МЕТОДИЧНІ ВКАЗІВКИ

для виконання лабораторних робіт

з дисципліни

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Частина 1

2016

Методичні вказівки для виконання лабораторних робіт з дисципліни «Організація баз даних та знань. Частина 1» призначений для студентів III курсу денної форми навчання, що навчаються за спеціальністю 122 «Комп'ютерні науки», рівень вищої освіти – бакалавр.

Укладач – к.ф.-м.н., доцент кафедри інформаційних технологій, Козловська Валентина Петрівна.

ЗМІСТ

Передмова

Курс “Організація баз даних та знань” – це складова частина державного стандарту вищої освіти на рівні бакалавра.

Метою дисципліни є формування у студентів теоретичних знань та практичних навичок, необхідних для роботи у будь-яких адміністративних, наукових та виробничних підрозділах, в яких використовують бази даних, а також здійснюють обслуговування та розробку інформаційних систем та систем баз даних.

Згідно з поставленою метою до задач дисципліни входить:

- Ознайомлення студентів з ґрунтовними поняттями теорії баз даних, з найбільш розповсюдженими системами керування базами даних (СКБД);
- Надання студентам навичок проектування та створення баз даних; обробки та оновлення даних у базі;
- Надання понять захисту та цілісності даних та навичок забезпечення цілісності даних в розробленій базі даних;
- Створення та відлагодження прикладних програмних для обробки баз даних;
- Ознайомлення студентів з ґрунтовними поняттями теорії баз знань.

Після освоєння цієї дисципліни студент **повинен знати**: призначення СКБД; стандарти СКБД; моделі даних, переваги, що надає реляційна модель; поняття теорії нормалізації бази даних; поняття захисту даних та цілісності даних, також методи забезпечення захисту та цілісності даних; поняття одночасної роботи з базою даних, проблеми, які при цьому виникають, та методи вирішення цих проблем; ґрунтовні понятті теорії баз знань, такі, як: складові частини системи управління базами знань, розподіл знань на алгоритмічні, концептуальні, фактуальні.

За результатами навчання студент **повинен вміти**: створювати реляційні бази даних у середовищі СКБД MS SQL Server; створювати програми обробки бази даних; створювати запити мовою SQL; маніпулювати даними в реляційних СКБД.

Ця дисципліна базується на знаннях та навичках програмування, отриманих під час вивчення дисциплін “Алгоритмізація та програмування” та

“Об’єктно-орієнтоване програмування” та забезпечує засвоєння декількох дисциплін з циклу професійно-орієнтованих дисциплін.

У процесі самостійного вивчення курсу студент повинен керуватися його програмою і вивчити за конспектом лекцій та літературою, що рекомендована викладачем, відповідний теоретичний матеріал.

Лабораторна робота №1 Створення бази даних

Мета роботи:

1. Знайомство з програмою Query Analyzer MS SQL Server;
2. Отримання практичних навичок створення бази даних мовою SQL у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

1. Синтаксис команди CREATE TABLE.
2. Типи даних MS SQL Server.

Теоретичні відомості. Реляційна СКБД MS SQL Server

СКБД клієнт-серверної архітектури

Перші системи керування базами даних (СКБД) були створені за архітектурою локальних СКБД, тобто СКБД обслуговувала тільки той комп'ютер, на якому сама розташовувалася. До локальних СКБД архітектури з файловим сервером належить, наприклад, СКБД Visual FoxPro. Архітектура локальних СКБД має суттєві недоліки.

З метою усунення недоліків цих підходів була розроблена технологія "клієнт/сервер". У цій технології використовується принцип взаємодії програмних компонентів, при якому вони утворюють єдину систему. Як видно з назви, існує якийсь *клієнтський процес*, що вимагає певних ресурсів, і *серверний процес*, що надає ці ресурси. Ці процеси можуть бути на різних комп'ютерах, з'єднаних у мережу різної топології. Комп'ютер, що забезпечує серверний процес, називається *сервером бази даних*. Звичайно на цьому комп'ютері перебуває база даних. Також база даних може перебувати на іншому комп'ютері, до якого прямо має доступ тільки сервер.

На сервері перебуває та частина ПЗ бази даних, що виконує функції забезпечення одночасної роботи, цілісності бази даних і її захисту. Цю частину програмного забезпечення називає *серверним ПЗ*, а часто просто *сервером*. Інша частина програмного забезпечення роботи з базою даних, називається *клієнтським ПЗ*. Вона виконує, в основному, наступні функції:

- керує інтерфейсом користувача;
- приймає й перевіряє синтаксис введеного користувачем запиту;
- виконує додаток;
- відображає отримані дані користувачеві.

