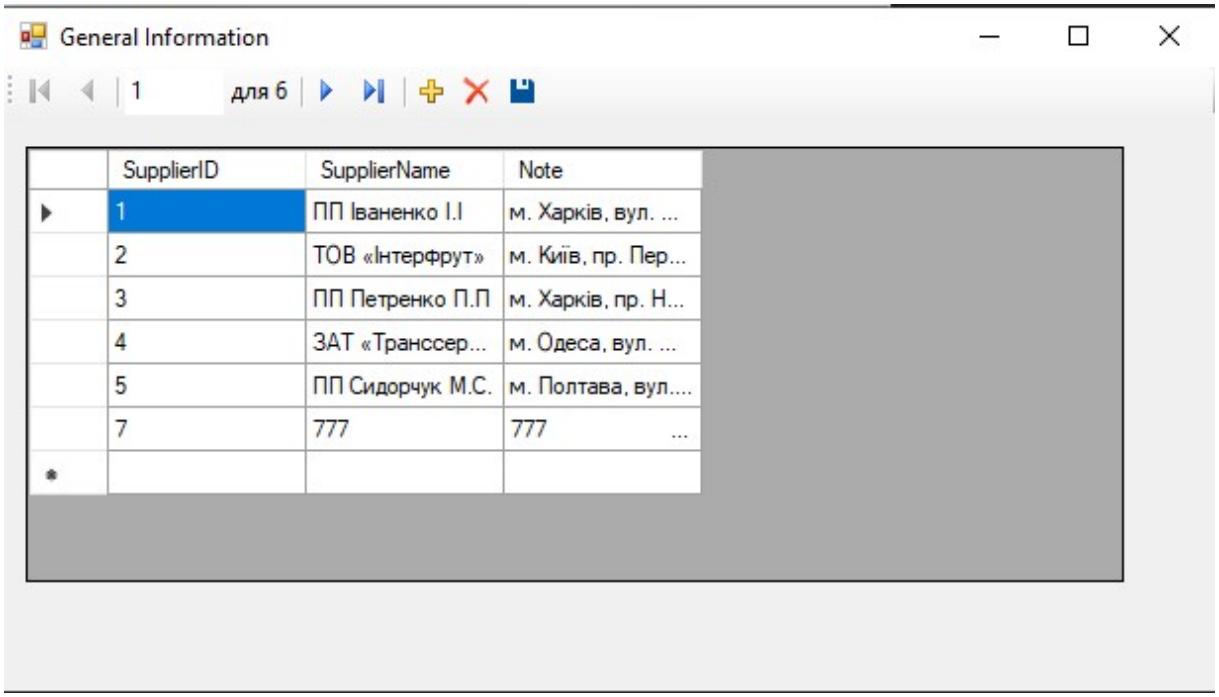



Рисунок 10.11

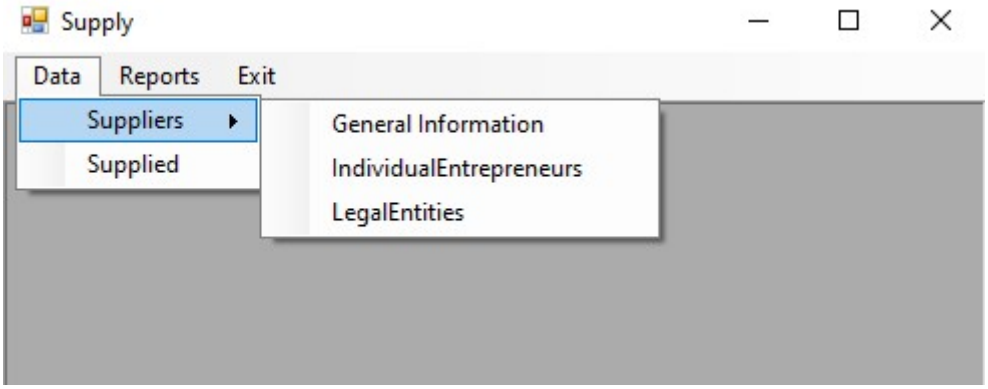



Рисунок 10.12

10. Послідовність дій при створенні такої форми аналогічна послідовності дій при створенні форми для роботи із загальними відомостями про постачальників. Припустимо, що при створенні нової форми їй було присвоєно ім'я Form3.cs. Після створення форми та розміщення на ній об'єктів управління її необхідно підключити до головної форми. Після цього застосунок потрібно запустити та перевірити

працездатність нової форми. Її зовнішній вигляд може бути, наприклад, таким, як показано на рисунку 10.13. Очевидно, що користувач, який працює з цією формою, повинен пам'ятати коди відповідних постачальників, що не завжди можливо. У зв'язку з цим модернізуємо форму таким чином, щоб введення коду постачальника здійснювалося не вручну, а шляхом вибору коду зі списку кодів, що вже є в таблиці Suppliers.

11. Треба відкрити цю форму в режимі Design, клацнути по об'єкту DataGridView і у вікні властивостей знайти властивість Columns. Натиснути кнопку «…» для того, щоб отримати доступ до стовпців DataGridView. В результаті на екрані з'явиться вікно, що містить список стовпців та їх властивості (рисунок 10.14). Виберемо стовпець КодПостачальника та властивість ColumnType (рисунок 10.15). Встановимо для цього стовпця тип DataGridViewComboBoxColumn (рисунок 10.14).

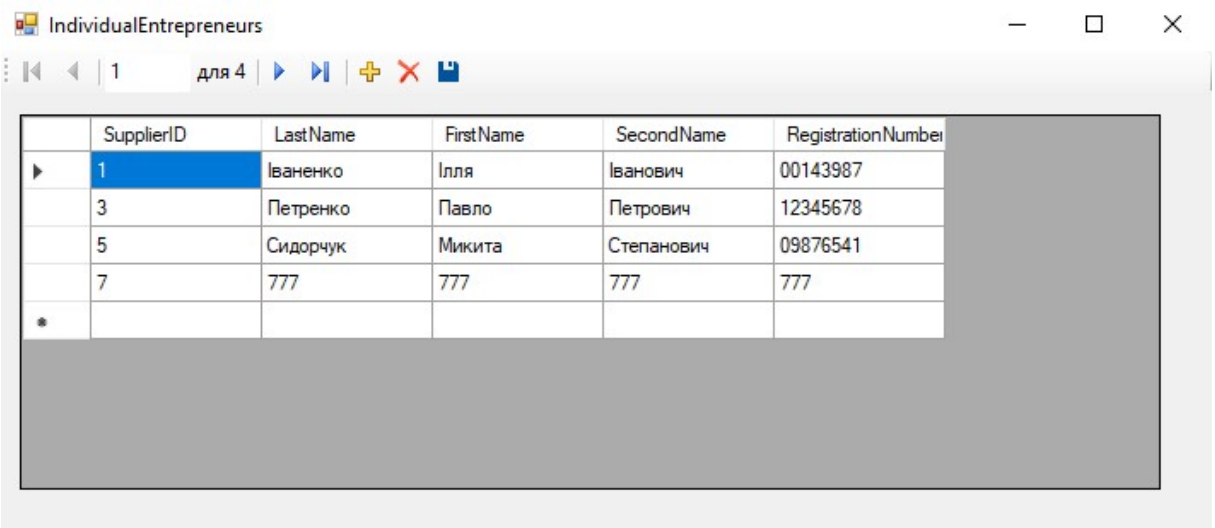



Рисунок 10.13

12. Встановимо для цього стовпця значення властивості DataSource. Для цього викличемо список джерел даних та виберемо джерело даних, яке відповідає таблиці Suppliers (рисунок 10.15).

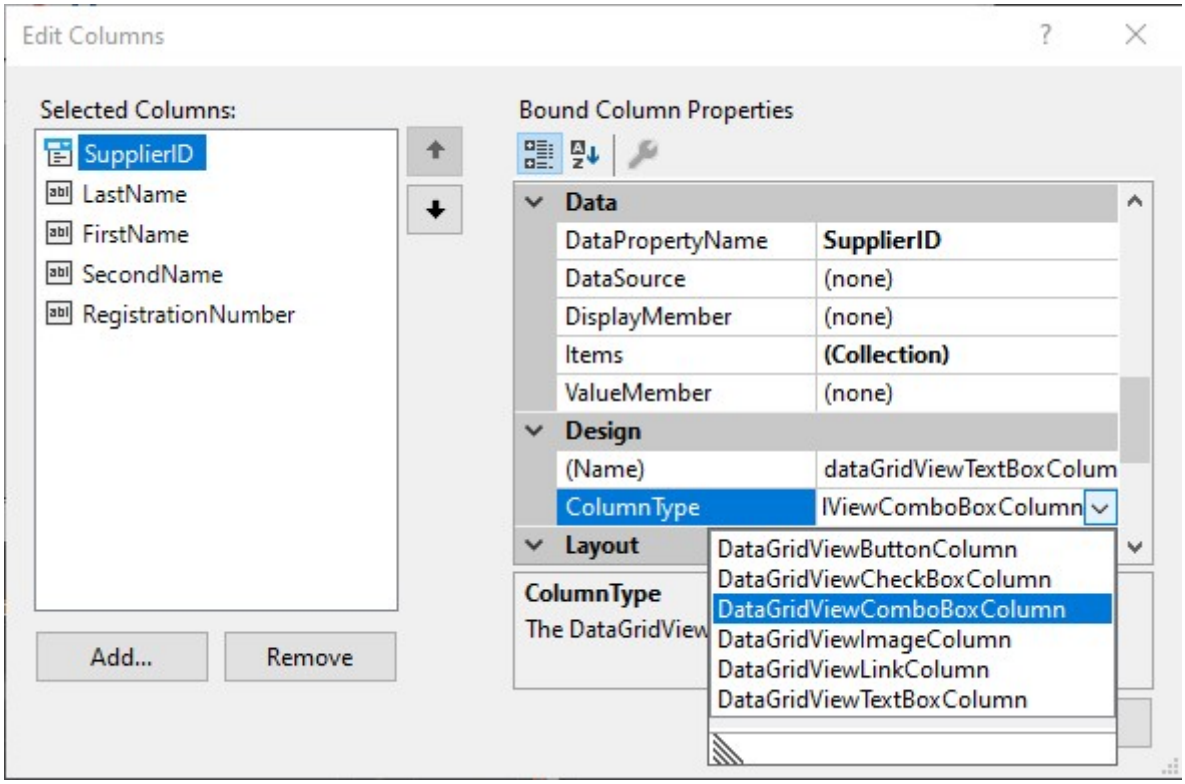



Рисунок 10.14

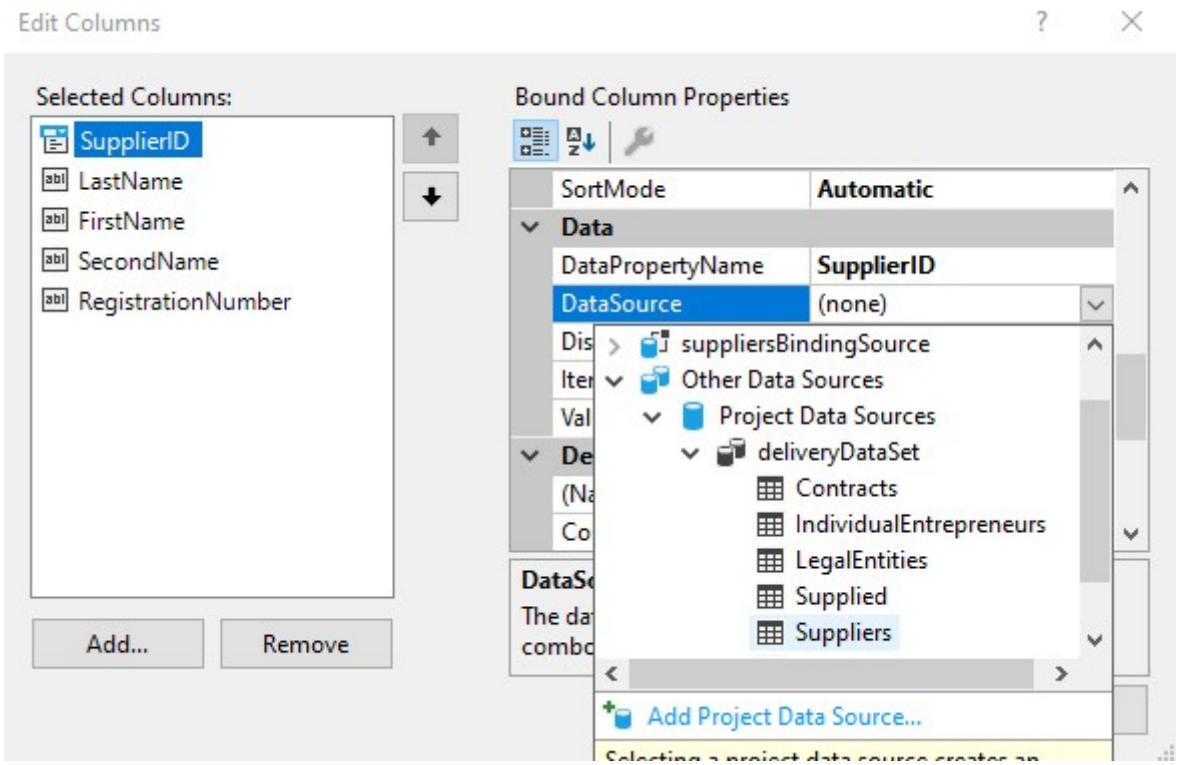



Рисунок 10.15

13. Для властивості DisplayMember встановити значення SupplierID (рисунок 10.16). Після цього вікно Edit Columns закрити натиснувши кнопку OK. В результаті в стовпці SupplierID має з'явитися кнопка поля зі списком.