Цей тип архітектури має перелічені нижче переваги.

- Забезпечується більш широкий доступ до існуючих баз даних.
- Підвищується загальна продуктивність системи. Оскільки клієнти й сервер знаходяться на різних комп'ютерах, їхні процесори здатні виконувати додатки паралельно.

- Вартість апаратного забезпечення зменшується. Досить потужний комп'ютер з великим пристроєм зберігання потрібен тільки серверу – для зберігання даних та керування базою даних.
- Скорочуються комунікаційні витрати. Додатки виконують частину операцій на клієнтських комп'ютерах і посилають через мережу тільки запити до бази даних, що дозволяє істотно скоротити обсяг даних, що пересилають по мережі.
- Підвищується рівень несуперечності даних. Сервер може самостійно керувати перевіркою цілісності даних, оскільки всі обмеження визначаються й перевіряються в одному місці.

До найбільш відомих СКБД з архітектурою клієнт/сервер відносяться наступні СКБД: MS SQL Server, MySQL, Oracle, Firebird. Найбільш захищеними СКБД вважаються MS SQL Server та Oracle, але СКБД MySQL та Firebird дозволяють більш високу швидкість обміну даних між сервером та клієнтом. Тому останні СКБД звичайно використовують для розробки Web-сайтів. Для баз даних з високою ступенем захисту – банківська справа, секретні дані, сайти інтернет-магазинів, що здійснюють банківські розрахунки – використовують MS SQL Server та Oracle.

Іменування об’єктів в СКБД MS SQL Server 2000

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

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

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

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

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

SET QUOTED_IDENTIFIER {ON | OFF}

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

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

SET QUOTED_IDENTIFIER ON

або

SET QUOTED_IDENTIFIER OFF

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

Доступ до об'єктів

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

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

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

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

Як видно з синтаксису команди, обов'язковою частиною повного імені об'єкта є тільки власне ім'я об'єкта, інші частини можуть бути відсутніми. Але

можуть бути присутніми деякі з необов'язкових частин імені. Наприклад, на одному сервері є декілька баз даних, що належать одному користувачеві. Користувач працює на цьому сервері з таблицями з декількох своїх баз даних. Тоді імена таблиць з різних баз даних будуть записуватись наступним чином:

`Database1..TableName1` або `Database2..TableName2`

Робота з програмою Query Analyzer MS SQL Server 2000

Для роботи з базою даних потрібно завантажити програму Query Analyzer MS SQL Server 2000. Після завантаження відкриється вікно програми (рис.1).