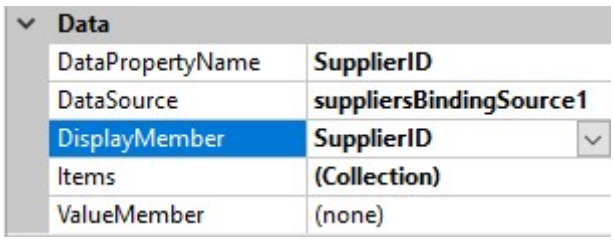



Рисунок 10.16

14. Запустити застосунок та перевірити працездатність зміненої форми. Її зовнішній вигляд може бути, наприклад, таким, як показано на рисунку 10.17.

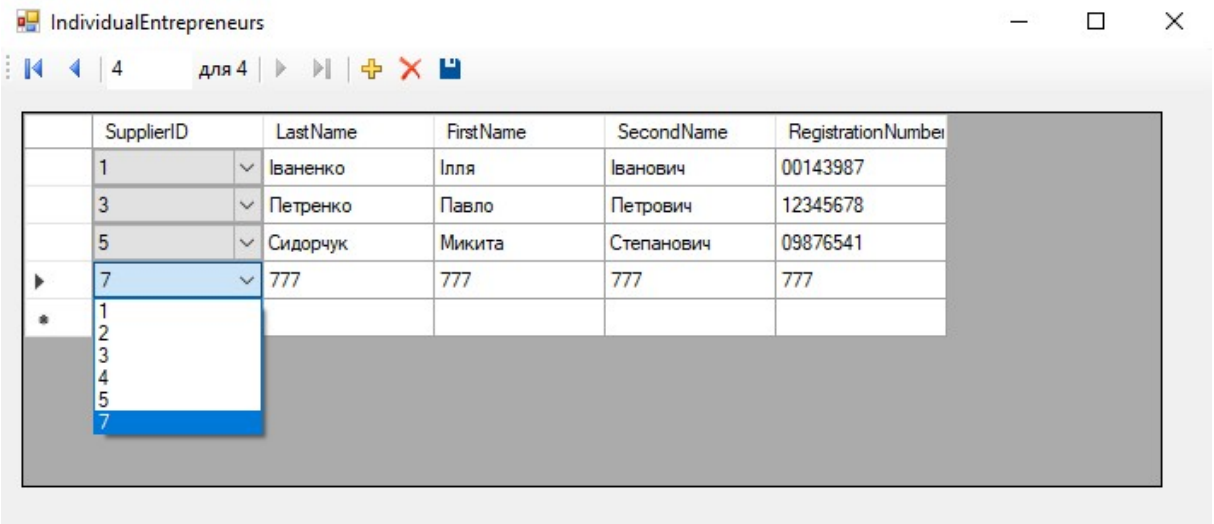



Рисунок 10.17

15. Перевірити працездатність форми під час введення даних. При цьому можна перевірити працездатність не тільки форми, але й раніш створеного тригера. Для цього потрібно спробувати ввести довільні дані про постачальника 2 як про фізичну особу (рисунок 10.18) та спробувати

зберегти ці дані. В цьому випадку на екран буде виведено повідомлення, пов'язане з роботою тригера (рисунок 10.19). Новий запис не буде збережено.

16. Аналогічно можна розробити та підключити у застосунок форму, що забезпечує роботу з даними про постачальників – юридичних осіб.

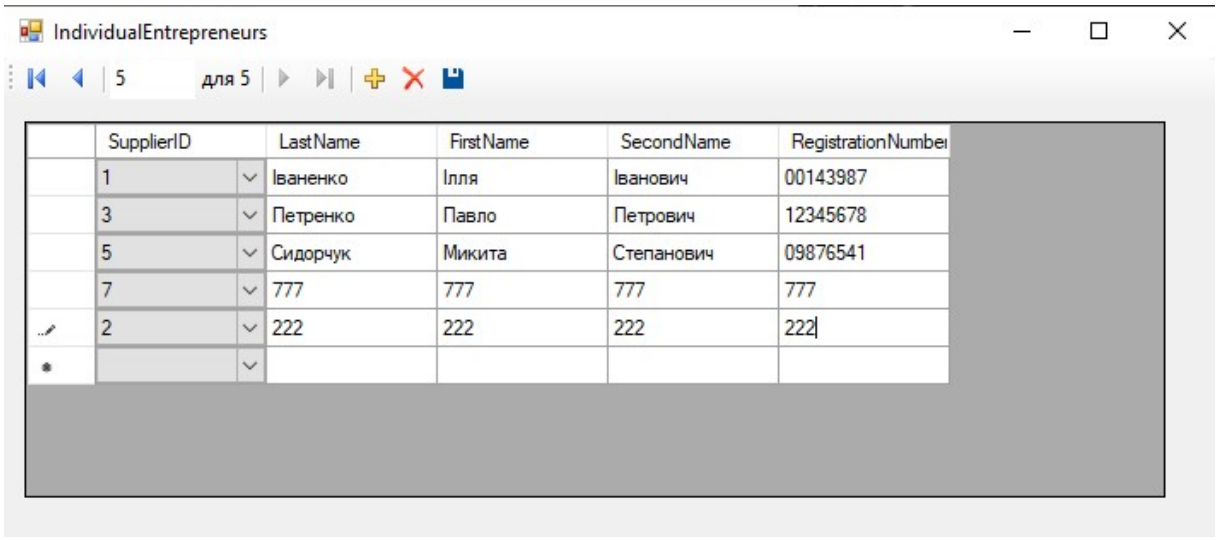



Рисунок 10.18

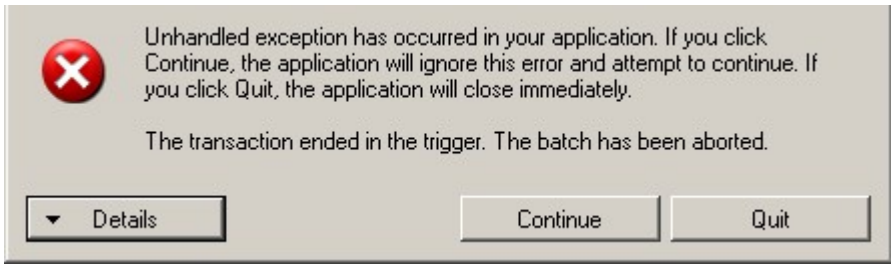



Рисунок 10.19

У цілому розроблені форми є працездатними, але вони забезпечують користувачеві лише мінімальний набір засобів до роботи з даними. Крім того, ці форми мають дуже мало засобів контролю за діями користувача. Наприклад, при натисканні кнопки видалення запису, застосунок не вимагає від користувача підтвердження видалення, отже помилкове

видалення може призвести до втрати даних. Також, якщо користувач після вводу нових даних не натисне кнопку зберігання (наприклад, забувши про це), та закриє форму, то нові дані не будуть збережені та ін.

### 10.2.4 Створення форми, що забезпечує роботу з кількома таблицями

Розглянемо процес створення форми, за допомогою якої кінцевий користувач буде мати можливість переглядати та модифікувати дані про договори та продукцію, яку було закуплено на підставі цих договорів.

1. Створити нову форму, наприклад, з ім'ям Form5.cs. Для властивості Text встановити значення Supplied.

2. Розмістити на формі об'єкт TabControl (для цього потрібно вибрати цей об'єкт у панелі Toolbox та перетягнути на форму). Для властивості Name встановити значення tabControl1. Об'єкт повинен містити дві вкладки з іменами tabPage1 та tabPage2 відповідно. Для властивості Text цих вкладок встановити значення List of contracts та New contract відповідно. Підключити нову форму до пункту меню Supplied. Текст функції, що забезпечує таке підключення, наведено на рисунку 10.20. Запустити застосунок та перевірити працездатність нової форми. Форма може мати вигляд, який наведено на рисунку 10.21. Закрити програму і повернутися в режим Design для Form5.cs.

```
private void suppliedToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form5 frm = new Form5();
    frm.Show();
}
```

Рисунок 10.20

3. Розмістити на вкладці tabPage1 об'єкт типу DataGridView. В результаті форма (в режимі Design) матиме вигляд, наведений на рисунку 10.22). Для об'єкта встановити ім'я dataGridView1.

4. Клацнути по кнопці у верхньому правому кутку об'єкта dataGridView1 для того, щоб відкрити вікно DataGridView Tasks (рисунок 10.23). За допомогою цього вікна визначити джерело даних для dataGridView1 (рисунок 10.23), вибравши його зі списку джерел даних.

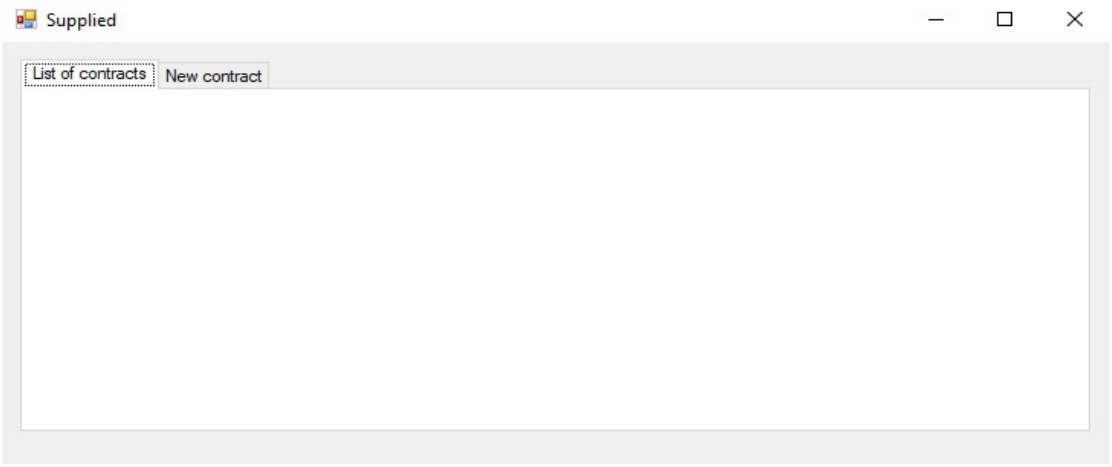



Рисунок 10.21

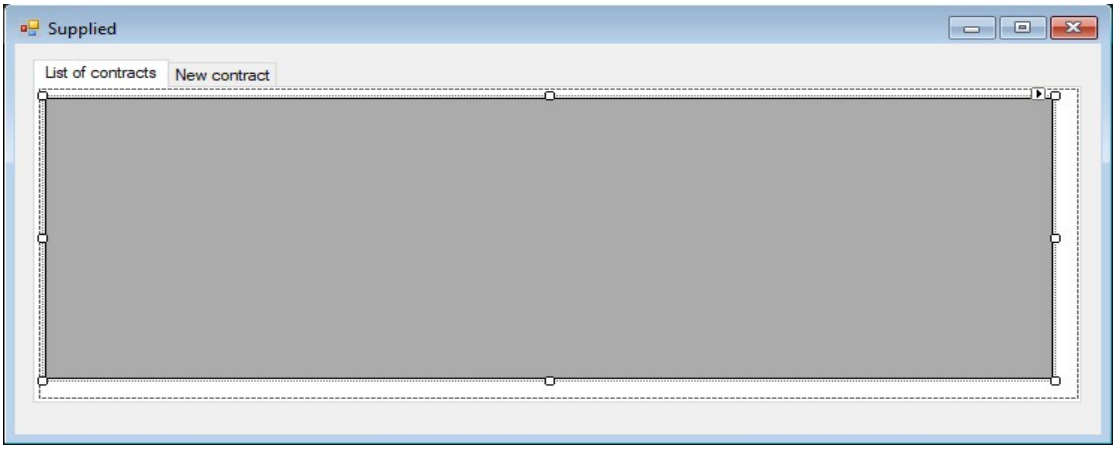



Рисунок 10.22

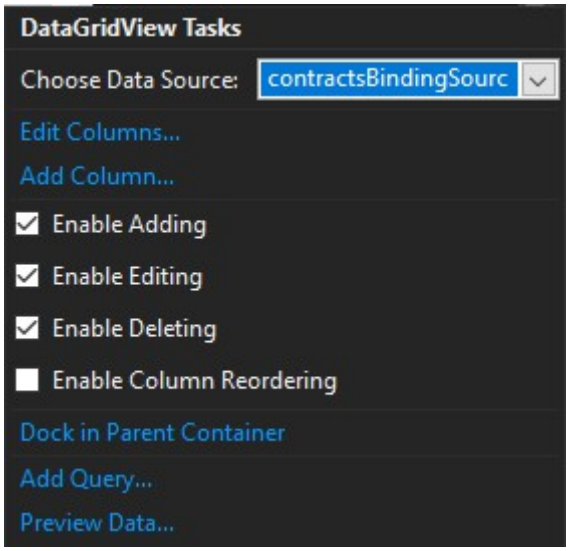



Рисунок 10.23

5. Перевірити працездатність зміненої форми. Форма може мати вигляд, аналогічний наведеному на рисунку 10.24. Закрити застосунок і повернутися в режим Design для Form5.cs. Встановити для властивості об'єкта ReadOnly dataGridView1 значення True.

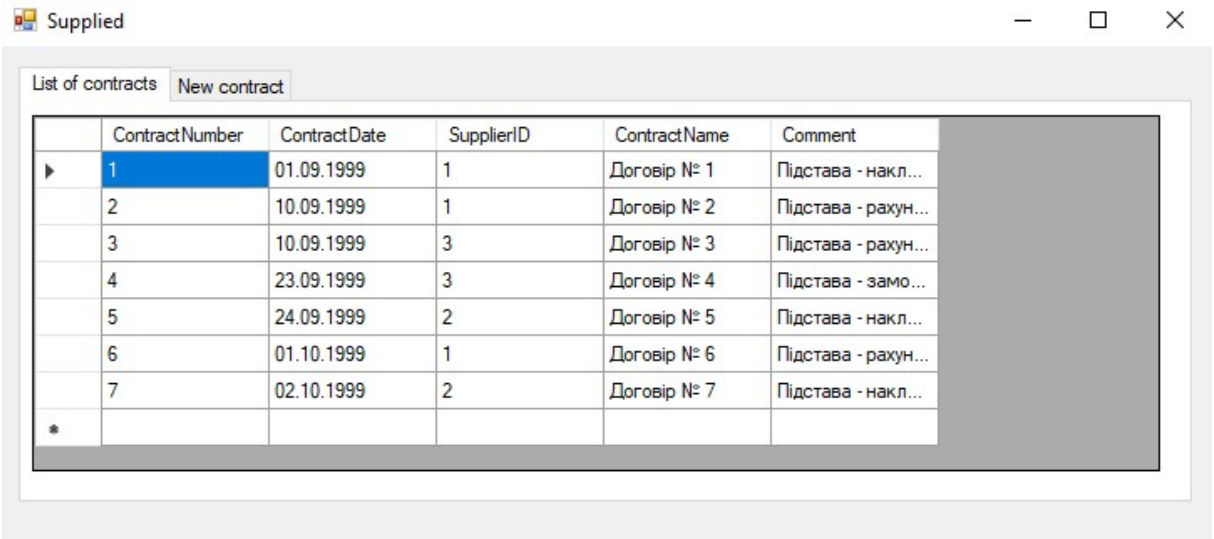



Рисунок 10.24

6. Як видно з рисунку 10.24, створений список договорів не цілком зручний для роботи. Це, зокрема, стосується інформації про постачальника. Припустимо, що від користувачів надійшла вимога, щоб

замість коду постачальника у списку договорів виводилася назва постачальника, а поруч, у дужках для постачальників – фізичних осіб має виводитися прізвище, ім'я та по-батькові, а для юридичних осіб – податковий номер. Щоб отримати такі дані, створимо у базі даних представлення, що дозволяє сформувати таку інформацію про постачальників. Текст визначального запиту такого представлення може мати вигляд, наведений на рисунку 10.25. Таке представлення необхідно зробити за допомогою SQL Server Management Studio та зберегти з ім'ям View_3. Потім у вікні Data Sources потрібно клацнути правою кнопкою миші по джерелу даних deliveryDataSet і в меню вибрати пункт Configure DataSet with Wizard ... У вікні потрібно повторно відзначити пункт Views. В результаті новостворене представлення має з'явитися у списку об'єктів джерела даних. Змінене джерело даних треба зберігти.

```
SELECT     dbo.Suppliers.SupplierID, RTRIM(CAST(dbo.Suppliers.SupplierName AS char(20))) + ' (' +
                      ISNULL(RTRIM(dbo.IndividualEntrepreneurs.LastName) + ' ' + RTRIM(dbo.IndividualEntrepreneurs.FirstName)
                      + ' ' + RTRIM(dbo.IndividualEntrepreneurs.SecondName), RTRIM(dbo.LegalEntities.TaxNumber)) + ')' AS Supplier
FROM       dbo.Suppliers LEFT OUTER JOIN
             dbo.LegalEntities ON dbo.Suppliers.SupplierID = dbo.LegalEntities.SupplierID LEFT OUTER JOIN
             dbo.IndividualEntrepreneurs ON dbo.Suppliers.SupplierID = dbo.IndividualEntrepreneurs.SupplierID
```

Рисунок 10.25

7. Для об'єкта dataGridView1 відкрити вікно DataGridView Tasks та вибрати пункт Edit Columns… Змінити назви стовпців, якщо це потрібно (це можна зробити, змінивши властивість HeaderText для кожного стовпця (рисунок 10.26)). Також можна змінити ширину стовпців (це можна зробити, змінивши властивість Width для кожного стовпця). Ширину шпальт встановити виходячи з реальних розмірів даних.

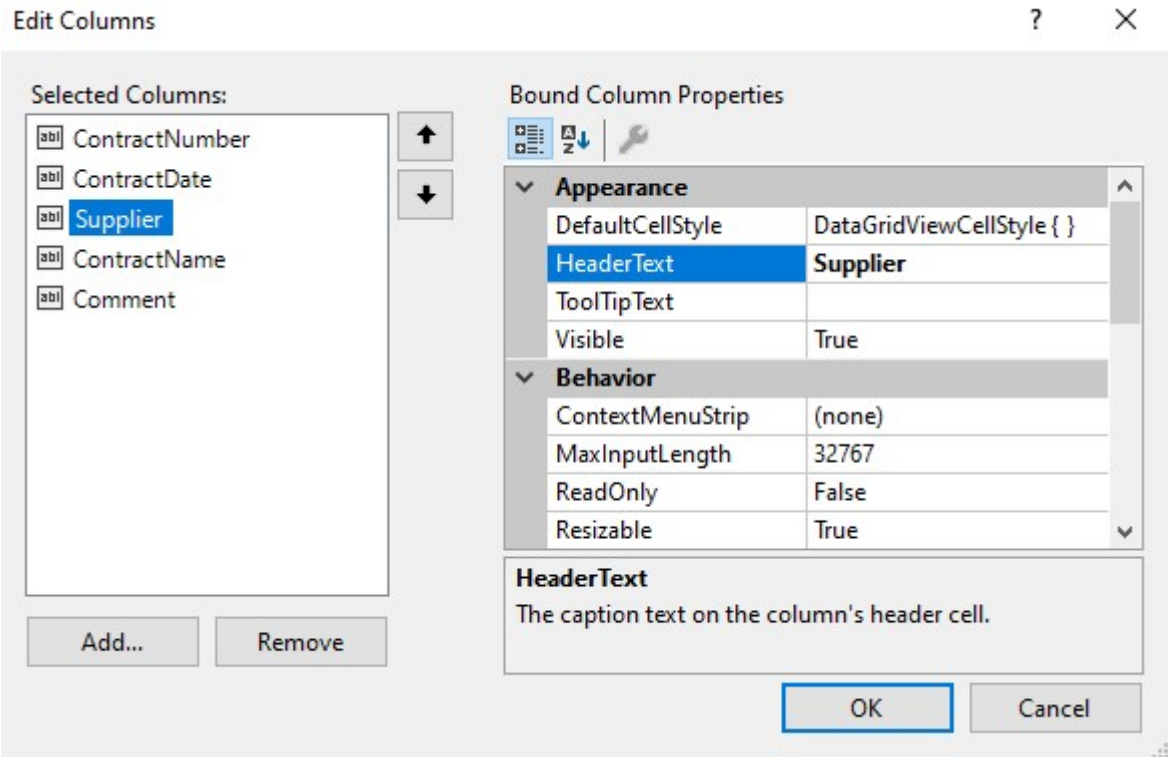



Рисунок 10.26

8. Вибрати стовпець Supplier та властивість ColumnType. Установіть для цього стовпця тип DataGridViewComboBoxColumn. Встановити для цього стовпця значення властивості DataSource. Для цього викликати список джерел даних та вибрати джерело даних, що відповідає представленню View_3 (рисунок 10.27). Для властивості DisplayMember встановити значення Supplier, а властивості ValueMember – SupplierID (рисунок 10.27). Після цього вікно Edit Columns закрити натиснувши кнопку OK. В результаті в стовпці Supplier має з'явитися кнопка поля зі списком. Перевірити працездатність зміненої форми. Форма може мати вигляд, аналогічний наведеному на рисунку 10.28. Закрити програму і повернутися в режим Design для Form5.cs.

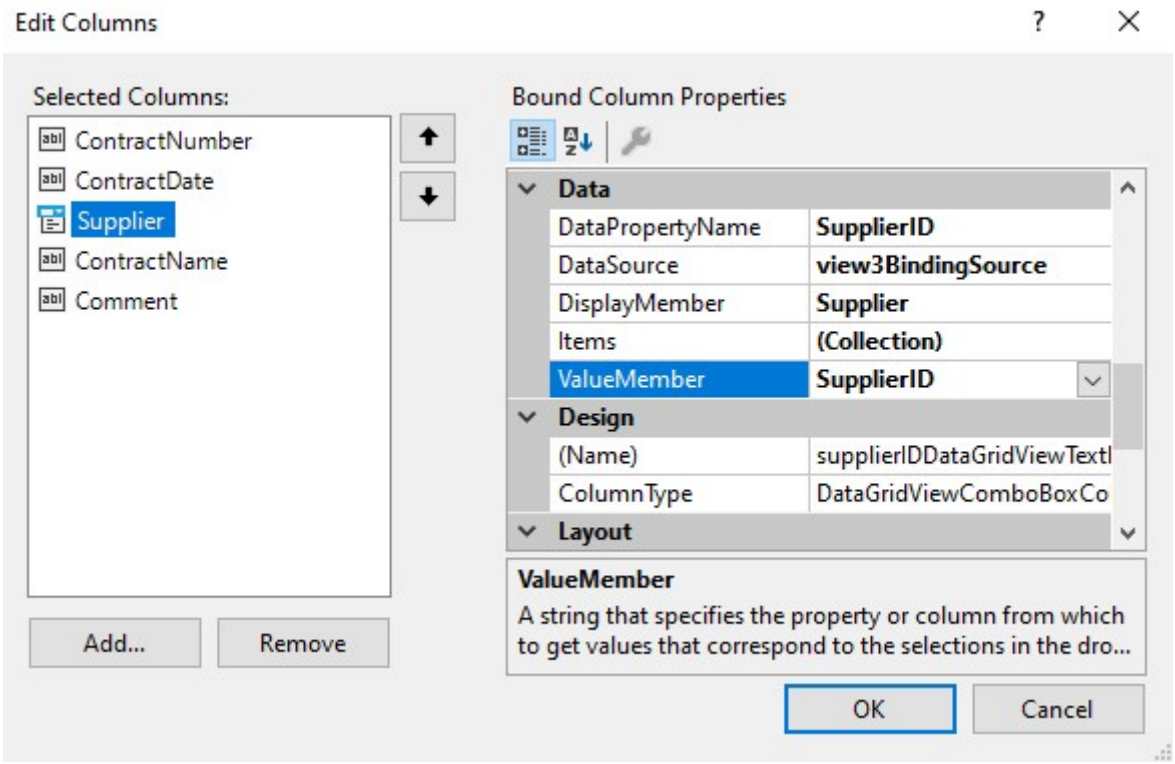



Рисунок 10.27

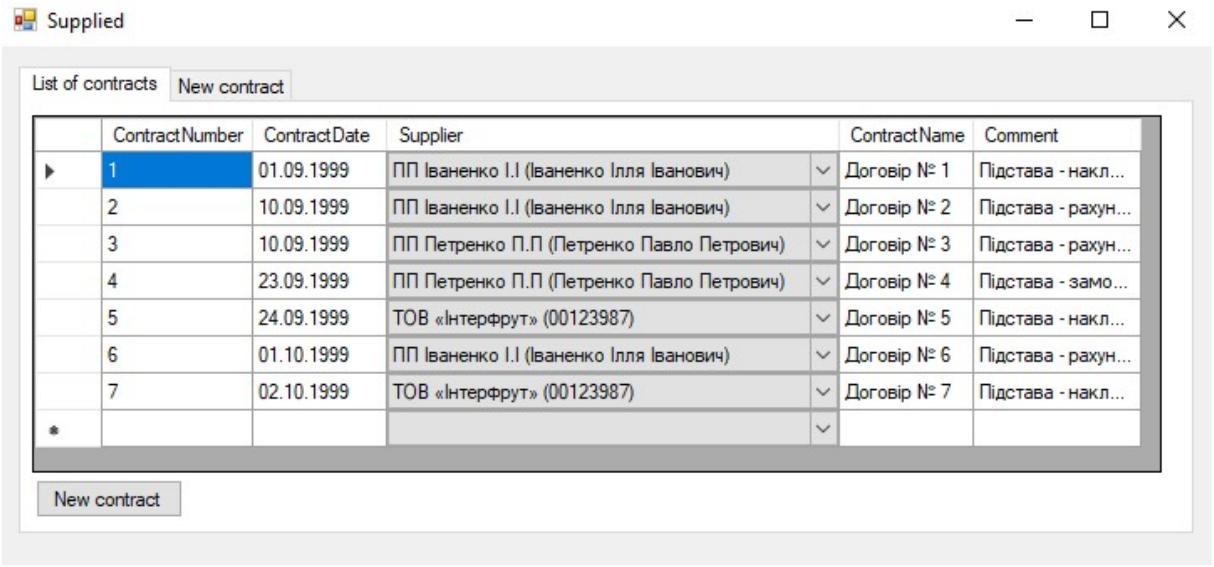



Рисунок 10.28

9. Список договорів у цій формі можна лише переглядати. Зміна списку договорів неможлива. Щоб забезпечити можливість користувачеві

створювати нові договори, внесемо зміни у форму. На вкладці List of contracts створимо кнопку. Для цього в панелі Toolbox виберемо об'єкт типу Button та розмістимо його на формі (рисунок 10.28). Для властивості Text необхідно встановити значення New contract. Для події Click цієї кнопки потрібно створити функцію – обробник події. Текст цієї функції наведено рисунку 10.29. Перевірити працездатність зміненої форми. В результаті натискання кнопки має відкриватися вкладка New contract. Закрити програму і повернутися в режим Design для Form5.cs.