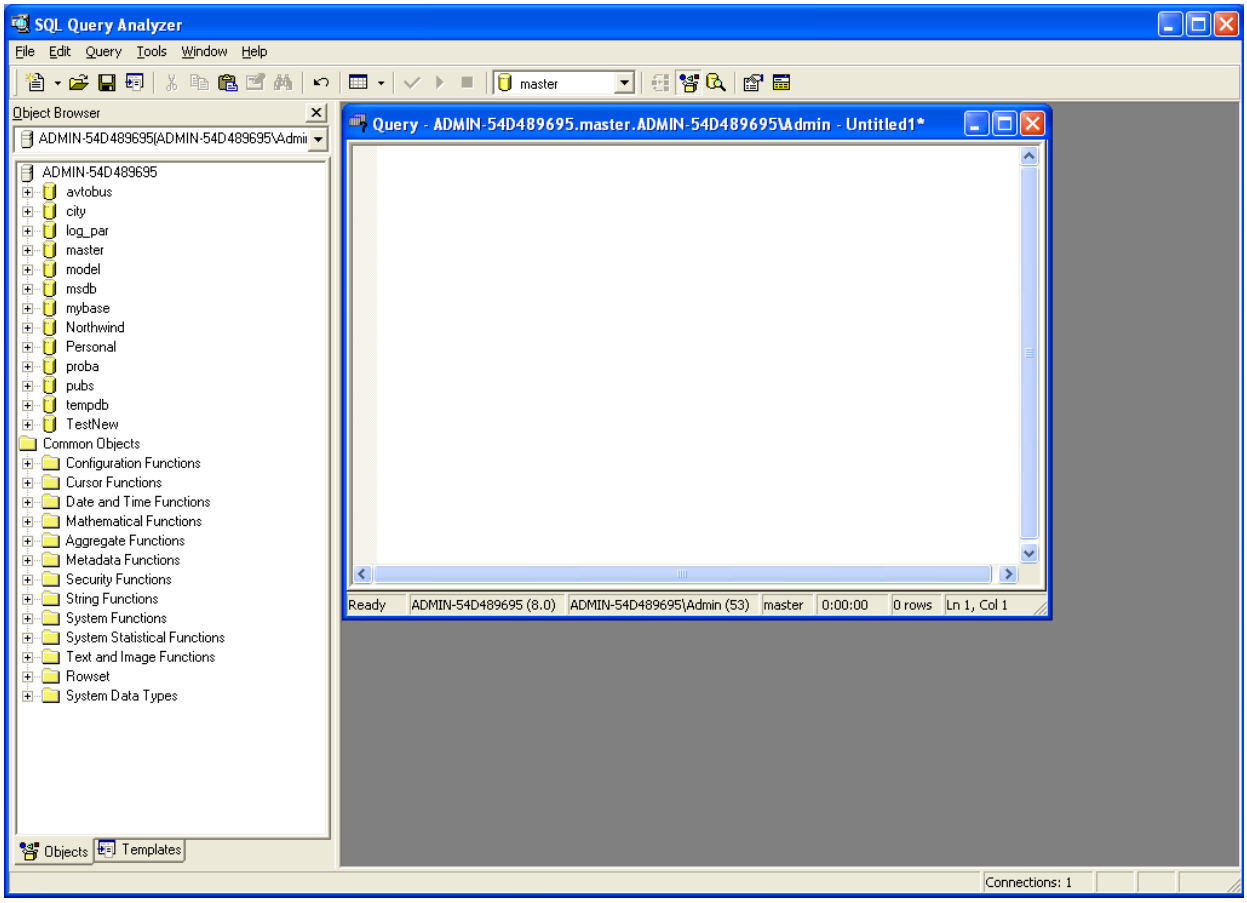


Рисунок 1 – Вигляд вікна програми MS SQL Server 2000

В лівій частині вікна знаходиться поле **Object Browser**, в якому перелічені стандартні об'єкти програми (**Common Objects)**: типи даних, функції різних типів, – та доступні для користувача бази даних (вище **Common Objects**). Праворуч на темному фоні розташоване вікно для вводу команд SQL-коду з назвою "Query - …".

Якщо поля **Object Browser** немає у вікні програми, то викликати його можна за допомогою пунктів головного меню **Tools/Object Browser/"Show/Hide"**, або клавішею F8.

В полі **Object Browser** зручно переглядати вміст бази даних. Якщо клацнути по значку “+” біля імені бази даних, випадає перелік об’єктів бази даних: User Tables (таблиці користувача), System Tables (системні таблиці), Views (представлення), Stored Procedures (збережені процедури), Functions (функції, визначені користувачем), User Defined Data Types (визначені користувачем типи даних). Якщо клацнути по значку “+” біля слів “User Tables”, то з’явиться перелік усіх таблиць користувача в базі даних. Для перегляду структури таблиці: назв атрибутів та їх типів – треба клацнути по значку “+” біля імені таблиці (рис.2).