Примітка. З функціональної точки зору наявність кнопки New contract у формі не є обов'язковою. Вона додана з метою ілюстрації можливостей керування об'єктами форми.

```
private void button1_Click(object sender, EventArgs e)
{
    this.tabControl1.SelectedTab = tabPage2;
}
```

Рисунок 10.29

10. Відкрити вкладку New contract (tabPage2). На цій вкладці розмістити чотири об'єкти типу Label, за допомогою яких формуються коментарі для полів введення даних (рисунок 10.30). Для введення дати укладання договору можна використовувати об'єкт типу DateTimePicker. Для цього об'єкта встановлюється ім'я dateTimePicker1. Для введення назви договору та коментаря до договору необхідно використовувати об'єкт типу TextBox. Для цих об'єктів встановлюються назви textBox1 та textBox2. Для введення даних про постачальника необхідно використовувати об'єкт типу ComboBox. Для цього об'єкта встановлюється ім'я comboBox1. Також на вкладці потрібно розмістити два об'єкти типу Button. Для властивості Text цих кнопок потрібно встановити значення Save та Cancel. Для імен цих об'єктів встановити значення button2 та button3 відповідно.

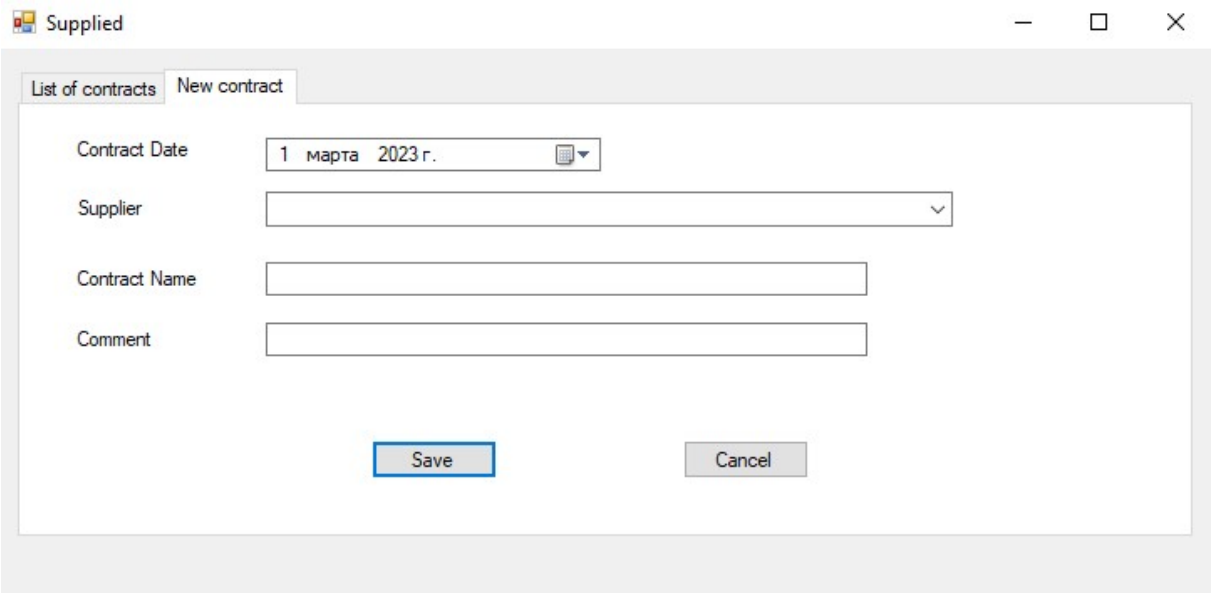



Рисунок 10.30

11. Для об'єкта comboBox1 потрібно встановити джерело даних – список постачальників. Для цього потрібно клацнути мишею по об'єкту і натиснути кнопку, що з'явиться у верхньому правому кутку об'єкта. Внаслідок цього з'явиться вікно ComboBox Tasks. У цьому вікні потрібно увімкнути прапорець для пункту Use data bound items (рисунок 10.31), а потім встановити джерело даних, а також значення для властивостей DisplayMember та ValueMember (рисунок 10.31). Як джерело даних можна використовувати уявлення View_3.

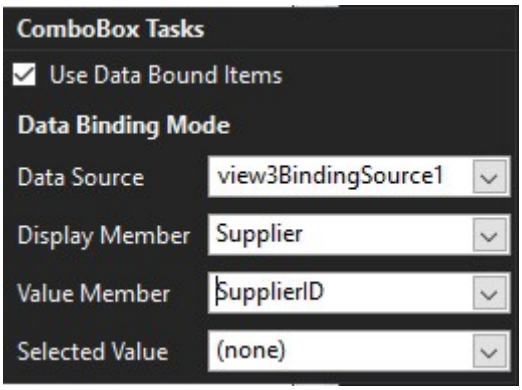



Рисунок 10.31

12. Для об'єкта button2 (кнопка Save) створити функцію – обробник події Click. Текст функції наведено рисунку 2.32.

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        int codeSupplier = int.Parse(this.comboBox1.SelectedValue.ToString());
        DateTime datedgvr = Convert.ToDateTime(this.dateTimePicker1.Text);
        string dgvrName = Convert.ToString(this.textBox1.Text);
        string dgvrComment = Convert.ToString(this.textBox2.Text);
        conn = new SqlConnection();
        conn.ConnectionString = "integrated security=SSPI;data source=\".\"; persist security info=False; initial catalog = delivery";
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText = "INSERT INTO Contracts (SupplierID,ContractDate,ContractName,Comment) VALUES (@codeSupplier,@datedgvr,@dgvrName,@dgvrComment)";
        myCommand.Parameters.Add("@codeSupplier", SqlDbType.Int, 4);
        myCommand.Parameters["@codeSupplier"].Value = codeSupplier;
        myCommand.Parameters.Add("@datedgvr", SqlDbType.DateTime, 8);
        myCommand.Parameters["@datedgvr"].Value = datedgvr;
        myCommand.Parameters.Add("@dgvrName", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@dgvrName"].Value = dgvrName;
        myCommand.Parameters.Add("@dgvrComment", SqlDbType.NVarChar, 50);
        myCommand.Parameters["@dgvrComment"].Value = dgvrComment;

        int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
        if (UspeshnoeIzmenenie != 0)
        {
            MessageBox.Show("Changes complited", "Record modified");
        }
        else
        {
            MessageBox.Show("Changes didn't complited", "Record modified");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
    finally
    {
        conn.Close();
    }
    this.contractsTableAdapter.Fill(this.deliveryDataSet.Contracts);
    this.tabControl1.SelectedTab = tabPage1;
}
}
```

Рисунок 10.32

Крім введення тексту функції, також потрібно змінити програмний код. Зміни наведено на рисунку 10.33 (цей код знаходиться на початку програмного коду форми).

13. Перевірити працездатність зміненої форми. Для цього потрібно запустити застосунок та ввести дані нового договору (наприклад, як на рисунку 10.34). Натиснути кнопку Save. У разі успішного введення даних на екран буде виведено повідомлення (рисунок 10.35). В результаті вкладку New contract буде закрито та відкрито вкладку List of contracts, в якій з'явиться новий договір (рисунок 10.36).

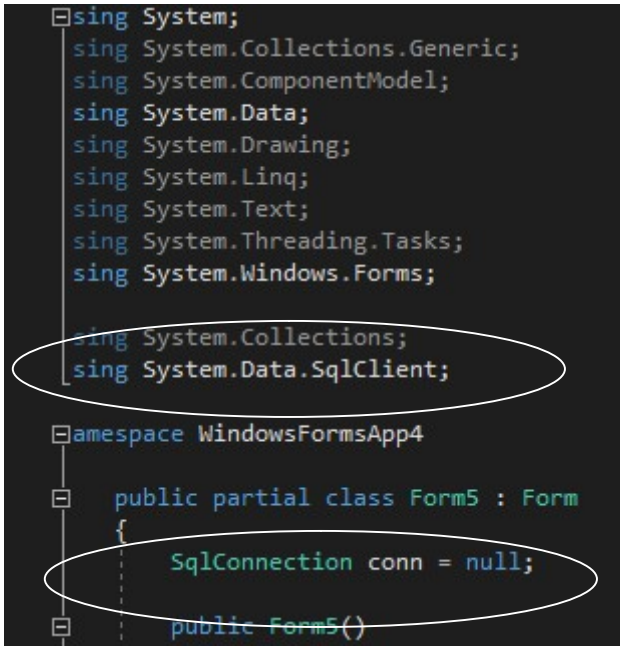



Рисунок 10.33

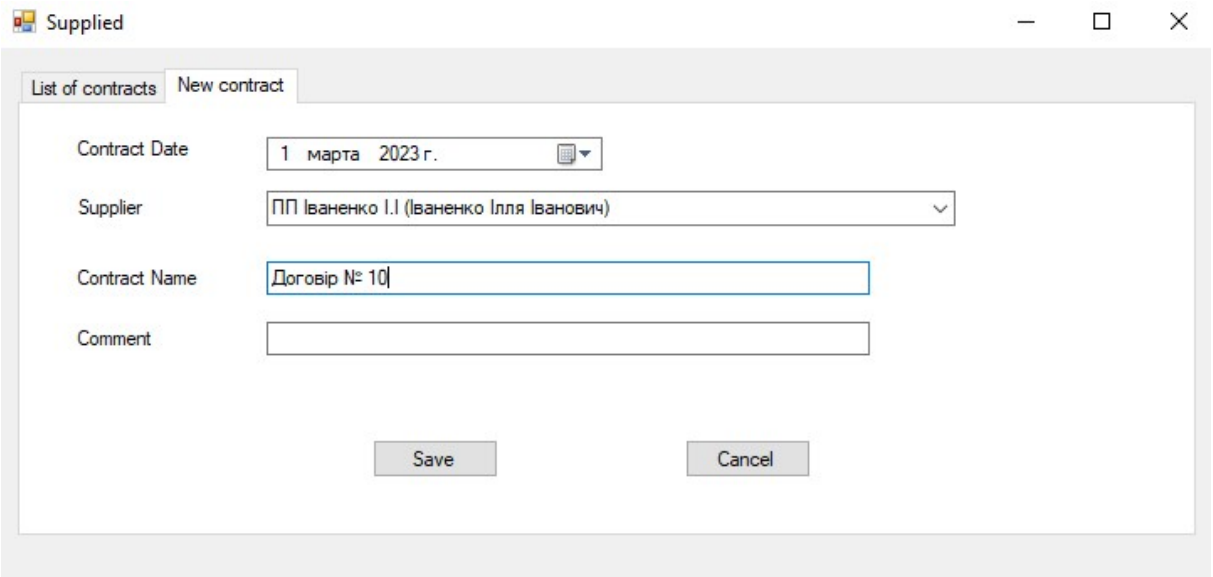



Рисунок 10.34

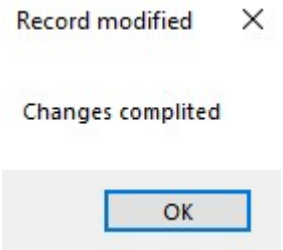



Рисунок 10.35

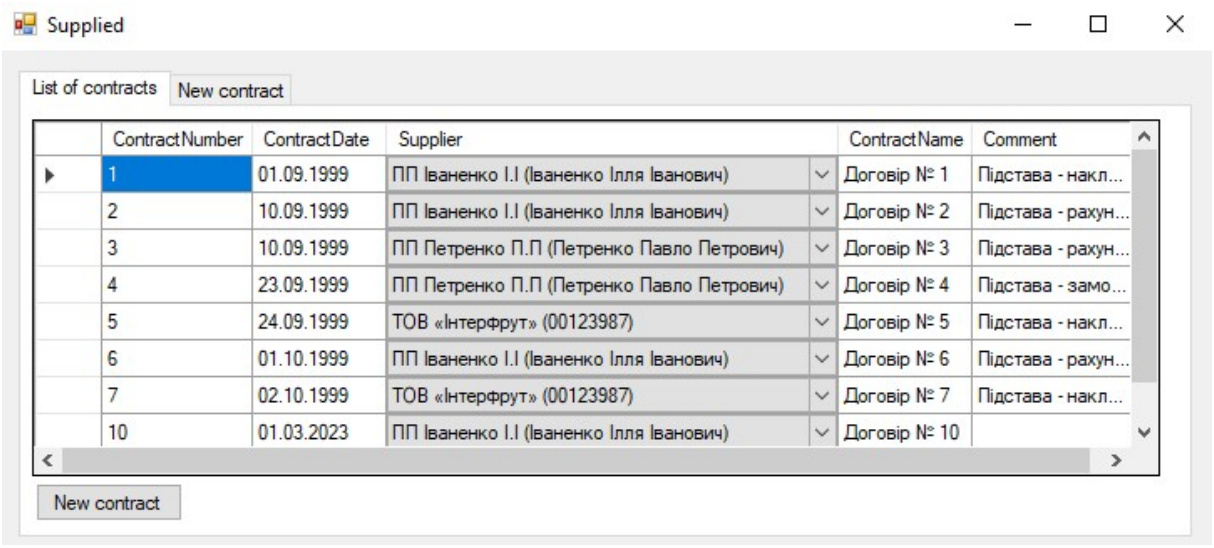



Рисунок 10.36

14. Закрити програму та повернутися в режим Design для Form5.cs. На вкладці New contract для об'єкта button3 (кнопка Cancel) створити функцію – обробник події Click. Текст функції наведено на рисунку 10.37. Перевірити роботу цієї кнопки (при її натисканні закривається вкладка New contract та відкривається вкладка List of contracts без додавання нового договору).

```
private void button3_Click(object sender, EventArgs e)
{
    this.tabControl1.SelectedTab = tabPage1;
}
```

Рисунок 10.37

15. У процесі роботи з даними про закупівлі продукції може виникнути необхідність видалення раніше укладених договорів. Для цього на вкладці List of contracts створимо відповідну кнопку. Для властивості Text цієї кнопки необхідно встановити значення Delete contract. Для цього об'єкта встановити ім'я button4. Для об'єкта button4 створити функцію – обробник події Click. Текст функції наведено на рисунку 10.38.

16. Перевірити роботу цієї кнопки. Для видалення договору потрібно вибрати договір у списку договорів (наприклад, останній створений договір) та натиснути кнопку Delete contract. В результаті на екрані з'явиться вікно (рисунок 10.39), за допомогою якого потрібно підтвердити видалення договору або відмовитись від цієї операції. При натисканні кнопки Так договір буде видалено та список договорів буде оновлено з урахуванням видалення договору. Таким чином, розроблена форма дозволяє бачити список договорів, додавати до бази даних нові договори та видаляти раніше створені договори.

```
private void button4_Click(object sender, EventArgs e)
{
    int dgvrNumber = int.Parse(dataGridView1.Rows[dataGridView1.CurrentCell.RowIndex]
        .Cells["ContractNumberDataGridViewTextBoxColumn"].Value.ToString());
    DialogResult result = MessageBox.Show("Contract Number " + Convert.ToString(dgvrNumber) +
        " will be deleted. Are you sure?", "Warning", MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
        MessageBoxDefaultButton.Button2);
    switch (result)
    {
        case DialogResult.Yes:
        {
         try
          {
           conn = new SqlConnection();
           conn.ConnectionString = "integrated security=SSPI;data source=\".\"; " +
                      "persist security info=False; initial catalog = delivery";
           conn.Open();
           SqlCommand myCommand = conn.CreateCommand();
           myCommand.CommandText = "DELETE FROM Contracts WHERE ContractNumber = @dgvrNumber ";
           myCommand.Parameters.Add("@dgvrNumber", SqlDbType.Int, 4);
           myCommand.Parameters["@dgvrNumber"].Value = dgvrNumber;

           int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
           if (UspeshnoeIzmenenie != 0)
              {
               MessageBox.Show("Changes complited", "Record modified");
              }
              else
              {
               MessageBox.Show("Changes didn't complited", "Record modified");
              }
          }
          catch (Exception ex)
          {
           MessageBox.Show(ex.ToString());
          }
          finally
          {
           conn.Close();
          }
          this.contractsTableAdapter.Fill(this.deliveryDataSet.Contracts);
          break;
        }
    }
}
```

Рисунок 10.38

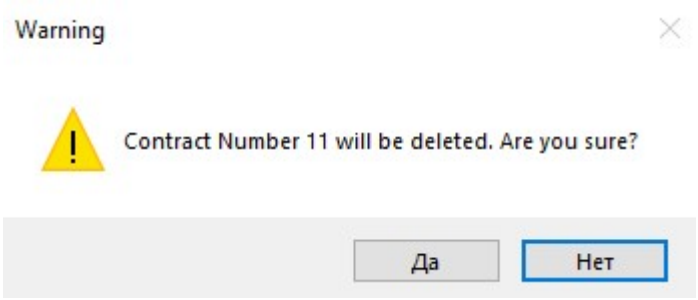



Рисунок 10.39

17. Робота з договорами щодо закупівлі продукції не буде зручною, якщо користувач не матиме можливості працювати з даними про товари, закуплені на підставі договорів. У зв'язку з цим функціональність розробленої форми потрібно розширити, забезпечивши користувачеві можливість працювати з даними про закуплені товари. Насамперед необхідно збільшити вертикальний розмір форми. Після цього під об'єктом tabControl1 потрібно розмістити приблизно такий самий (за розміром) об'єкт TabControl. Для цього об'єкта встановити ім'я tabControl2. Об'єкт повинен містити дві вкладки з іменами tabPage3 та tabPage4. Для властивості Text цих вкладок встановити значення List of products і Add product відповідно. Перевірити працездатність зміненої форми. Зовнішній вигляд форми наведено на рисунку 10.40.

18. Розмістити на вкладці tabPage3 об'єкт типу DataGridView. Для об'єкта встановити ім'я dataGridView2. Клацнути по кнопці у верхньому правому кутку об'єкта dataGridView2 для того, щоб відкрити вікно DataGridView Tasks (рисунок 10.41). За допомогою цього вікна визначити джерело даних dataGridView1 (рисунок 10.41), вибравши його зі списку джерел даних. При виборі джерела даних необхідно вибрати джерело, яке дозволить зв'язати дані про договори та поставлені на підставі цих договорів товари (рисунок 10.41). Встановити для властивості об'єкта ReadOnly dataGridView2 значення True.

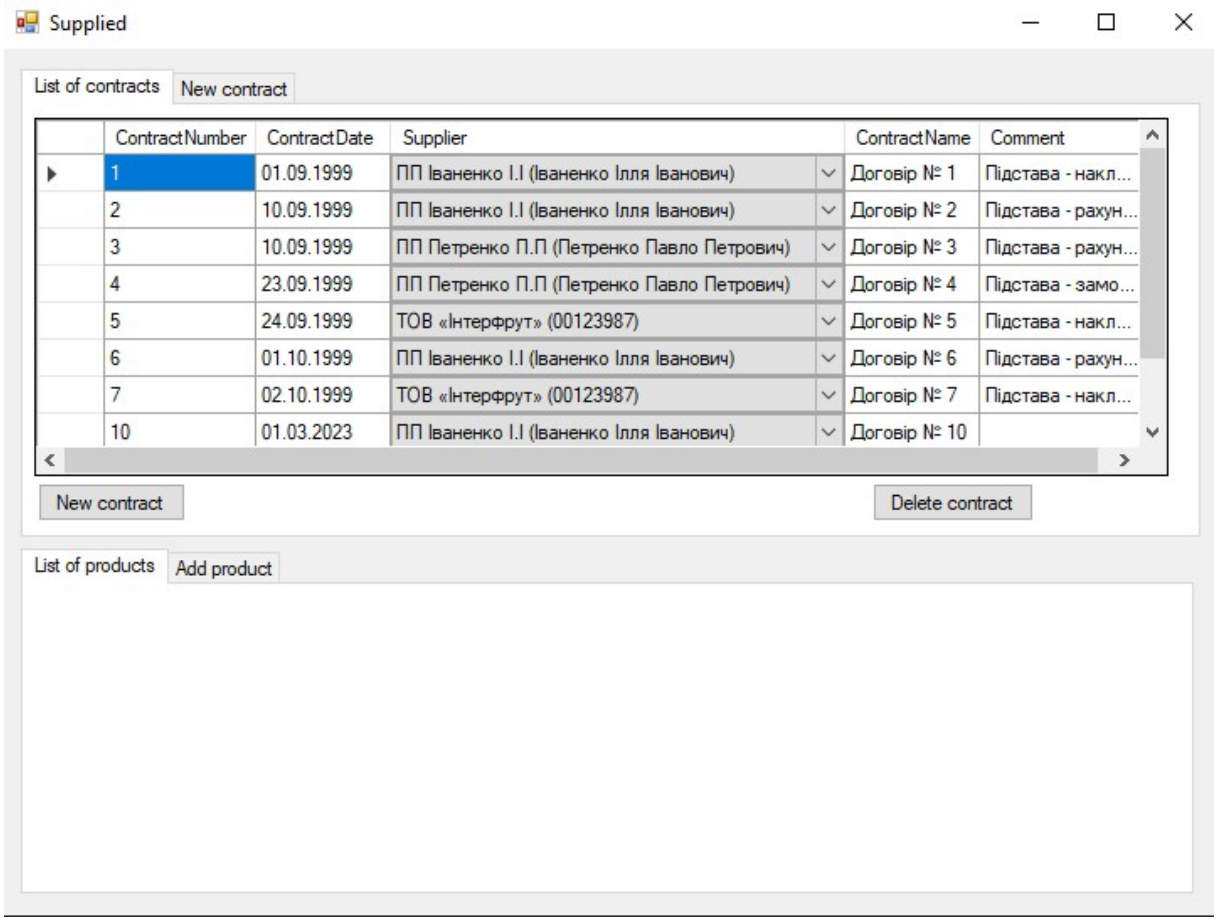



Рисунок 10.40

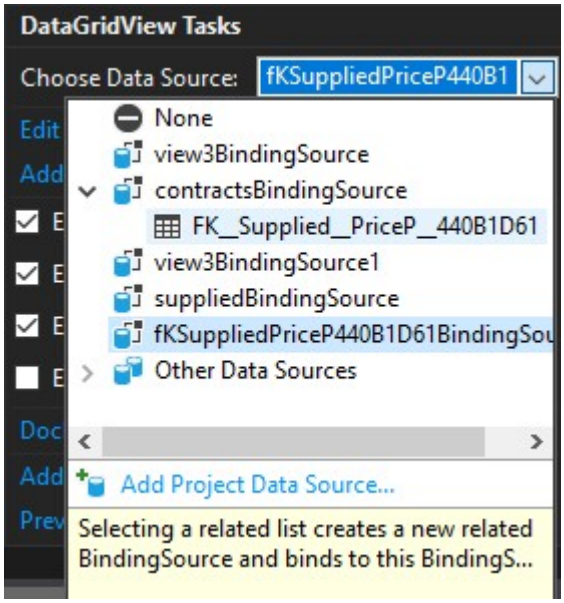



Рисунок 10.41

19. Перевірити працездатність зміненої форми. Зовнішній вигляд форми наведено на рисунку 10.42. Слід звернути увагу, що при виборі договору у списку договорів, у списку товарів має виводитись список товарів, закуплених саме на підставі обраного договору.

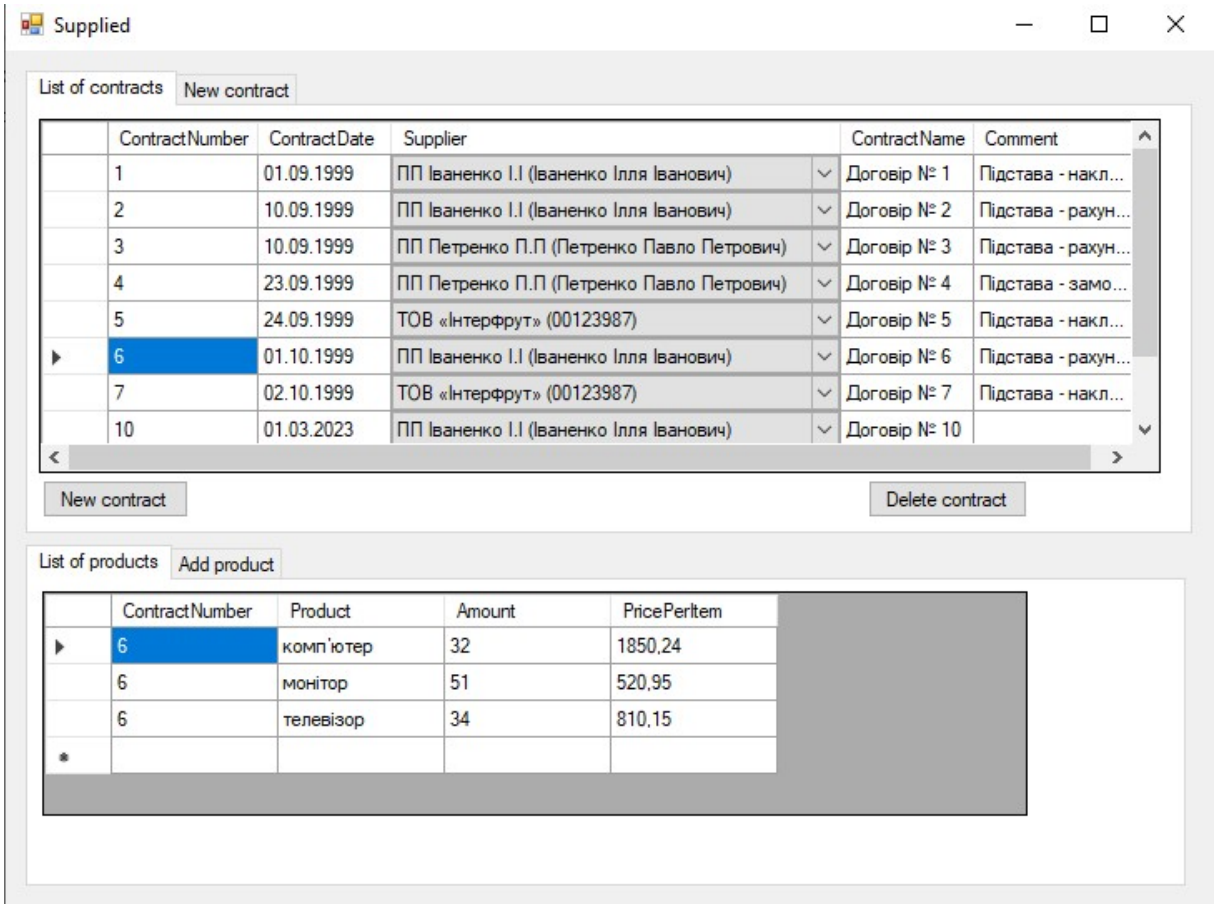



Рисунок 10.42

20. Список товарів у цій формі можна лише переглядати. Зміна списку товарів неможлива. Для того щоб забезпечити можливість користувачеві вводити дані про закуплені товари, внесемо зміни до форми. Для цього необхідно відкрити вкладку Add product (tabPage4). На цій вкладці розміщено три об'єкти типу Label, за допомогою яких формуються коментарі для полів введення даних (рисунок 10.43). Для введення назви товару, кількості одиниць та ціни за одиницю необхідно використовувати об'єкти типу TextBox. Для цих об'єктів встановлюються імена TextBox3,

TextBox4, TextBox5 відповідно. Також на вкладці потрібно розмістити два об'єкти типу Button. Для властивості Text цих кнопок потрібно встановити значення Save та Cancel. Для імен цих об'єктів встановити значення button5 та button6 відповідно.

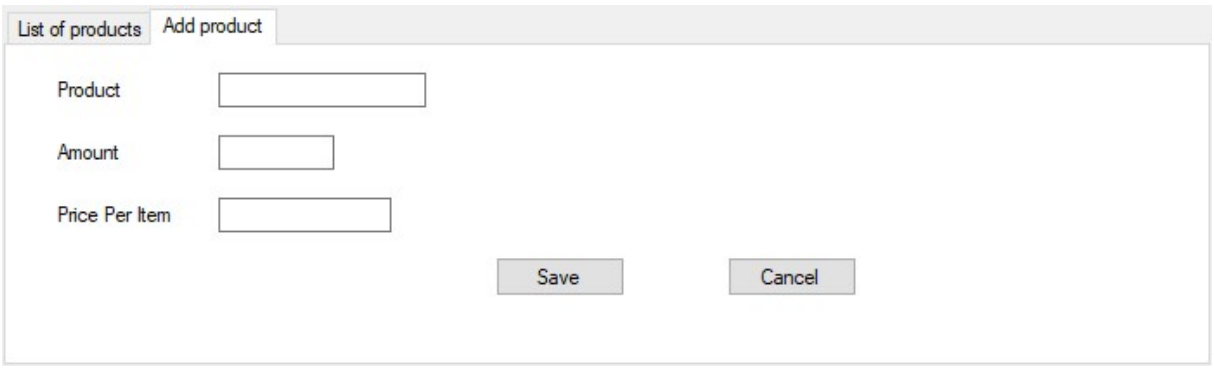



Рисунок 10.43

21. Для об'єкта button5 створити функцію – обробник події Click. Текст функції наведено на рисунку 10.44.

22. Перевірити працездатність форми. Для цього додати новий товар до будь-якого договору. Припустимо, що товар додається до договору 7. Дані нового товару наведено на рисунку 10.45. Після натискання кнопки Save на екран має бути виведене повідомлення, що підтверджує успішне введення даних, потім вкладку Add product буде закрито та відкрито вкладку List of products. Новий товар має бути присутнім у списку (рисунок 10.46).

23. Закрити застосунок і повернутися в режим Design для Form5.cs. На вкладці Add product для об'єкта button6 (кнопка Cancel) створити функцію – обробник події Click. Текст функції наведено на рисунку 10.47. Перевірити роботу цієї кнопки (при її натисканні закривається вкладка Add product та відкривається вкладка List of products без додавання нового товару).