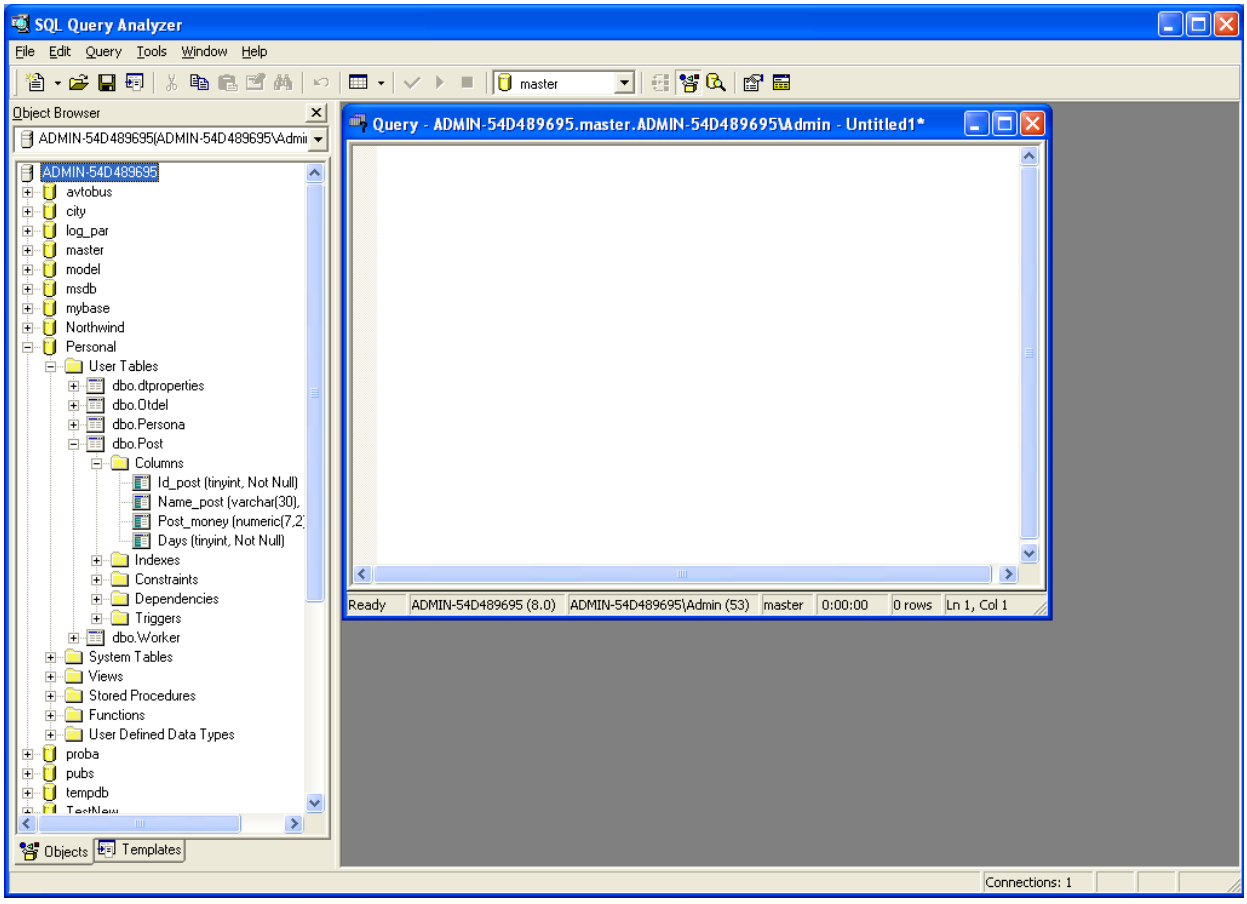


Рисунок 2 – Перегляд об’єктів БД у **Object Browser**

Написаний у вікні “Query…” SQL-код можна виконувати або весь, або виділений фрагмент коду. Для виконання коду потрібно натиснути клавішу F5, або клацнути мишкою по кнопці з трикутником на панелі інструментів:  (кнопка Execute Query). Відповідь SQL Server на введену команду (команди) з’являється унизу вікна “Query…”. Написаний код можна зберігати у файлах з розширенням .sql (текстові файли), і потім використовувати.

Якщо код має синтаксичну помилку, то він не буде виконуватись, і SQL Server надасть повідомлення про помилку. Але помилка може бути *логічною*, тобто написана команда є правильною командою мови Transact-SQL, але вона виконує не ті дії, які потрібні користувачу. Якщо така команда вже виконана,

але результат визиває підозри, потрібно уважно переглянути код команди, та усунути помилку в коді.

Створення БД в MS SQL Server 2000

Створення бази даних

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

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

Зауваження: Правом створення баз даних, точніше виконанням команди CREATE DATABASE, по умовчанню володіють тільки члени фіксованих ролей сервера sysadmin і dbcreator. Проте, право створення бази даних може бути надано не тільки включенням облікового запису в одну із зазначених ролей сервера, але й шляхом надання обліковому запису права CREATE DATABASE. Надати обліковому запису право CREATE DATABASE можуть тільки члени фіксованої ролі сервера sysadmin або securityadmin.

Розглянемо призначення основних параметрів команди:

database_name

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

ON

Присутність цього ключового слова свідчить про те, що далі треба визначення файлів даних, а також груп файлів, з яких буде складатися база даних. Якщо параметр ON опускається, то для бази даних буде створений єдиний файл даних. За замовчуванням розмір цього файлу дорівнює 768

Кбайт (0,75 Мбайт). Однак цей розмір може бути змінений шляхом внесення відповідних змін у базу даних `Model`.