```
private void button5_Click(object sender, EventArgs e)
{
    try
    {
        int dgvrNumber = int.Parse(dataGridView1.Rows[dataGridView1.CurrentCell.RowIndex]
            .Cells["ContractNumberDataGridViewTextBoxColumn"].Value.ToString());
        string tovar = Convert.ToString(this.textBox3.Text);
        int tovar_kol = int.Parse(this.textBox4.Text.ToString());
        Decimal tovar_cena = Convert.ToDecimal(this.textBox5.Text);
        conn = new SqlConnection();
        conn.ConnectionString = "integrated security=SSPI;data source=\".\"; " +
                                "persist security info=False; initial catalog = delivery";
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandText = "INSERT INTO Supplied (ContractNumber,Product,Amount,PricePerItem) " +
                                    "VALUES (@dgvrNumber,@tovar,@tovar_kol,@tovar_cena)";
        myCommand.Parameters.Add("@dgvrNumber", SqlDbType.Int, 4);
        myCommand.Parameters["@dgvrNumber"].Value = dgvrNumber;
        myCommand.Parameters.Add("@tovar", SqlDbType.NVarChar, 20);
        myCommand.Parameters["@tovar"].Value = tovar;
        myCommand.Parameters.Add("@tovar_kol", SqlDbType.Int, 8);
        myCommand.Parameters["@tovar_kol"].Value = tovar_kol;
        myCommand.Parameters.Add("@tovar_cena", SqlDbType.Decimal, 8);
        myCommand.Parameters["@tovar_cena"].Value = tovar_cena;

        int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
        if (UspeshnoeIzmenenie != 0)
        {
            MessageBox.Show("Changes complited", "Record modified");
        }
        else
        {
            MessageBox.Show("Changes didn't complited", "Record modified");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
    finally
    {
        conn.Close();
    }
    this.suppliedTableAdapter.Fill(this.deliveryDataSet.Supplied);
    this.tabControl2.SelectedTab = tabPage3;
}
```

Рисунок 10.44

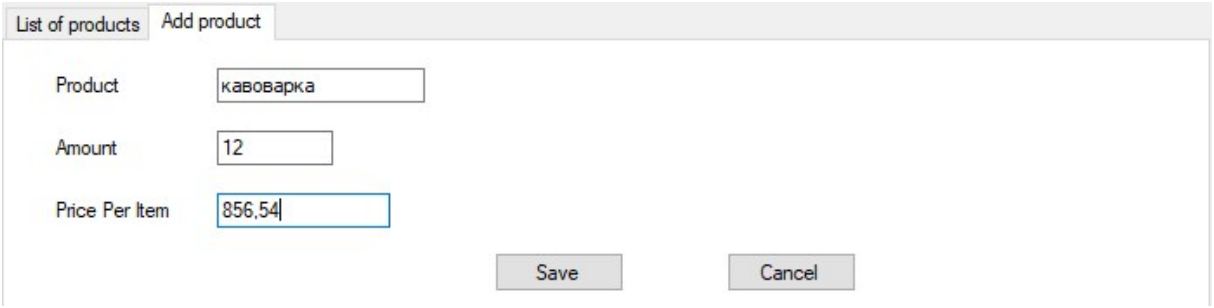



Рисунок 10.45

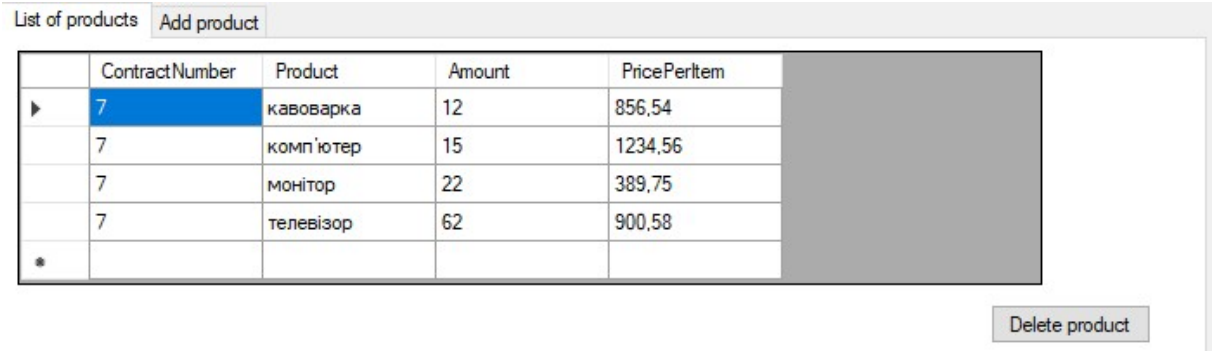



Рисунок 10.46

```
private void button6_Click(object sender, EventArgs e)
{
    this.tabControl2.SelectedTab = tabPage3;
}
```

Рисунок 10.47

24. У процесі роботи з даними про закупівлі продукції може виникнути необхідність видалення раніше введених товарів. Для цього на вкладці List of products створимо відповідну кнопку (рисунок 10.46). Для властивості Text цієї кнопки необхідно встановити значення Delete product. Для імені цього об'єкта встановити значення button7. Для об'єкта button7 створити функцію – обробник події Click. Текст функції наведено на рисунку 10.48.

25. Перевірити роботу кнопки Delete product. Для видалення товару потрібно вибрати товар у списку товарів, поставлених за договором (наприклад, за договором 7) та натиснути кнопку Delete product. В результаті на екрані з'явиться вікно (рисунок 10.49), за допомогою якого потрібно підтвердити видалення товару або відмовитись від цієї операції.

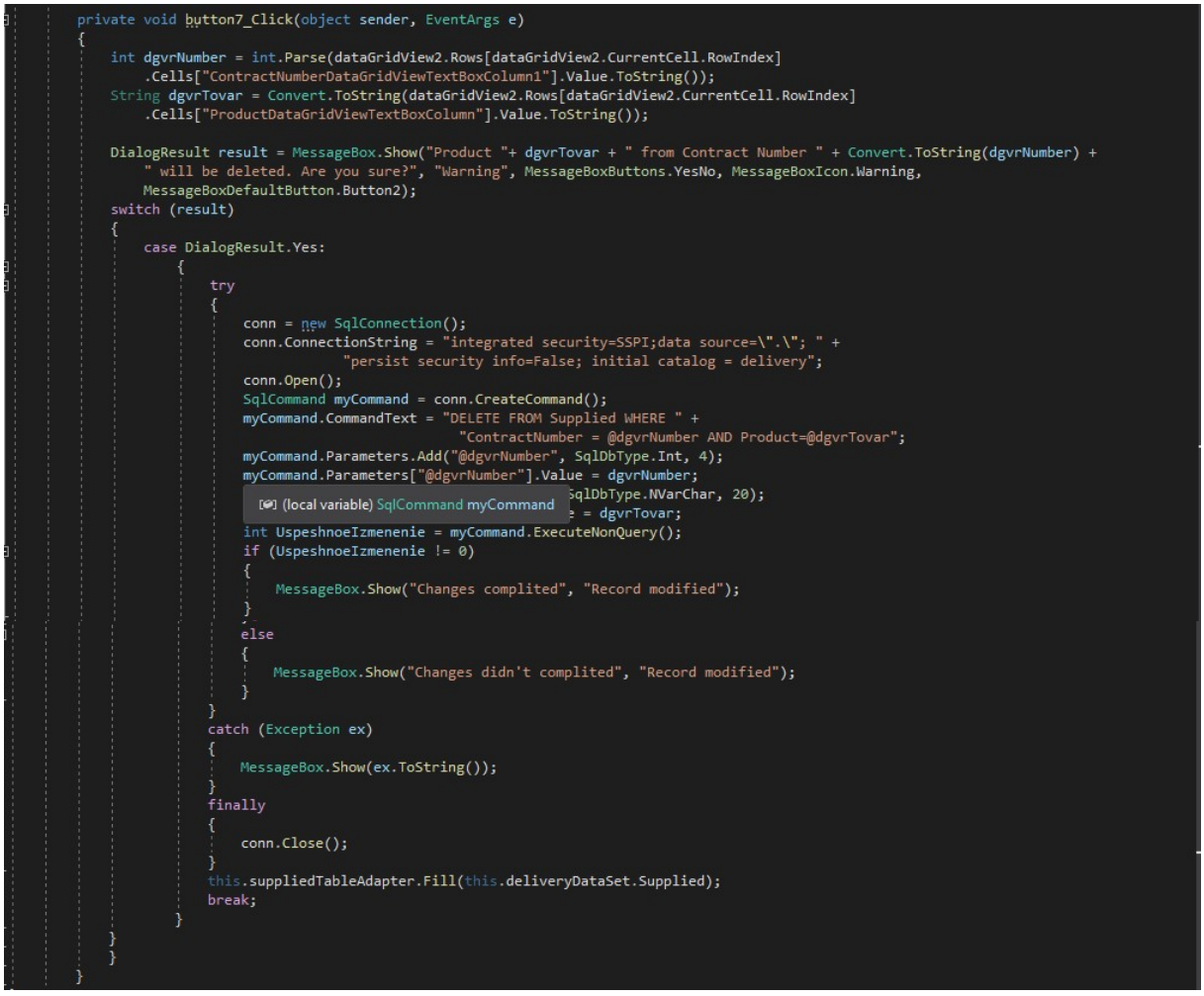



Рисунок 10.48

26. Для підвищення зручності роботи користувача з даними про закупівлі продукції необхідно, щоб користувач, вибираючи конкретний договір, бачив не лише список товарів, а й міг би бачити підсумкові дані за договором – загальну кількість одиниць закуплених товарів та загальну суму закупки. Для формування цих даних можна використовувати збережену процедуру. Таку збережену процедуру вже було створено при виконанні попередніх лабораторних робіт, але, про всяк випадок, текст запиту, за допомогою якого буде створена така збережена процедура, наведено на рисунку 10.50.

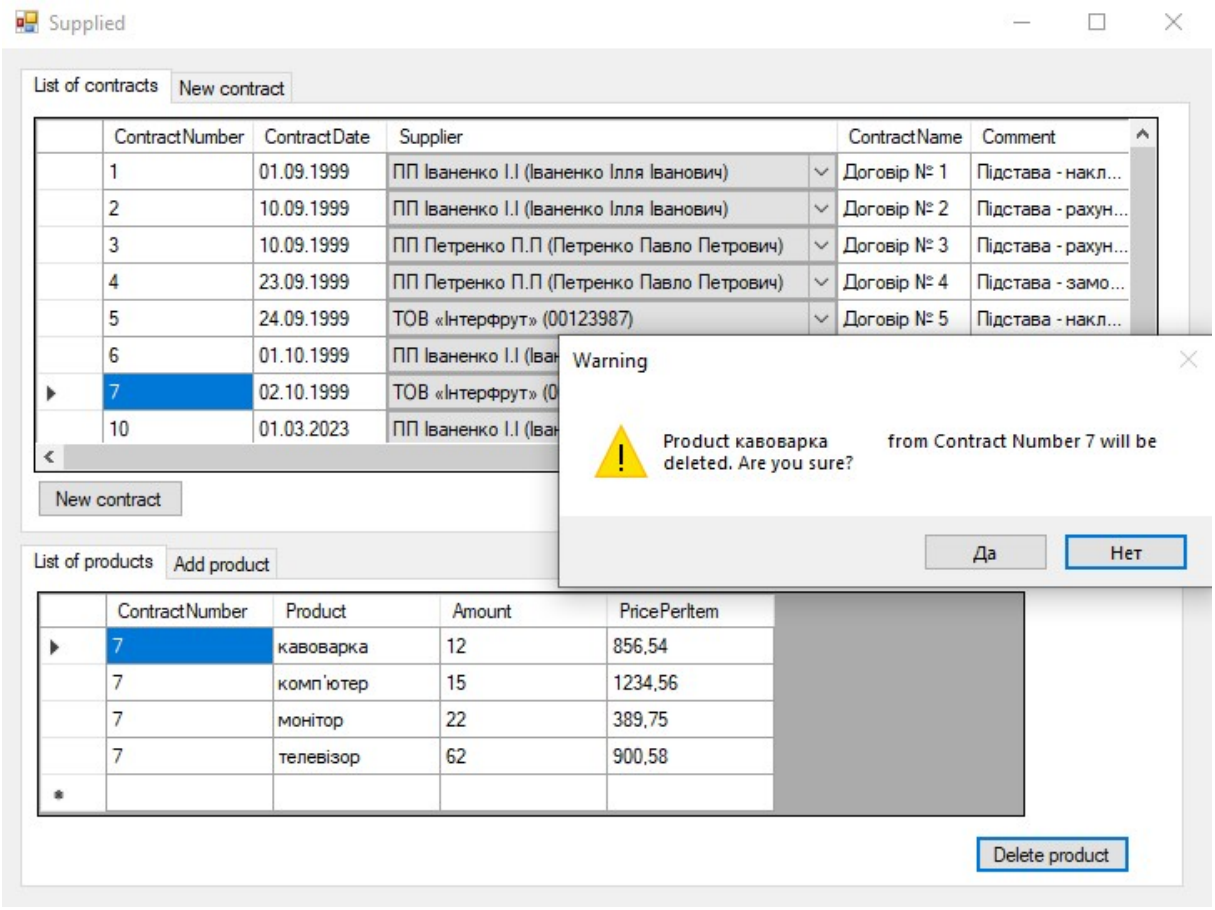



Рисунок 10.49

```
CREATE PROCEDURE [dbo].[sp_sum_dgvr]
    @dgvrNumber int,
    @sum_kol_vo int output,
    @sum_summa decimal(8,2) out
AS
BEGIN
    SET NOCOUNT ON
    SELECT @sum_kol_vo=SUM(Amount),
           @sum_summa= SUM(Amount*Pric
        FROM Supplied
        WHERE ContractNumber=@dgvrNum
```

Рисунок 10.50

27. На вкладці List of products (tabPage3) розмістити два об'єкти типу Label та два об'єкти типу TextBox. Для об'єкта типу Label для властивості Text встановити значення Total Amount та Total Cost (рисунок 10.51). Для об'єктів типу TextBox встановити імена textBox6 та textBox7 відповідно.

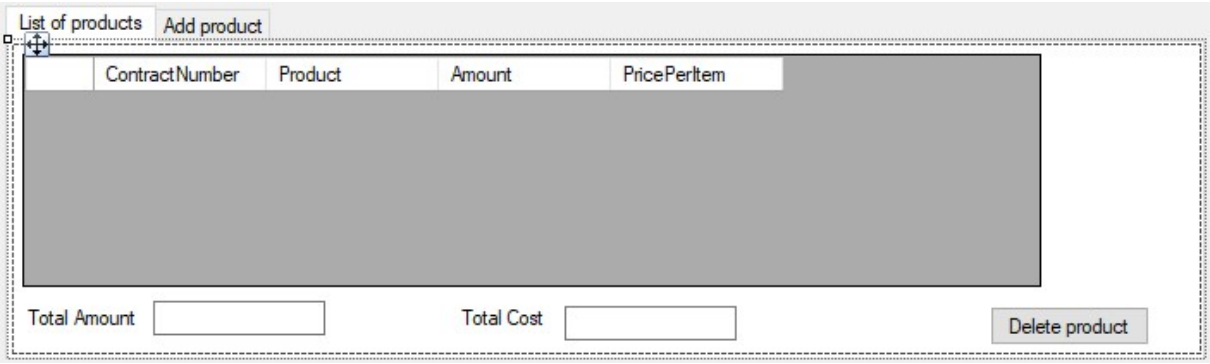



Рисунок 10.51

28. Для об'єкта dataGridView1 створити функцію – обробник події CellClick. Текст функції наведено на рисунку 10.52. За допомогою вікна Properties перевірити, що подія та функція-обробник справді пов'язані (рисунок 10.53).

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        int dgvrNumber = int.Parse(dataGridView1.Rows[dataGridView1.CurrentCell.RowIndex]
           .Cells["ContractNumberDataGridViewTextBoxColumn"].Value.ToString());
        conn = new SqlConnection();
        conn.ConnectionString = "integrated security=SSPI;data source=\".\"; " +
                                "persist security info=False; initial catalog = delivery";
        SqlCommand myCommand = conn.CreateCommand();
        myCommand.CommandType = CommandType.StoredProcedure;
        myCommand.CommandText = "[sp_sum_dgvr]";
        myCommand.Parameters.Add("@dgvrNumber", SqlDbType.Int, 4);
        myCommand.Parameters["@dgvrNumber"].Value = dgvrNumber;
        myCommand.Parameters["@dgvrNumber"].Direction = ParameterDirection.Input;
        myCommand.Parameters.Add("@sum_kol_vo", SqlDbType.Int, 4);
        myCommand.Parameters["@sum_kol_vo"].Direction = ParameterDirection.Output;
        myCommand.Parameters.Add("@sum_summa", SqlDbType.Money);
        myCommand.Parameters["@sum_summa"].Direction = ParameterDirection.Output;
        conn.Open();
        myCommand.ExecuteScalar();
        textBox6.Text = Convert.ToString(myCommand.Parameters["@sum_kol_vo"].Value);
        textBox7.Text = Convert.ToString(myCommand.Parameters["@sum_summa"].Value);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
    finally
    {
        conn.Close();
    }
}
```

Рисунок 10.52

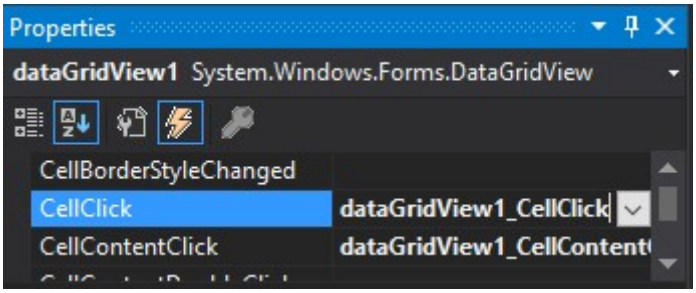



Рисунок 10.53

29. Перевірити працездатність форми. В результаті при виборі договору у списку договорів повинні розраховуватися та виводитися на екран підсумкові дані щодо договору (рисунок 10.54). Закрити застосунок та повернутися в режим Design для Form5.cs. Закрити форму Form5.cs.

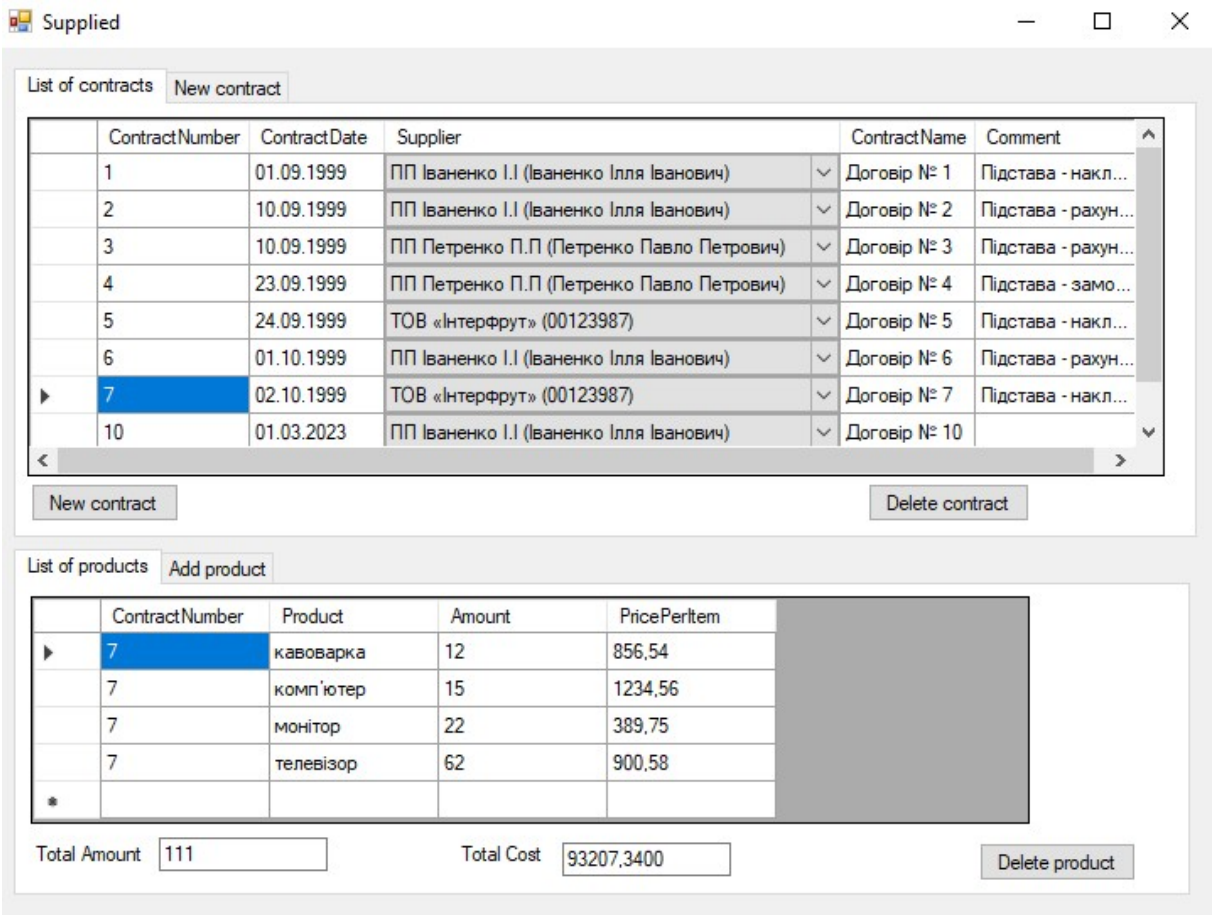



Рисунок 10.54

### 10.2.5 Створення звітної форми з узагальненими даними

Розглянемо процес створення форми, за допомогою якої кінцевий користувач буде мати можливість переглядати узагальнені дані про договори закупівлі.

Припустимо, що кінцевим користувачам треба бачити список договорів (із зазначенням номера, дати поставки та даних про постачальника), загальну кількість поставлених товарів та загальну суму за кожним договором. При формуванні даних постачальника для усіх постачальників треба вивести контактні дані, для постачальників-фізичних осіб вивести прізвище та ініціали, для постачальників-юридичних осіб – податковий номер. При цьому прізвище та ініціали або податковий номер повинні бути виведені в одному полі, тобто, або прізвище та ініціали, або податковий номер. У результат такого звіту мають бути включені лише ті договори, на підставі яких товари дійсно поставлялися (тобто результат запиту не повинен потрапити так звані «порожні» договори).

Щоб отримати такі дані, створимо у базі даних представлення. Текст визначального запиту такого представлення може мати вигляд, наведений на рисунку 10.55. Нескладно помітити, що подібний запит вже було розроблено при виконанні попередніх лабораторних робіт. Отже, можна використати текст вже існуючого запиту. Таке представлення необхідно зробити за допомогою SQL Server Management Studio та зберегти з ім'ям View_4. Потім у вікні Data Sources потрібно клацнути правою кнопкою миші по джерелу даних deliveryDataSet і в меню вибрати пункт Configure DataSet with Wizard ... У вікні потрібно повторно відзначити пункт Views. В результаті новостворене представлення має з'явитися у списку об'єктів джерела даних. Змінене джерело даних треба зберегти.

```
SELECT Contracts.ContractNumber, ContractDate, SUM(Amount) AS кількість, SUM(Amount*PricePerItem) AS сума,
     CAST(Note AS CHAR(40)) AS КонтактиПостачальника,
     ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
          AS Постачальник
FROM ((Suppliers LEFT JOIN IndividualEntrepreneurs ON Suppliers.SupplierID=IndividualEntrepreneurs.SupplierID)
     LEFT JOIN LegalEntities ON Suppliers.SupplierID=LegalEntities.SupplierID)
          INNER JOIN Contracts ON Suppliers.SupplierID=Contracts.SupplierID
     INNER JOIN Supplied ON Supplied.ContractNumber=Contracts.ContractNumber
GROUP BY Contracts.ContractNumber, ContractDate, CAST(Note AS CHAR(40)),
ISNULL(LegalEntities.TaxNumber, RTRIM(LastName)+' '+SUBSTRING(FirstName,1,1)+'.'++SUBSTRING(SecondName,1,1)+'.')
```

Рисунок 10.55

Для підключення такого представлення до застосунку треба виконати наступну послідовність дій.

1. Створити нову форму, наприклад, з ім'ям Form6.cs та додати її до проекту. Для властивості Text форми встановити значення Aggregate data of contracts.

2. В результаті до проекту буде додано нову форму. Вибравши у списку об'єктів джерела даних представлення View_4, потрібно за допомогою миші перетягнути її на форму. В результаті на формі з'являться об'єкти типу DataGridView та BindingNavigator (рисунок 10.56), за допомогою яких можна переглянути дані які будуть сформовані за допомогою представлення. Форму Form6.cs можна зберегти та закрити.