`LOG ON`

Після цього ключового слова вказується набір файлів, з яких буде складатися журнал транзакцій бази даних. Якщо приводиться більше одного файлу, то вони повинні бути розділені комою. Як видно із синтаксису, параметр LOG ON може не вказуватися. У цьому випадку журнал транзакцій буде складатися з єдиного файлу, розмір якого буде становити 25% від загального об'єму всіх файлів даних, але не менш 512 Кбайт, і буде мати ім'я, що генерується на основі імені бази даних, до якого додається суфікс `Log`.

`COLLATE` *`collation_name`*

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

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

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

`FOR LOAD`

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

`FOR ATTACH`

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

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

```
CREATE DATABASE mydatabase
```

Видалення бази даних

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

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

За допомогою єдиного параметра команди вказується ім'я бази даних, яку необхідно видалити. Однак за одну операцію можна видалити множину баз даних, просто перелічивши їхні імена через кому. При видаленні бази даних відбувається видалення рядка таблиці `sysdatabases`, що описує відповідну

базу даних. Також виконується й фізичне видалення всіх файлів, з яких складалася база даних. Проте, маючи резервну копію бази даних, згодом можна відновити її знову

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

Робота з таблицями

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

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

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

Створення таблиць

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

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

- **Check** — за допомогою логічних умов накладає обмеження на значення, які можуть зберігатися в стовпці;
- **Null** — задає можливість зберігання невизначених значень;
- **Default** — призначає значення по умовчанню;
- **Unique** — гарантує унікальність значень у стовпці;

- **Primary Key** — визначає первинний ключ;
- **Foreign Key** — визначає зовнішній ключ;
- **No Action** — пропонує не виконувати в залежній таблиці ніяких дій при видаленні або відновленні рядків у головній таблиці;
- **Cascade** — у цьому випадку буде здійснюватися каскадна зміна значень у залежній таблиці при внесенні змін у головну таблицю.

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

Таблиця 1 – Список типів даних, що використовуються для стовпців

Тип даних	Короткий опис
bigint	Цілочисельний тип даних, що займає 8 байт
binary	Двійкові дані фіксованої довжини до 8000 байт
bit	Один біт, приймає значення або 0, або 1
char	Символьні дані не Unicode фіксованої довжини до 8000 символів
datetime	Дата й час високої точності (8-байтовий) від 1 січня 1753 року до 31 грудня 9999 року, час визначається з точністю до 3.33 мілісекунд
decimal	Нецілочисельний тип даних фіксованої точності для якого вказується загальна кількість цифр в числі, та скільки з них цифр дробової частини, наприклад, decimal (7,2)
float	Нецілочисельний тип даних приблизної точності з діапазоном значень від 1.79E + 308 до 1.79E + 308
image	Двійкові дані довжиною до 2 Гбайт
int	Цілочисельний тип даних, що займає 4 байти; діапазон значень від -2^{31} (-2 147 483 648) до 2^{31}- 1 (2 147 483 647).
money	Грошовий тип даних високої точності (8-байтовий)
nchar	Символьні дані Unicode фіксованої довжини до 4000 символів
ntext	Текстові дані Unicode довжиною до 1 Гбайта
nvarchar	Символьні дані Unicode змінної довжини до 4000 символів
numeric	Нецілочисельний тип даних фіксованої точності (аналогічний decimal)
real	Нецілочисельний тип даних приблизної точності з діапазоном значень від –3.40E + 38 до 3.40E + 38; займає 4 байти

Тип даних	Короткий опис
smalldatetime	Дата й час низької точності (4-байтовий) від 1 січня 1 1900 року до 6 червня 2079 року, час визначається з точністю до половини хвилини. В smalldatetime значення секунд не більше 29.998 відкидаються; значення секунд від 29.999 – додає ще одну хвилину до часу
smallint	Цілочисельний тип даних, що займає 2 байти; діапазон значень від -32 768 до 32 767
smallmoney	Грошовий тип даних низкою точності (4-байтовий)
text	Текстові дані не Unicode довжиною до 2 Гбайт
tinyint	Цілочисельний тип даних, що займає 1 байт; діапазон значень від 0 до 255
varchar	Символьні дані не Unicode змінної довжини до 8000 символів

Часто набір полів таблиці є незручним для використання його як первинний ключ. Наприклад, у таблиці з описом людини в якості первинного ключа можна вибрати стовпець із номером паспорта і його серією. Однак іноді по тих або інших причинах людина міняє паспорт, і тоді змінюється значення серії і номеру паспорту, тобто повинно змінитися значення ключа таблиці. У цьому випадку необхідно виправити значення не тільки в головній таблиці, але й у залежних таблицях.

Ці й деякі інші причини змушують багатьох розроблювачів створювати в таблиці додатковий стовпець, єдине призначення якого – служити ідентифікаційним номером рядка в таблиці. Для цих полів зручно користуватися автоматичною нумерацією – під час додавання рядка в таблицю для поля з автонумерацією значення не вводиться, а SQL Server сам визначає наступне значення в стовпці.

Щоб дозволити поле таблиці для автоматичної нумерації, для нього необхідно встановити властивість IDENTITY (ID entity, ідентифікаційний номер сутності). При цьому користувач повинен задати початкове значення й крок приросту, наприклад, IDENTITY(10,5). Перший рядок, що вставляє в таблицю, буде мати значення, зазначене як початкове. Якщо початкове значення і крок приросту не вказано, то вони будуть дорівнюватись 1. Тобто запис IDENTITY(1,1) ідентичний запису IDENTITY. У кожній таблиці тільки для одного стовпця може бути встановлена властивість IDENTITY.

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

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

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

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

Параметри команди:

database_name

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

owner

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

table_name

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

<column_definition>

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

column_name AS computed_column_expression

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

У структурі таблиці зберігається тільки формула, по якій відбувається обчислення значень, тоді як самі значення не зберігаються. Тому неможлива вставка або зміна значень в полях, що обчислюють. При створенні стовпця,

що обчислюється, необхідно вказати його ім'я (аргумент *column_name*) і після ключового слова AS вираз (аргумент *computed_column_expression*), по якому буде обчислюватися значення поля.

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

<*table_constraint*>

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

[,...*n*]

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

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

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

Розглянемо призначення й використання параметрів команди.

column_name

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

data_type

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

DEFAULT *constant_expression*

За допомогою цього аргументу можна визначити значення за замовчуванням, що буде привласнюватися відповідному полю рядка, якщо при її вставці користувач явно не вказав конкретне значення. Значення за замовчуванням не може бути застосоване до стовпців з типом даних timestamp або із установленою властивістю IDENTITY. Як значення за замовчуванням можуть застосовуватися константи, системні змінні й

функції, а також будь-які вирази, побудовані на їхній основі. Використання посилань на інші стовпці заборонено.

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

Зазначення цього аргументу пропонує створити стовпець із підтримкою автоматичної нумерації. При вставці нового рядка в таблицю SQL Server 2000 автоматично забезпечує вставку в поле IDENTITY унікального значення, що монотонно збільшується при вставці кожного нового рядка. Властивість IDENTITY може бути встановлено тільки для стовпців з типом даних int, smallint, tinyint, decimal (p, 0) і numeric (p, 0). У межах однієї таблиці можна створити тільки один стовпець із установленою властивістю IDENTITY.

<column_constraint>

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

[...*n*]

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

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

Обмеження цілісності на рівні стовпців

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

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

Розглянемо призначення та використання основних аргументів:

CONSTRAINT *constraint_name*

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

NULL | NOT NULL

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

PRIMARY KEY

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

UNIQUE

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

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

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

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

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

ON DELETE {CASCADE | NO ACTION}

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

створюваній таблиці або скасуванні операції видалення рядка головної таблиці (при вказівці NO ACTION).

ON UPDATE {CASCADE | NO ACTION}

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

Обмеження цілісності на рівні таблиці

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

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

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

Параметри команди були розглянуті вище. Для демонстрації команд створення бази даних та створення таблиць, створимо часткову базу даних житлового фонду міста (`City`) з трьох таблиць: `Street` (Вулиця), `House`

(Будинок), Flat (Квартира). В таблиці Street буде лише два атрибута (стовпця): назва вулиці (name_street) та ідентифікатор вулиці в БД (id_street). В таблиці House будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці (number), кількість поверхів в будинку (count_floor), кількість квартир (count_flat). Оскільки будинки можуть не мати квартир (офіси, магазини), а також можуть мати не однакову кількість поверхів в різних парадних, то останні два атрибути можуть мати пусті значення (NULL). Первинний ключ визначається на рівні таблиці, тому що є складовим: складається з атрибутів id_street та number. В таблиці Flat будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці, номер квартири (flat_number), кількість кімнат у квартирі (rooms). У таблиці Flat на рівні таблиці визначаються і зовнішній ключ з двох атрибутів (первинний ключ таблиці House) і первинний ключ – вже з трьох атрибутів (додається ще номер квартири).