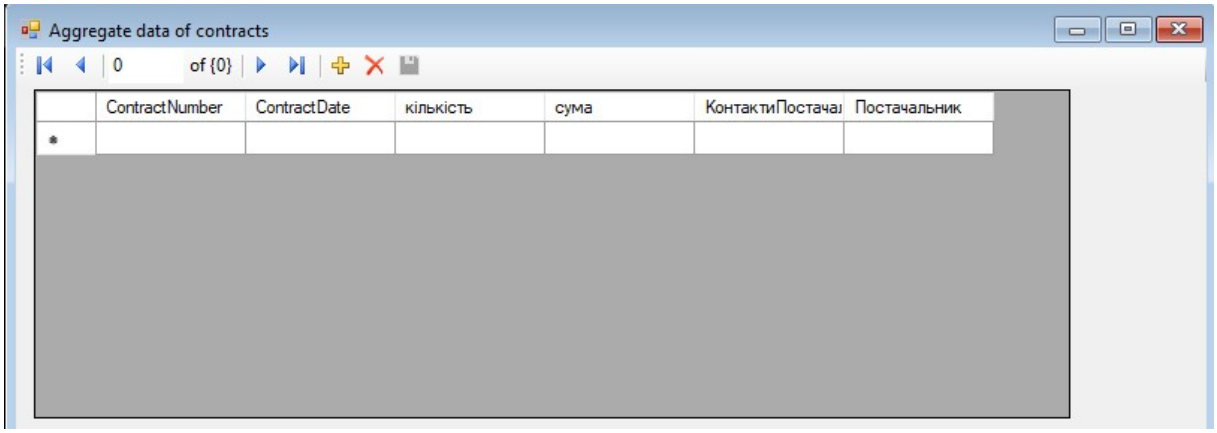



Рисунок 10.56

3. Тепер створену форму потрібно підключити до спільного меню програми. Для цього потрібно перейти на роботу з головною формою

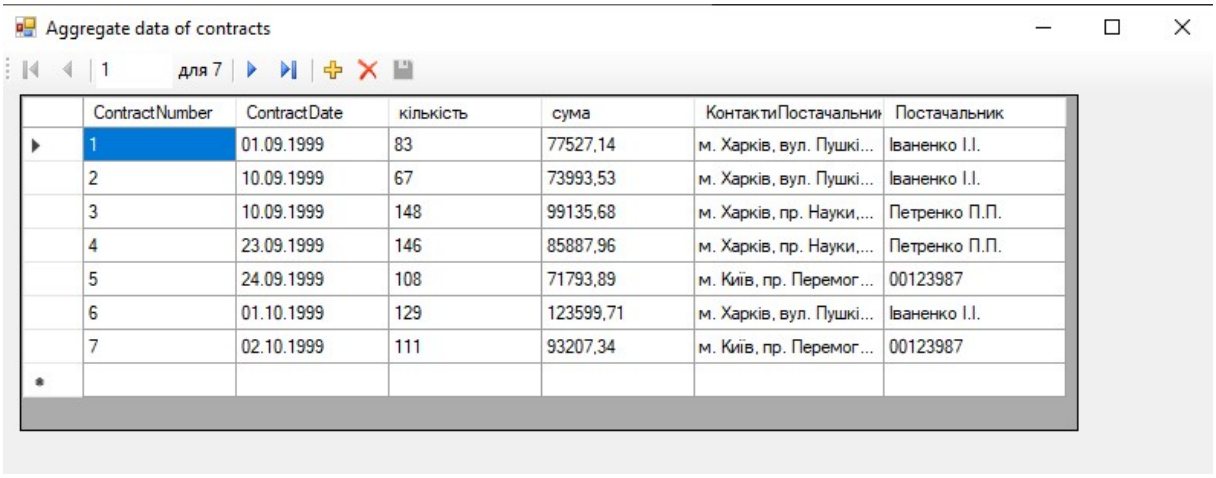
(Form1) (в режимі Design) і для пункту головного меню Reports створити підменю. Перший пункт цього підменю слід назвати Aggregate data of contracts (рисунок 10.57). Подвійне натискання миші на цьому пункті меню відкриє вікно коду, в якому можна ввести текст функції, що забезпечує відкриття форми при виборі цього пункту меню (рисунок 10.58).


Рисунок 10.57

```
private void toolStripMenuItem2_Click(object sender, EventArgs e)
{
    Form6 frm = new Form6();
    frm.Show();
}
```

Рисунок 10.58

4. Перевірити працездатність форми. В результаті при виборі відповідного пункту меню кінцевий користувач буде мати можливість переглядати узагальнені дані про договори закупівлі (рисунок 10.59).




| | ContractNumber | ContractDate | кількість | сума | КонтактиПостачальни | Постачальник |
|---|---|---|---|---|---|---|
| ▶ | 1 | 01.09.1999 | 83 | 77527,14 | м. Харків, вул. Пушкі... | Іваненко І.І. |
| | 2 | 10.09.1999 | 67 | 73993,53 | м. Харків, вул. Пушкі... | Іваненко І.І. |
| | 3 | 10.09.1999 | 148 | 99135,68 | м. Харків, пр. Науки,... | Петренко П.П. |
| | 4 | 23.09.1999 | 146 | 85887,96 | м. Харків, пр. Науки,... | Петренко П.П. |
| | 5 | 24.09.1999 | 108 | 71793,89 | м. Київ, пр. Перемог... | 00123987 |
| | 6 | 01.10.1999 | 129 | 123599,71 | м. Харків, вул. Пушкі... | Іваненко І.І. |
| | 7 | 02.10.1999 | 111 | 93207,34 | м. Київ, пр. Перемог... | 00123987 |
| * | | | | | | |

Рисунок 10.59

Зберегти папку проєкту і всі файли, що знаходяться в ній. Назву папки (наприклад, \WindowsFormsApplication1) слід подивитися на своєму робочому комп'ютері.

### 10.2.6 Звітність про виконання практичних завдань теми 10

Відповіднім чином оформлений та роздрукований звіт про виконання практичних завдань теми є документом, що підтверджує виконання студентом відповідної теми.

У звіті треба:

1) стисло описати основні етапи виконання завдання;

2) описати створений клієнтський застосунок та особливості роботи з ним;

3) зробити висновки за результатами виконання практичних завдань теми.

Звіт роздруковується на аркушах формату А4, він повинен мати відповідній титульний аркуш. Роздрукованій звіт здається студентом викладачу у файлі.

Звіт має буті оформлень за такими вимогами:

– параметри сторінки: лівий відступ – 3 см; правий – 1,5 см; верхній та нижній відступи по 2 см;

– шрифт Times New Roman, 14;

– налаштування абзацу: вирівнювання – за шириною, відступи зліва та справа – 0 см, відступ першого рядка – 1,25 см, інтервал перед та після абзацу – 0 пт, міжрядковій інтервал – одинарний; на вкладці «Положення на сторінці» відключити функцію «Заборона висячих рядків».

Усі скріншоти розміщені у звіті, є рисунками, отже повинні мати підписи та відповідну нумерацію.

В цілому оформлення звіту повинно відповідати стандарту НТУ «ХПІ» «ТЕКСТОВІ ДОКУМЕНТИ У СФЕРІ НАВЧАЛЬНОГО ПРОЦЕСУ» (https://blogs.kpi.kharkov.ua/v2/metodotdel/wp-content/uploads/sites/28/2025/06/STZVO-HPI-3.01-2025-2.pdf).

Ознакою того, що студент не тільки виконав практичні завдання, але й здав їх (тобто підтвердив наявність відповідних знань та навичок), є наявність на титульному аркуші підпису викладача та дати здачі.

Увага! При проведенні занять у дистанційній формі звіти надаються в електронному вигляді за допомогою корпоративної електронної пошти. Рекомендований формат файлів звітів – .doc / .docx / .pdf.

### 10.3 Питання для самоперевірки по темі 10

1. Типи користувачів автоматизованої обчислювальної системи.
2. Хто такі кінцеві користувачі?
3. Що таке інтерфейс користувача?
4. Складові інтерфейсу користувача.
5. Що таке графічний інтерфейс користувача?
6. Характеристики графічного інтерфейсу користувача.
7. Принципи побудови «дружнього» інтерфейсу користувача.
8. Правила проектування інтерфейсу користувача. Загальна характеристика.
9. Що значить «дати контроль користувачу»?
10. Що є основними положеннями надання контролю користувачу над інтерфейсом?
11. Що значить «безрежимність»?
12. Що значить «гнучкість»?
13. Що значить «можливість переривання»?
14. Що значить корисність»?
15. Що значить « поблажливість»?
16. Що значить «можливість орієнтування»?
17. Що значить «доступність»?
18. Що значить «спрощення користування»?
19. Що значить «пристосовуваність»?
20. Що значить «інтерактивність»?
21. Що значить «зменшити навантаження на пам'ять користувача»?

22. Що є основними положеннями зменшення навантаження на пам'ять користувача?

23. Що значить «запам'ятовування»?

24. Що значить «розпізнавання»?

25. Що значить «інформування»?

26. Що значить «терпимість»?

27. Що значить «швидкість»?

28. Що значить «інтуїтивність»?

29. Що значить «перенос»?

30. Що значить «контекст»?

31. Що значить «організація»?

32. Що значить «зробити інтерфейс сумісним»?

33. Що є основними положеннями розроблення сумісного інтерфейсу?

34. Що значить «послідовність інтерфейсу»?

35. Що значить «досвід»?

36. Що значить «прогнозування»?

37. Що значить «відношення»?

38. Що значить «передбачуваність»

39. Як у середовищі Microsoft Visual Studio створити проєкт Windows Forms Application?

40. Як додати меню у форму в проєкті Windows Forms Application?

41. Як додати форму в проєкт Windows Forms Application?

42. Як створити нове джерело даних в проєкті Windows Forms Application?

43. Об'єкт DataGridView, його призначення та особливості застосування як складової інтерфейсу в проєкті Windows Forms Application

44. Як за допомогою об'єктів DataGridView відобразити зв'язані дані в формі проєкту Windows Forms Application?

## РЕКОМЕНДОВАНА ЛІТЕРАТУРА


1. Advanced Database Systems. Toronto Academic Press, 2024. 258 p.
2. Beaulieu A. Learning SQL: Generate, Manipulate, and Retrieve Data. O'Reilly Media, Inc., 2020. 384 p.
3. Bush J. Learn SQL Database Programming. Packt Publishing, 2020. 550 p.
4. Carter P. 100 SQL Server Mistakes and How to Avoid Them. Manning Publications Co., 2025. 812 p.
5. Dabadie R. Mastering Visual Studio 2022: Develop apps like a pro with advanced Visual Studio techniques using C# and .NET. Packt Publishing, 2024. 324 p.
6. DAMA International. The DAMA Guide to the Data Management Body of Knowledge (DAMA-DMBOK2). 2nd ed., Technics Publications, 2017. 626 p.
7. Date C. J. Database Design and Relational Theory: Normal Forms and All That Jazz. Apress, 2019. 451 p.
8. DeBarros A. Practical SQL. 2nd Edition. No Starch Press, Inc., 2022. 440 p.
9. Eckstein J., Schultz B. Introductory Relational Database Design for Business, with Microsoft Access. John Wiley & Sons, 2018. 328 p.
10. Garcia M., Rojas H. Hands-On Visual Studio 2022: A developer's guide to exploring new features and best practices in VS2022 for maximum productivity. Packt Publishing, 2022. 350 p.
11. Hoffer J., Ramesh V., Topi H. Modern Database Management. Thirteenth Edition. Pearson Education Limited, 2020. 591 p.
12. Negi M. Fundamentals of Database Management System: Learn essential concepts of database systems. BPB Publications, 2019. 175 p.
13. Özsu M., Valduriez P. Principles of Distributed Database Systems. Springer Nature, 2019. 674 p.
14. Petković D. Microsoft SQL Server 2019. A Beginner's Guide. Seventh Edition. McGraw-Hill Education, 2020. 865 p.

15. Powell G. Database Modeling Step by Step. CRC Press, 2020. 268 p.

16. Purba S. Handbook of Data Management: 1999 Edition. CRC Press, 2019. 1101 p.

17. Salter K. Fundamentals of Database Systems. Toronto Academic Press, 2024. 274 p.

18. Sciore E. Database Design and Implementation: Second Edition. Springer Nature, 2020. 468 p.

19. Simon M. Getting Started with SQL and Databases: Managing and Manipulating Data with SQL. Apress, 2023. 390 p.

20. Stanley A. Visual Studio: A Simplified Guide For Beginners And Experts On Visual Studio. Kindle Paperwhite, 2023. 308 p.

21. Strauss D. Getting Started with Visual Studio 2022: Learning and Implementing New Features. Kindle Paperwhite, 2022. 332 p.

22. Taylor A. SQL All-in-One For Dummies, 4th Edition. John Wiley & Sons, Inc., 2024. 803 p.

23. The Creation and Management of Database Systems. Arcler Press, 2023, 262 p.

24. Yablonski J. Laws of UX: Using Psychology to Design Better Products & Services. O'Reilly, 2020. 150 p.

# СТИСЛІ ВІДОМОСТІ ПРО АВТОРІВ

КОПП Андрій Михайлович
Andrii.Kopp@khpi.edu.ua
Доктор філософії (PhD) з комп'ютерних наук,
доцент,
завідувач кафедри програмної інженерії та
інтелектуальних технологій управління
ім. А.В. Дабагяна НТУ «ХПІ»

ОРЛОВСЬКИЙ Дмитро Леонідович
Dmytro.Orlovskyi@khpi.edu.ua
канд. техн. наук, доцент,
професор кафедри програмної інженерії та
інтелектуальних технологій управління
ім. А.В. Дабагяна НТУ «ХПІ»

Навчальне видання

КОПП Андрій Михайлович
ОРЛОВСЬКИЙ Дмитро Леонідович

# ОСНОВИ РОБОТИ З БАЗАМИ ДАНИХ ІЗ ВИКОРИСТАННЯМ СУБД MICROSOFT SQL SERVER

Навчально-методичний посібник
для студентів спеціальностей
F2 «Інженерія програмного забезпечення» та F3 «Комп'ютерні науки»

Відповідальний за випуск доц. Копп А. М.
Роботу до видання рекомендував проф. Гамаюн І. П.

В авторській редакції

План 2025 р., поз. 109

Гарнітура Times New Roman. Ум. друк. арк. 15,8

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2.

Електронне видання