Для назви вулиці вибрано тип даних varchar(50) – рядок символів змінної довжини не більше 50 символів. Для номеру будинку вибрано тип даних char(8) – рядок з 8 символів, – щоб можна було зберігати номери будинків виду: 95а або 102/3. Інші атрибути будуть цілими числами або типу tinyint (якщо не перевищують значення 255), або типу smallint (якщо можуть мати значення більше за 255, але не більше 32767).

Команда: USE *database* – змінює активну базу даних на БД з іменем *database.*

Команда: GO – сигналізує про кінець пакету команд. Деякі команди не можуть виконуватись в одному пакеті, тому що друга команда може бути виконана тільки після успішного завершення першої команди. Наприклад, всі команди додавання записів в будь-яку одну таблицю можуть виконуватись в одному пакеті. Але додавання записів в таблицю з зовнішнім ключем (підлеглу таблицю) можливо тільки після того, як занесені відповідні дані в основну таблицю, в якій зовнішній ключ підлеглої таблиці є первинним ключем.

```
create database City
go
use city
go
create table Street(
id_street smallint identity primary key,
name_street varchar(50) not null)
go
create table House(
id_street smallint references street,
number char(8) not null,
count_floor tinyint,
count_flat smallint
primary key(id_street,number))
```

```
go
create table Flat(
id_street smallint not null,
number char(8) not null,
flat_number smallint not null,
rooms tinyint,
foreign key (id_street,number) references house,
primary key(id_street,number,flat_number)
)
```

Додавання записів в таблиці

Для цього використовується команда Transact-SQL INSERT:

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

Параметри команди:

INTO

Необов'язкове ключове слово.

table_name

Ім'я таблиці, в яку додаються записи.

column_list

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

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

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

VALUES

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

expression

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

Перелік запитань до лабораторної роботи №1

1. Які є об’єкти в базі даних?
2. Що таке «обмежені ідентифікатори»?
3. Яка команда Transact-SQL використовується для створення бази даних?
4. Яка команда Transact-SQL використовується для створення таблиці?
5. Які є основні типи даних в SQL Server 2000?
6. Які існують обмеження цілісності на рівні стовпця?
7. Яка команда Transact-SQL використовується для додавання записів в таблицю?

Завдання для лабораторної роботи №1:

Створення бази даних

Створити бази даних персоналу підприємства, в якій повинні бути виділені наступні базові сутності: **Відділ**, **Посада**, **Персона**. Ці сутності описують відділи підприємства, посади, що існують на підприємстві, та особові дані працівників підприємства (особові діла працівників).

У випадку заборони на підприємстві сумісництва працівників перерахованих сутностей достатньо. В цьому випадку сутність **Персона** повинна крім власне особистих атрибутів працівника: прізвище, ім’я, стать, дата народження, – повинна включати посилання на відділ та посаду, де працює дана особа.

Оскільки у загальному випадку зв'язок між таблицями **Персона** і **Відділ** (зв'язок між працівником та відділом, в якому він працює) та **Персона** і **Посада** (зв'язок між працівником та посадою, на якій він працює) є типу багато-до-багатьох, для створення реляційної бази даних потрібна похідна сутність **Працівник**, яка містить наступні відомості: якій працівник у якому відділі на якій посаді працює, на яку частину ставки, та є він основним працівником чи сумісником, дата прийому на роботу. Будемо вважати, що на підприємстві заборонено подвійне сумісництво, тобто особа може працювати або лише як основний працівник; або лише як сумісник; або як основний працівник і підробляти як сумісник у тому ж відділі, або у іншому, на тій самій посаді, або на іншій.

Крім того потрібна ще похідна сутність **Відпустка**, в якій буде вказано, якій основний працівник (код працівника) з якої дати йде у відпустку, та за який рік ця відпустка (будемо вважати, що всі працівники кожен рік повністю використовують свої відпустки).

Відношення (таблиці) бази даних **Персонал підприємства**:

Відділ (ідентифікатор відділу, назва відділу)

Посада (ідентифікатор посади, назва посади, зарплата, кількість днів щорічної відпустки, кількість ставок (кількість вакансій)).

Персона (ідентифікатор працівника, серія та номер паспорту, прізвище, ім'я, по-батькові, дата народження, стать, освіта, сімейне положення, кількість дітей)

Працівник (ідентифікатор працівника, ідентифікатор відділу, ідентифікатор посади, чи є основним працівником, розмір ставки, дата прийому на роботу)

Відпустка (ідентифікатор працівника, рік відпустки, дата початку відпустки)

Хоча MS SQL Server 2000 дозволяє використовувати ідентифікатори з кирилицею, більш зручним є використання ідентифікаторів, написаних тільки латиницею, тому схема бази даних буде наступна:

```
Otdel(Id_otd, Name_otd)
Post(Id_post, Name_post, Post_money, Days, Count_post)
Persona(Id_man, PassNo, Surname, Name1, Name2, Birthdate, Sex,
        Education, Married, Children)
Worker(Id_man, Id_otd, Id_post, main_work, Stavka, Date_work)
Vacation(Id_man, year_vac, date_vac)
```

Виконання завдання:

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

Створення бази даних командами мови SQL студенти вивчають в середині семестру. Тому для виконання даних лабораторних робіт вони використовують команди створення бази даних, наведені у даних методичних вказівках.

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

Первинними ключами таблиць Відділ, Посада, Персона будуть їх ідентифікатори. Унікальними повинні бути значення атрибутів: «назва відділу», «назва посади», «серія та номер паспорту».

В таблиці **Otdel** атрибут `Id_otd` є первинним ключем; в таблиці **Post** атрибут `Id_post` є первинним ключем; в таблиці **Persona** атрибут `Id_man` є первинним ключем; в таблиці **Worker** сукупність атрибутів: `Id_man, Main_work` – є первинним ключем.

В таблиці **Vacation** є потенційний (унікальний) складовий ключ: це сукупність атрибутів Id_man та year_vac. Його можна обрати у якості первинного ключа.

Для визначення типів даних спочатку визначається для кожного атрибуту його домен – область припустимих значень.

Для визначення типів даних для числових атрибутів треба визначити діапазон їх можливих значень. Будемо вважати, що різних посад на підприємстві менше 100, кількість днів щорічної відпустки також менше 100, кількість працівників може бути декілька сотень. Зарплатня не може перевищувати 99999 і може мати дрібну частину (копійки).

Для атрибутів, що мають значення цілого типу в діапазоні від 0 до 255 найкращим типом буде цілий тип розміром в один байт, тобто тип tinyint.

Для атрибутів, що мають значення цілого типу, які можуть перевищувати значення 255, але не перевищують значення 32767 найкращим типом буде цілий тип розміром в два байти, тобто тип smallint.

Для дати можна завжди вибирати тип smalldatetime (розміром в 4 байти), якщо потрібно зберігати саме дату, а не час, або дату+час. Якщо атрибут повинен зберігати час, або упаковані дату і час, то тип smalldatetime підходить у випадках, коли у запису часу суттєвими є тільки години і хвилини – наприклад, час відправлення потягу.

Таким чином отримуємо наступну структуру бази даних «Персонал підприємства», яка складається з п’яти таблиць:

Таблиця **Otdel** (Відділ)

Атрибут	Домен	Тип даних	Призначення
Id_otd	Ціле число менше 100	tinyint	Ідентифікатор відділу, первинний ключ
Name_otd	Рядок до 30 символів	varchar(30)	Назва відділу, унікальна

Таблиця **Post** (Посада)

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

Таблиця **Persona** (Персона)

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

Таблиця **Worker** (Працівник)

Атрибут	Домен	Тип даних	Призначення
Id_man	Ціле число менше 10000	smallint	Ідентифікатор працівника, зовнішній ключ
Id_otd	Ціле число менше 100	tinyint	Ідентифікатор відділу, зовнішній ключ
Id_post	Ціле число менше 100	tinyint	Ідентифікатор посади, зовнішній ключ
main_work	Логічне (так, ні)	bit	Чи приймають на основну роботу
stavka	Дійсне число від 0.1 до 1 з 2 цифрами дробової частини	numeric(4,2)	Частина ставка, на яку приймають працівника
date_work	Дата 20-21 століття	smalldatetime	Дата прийому на роботу
Id_man + main_work – унікальний ключ, що обирається як первинний			

Таблиця **Vacation** (Відпустка)

Атрибут	Домен	Тип даних	Призначення
Id_man	Ціле число менше 10000	smallint	Ідентифікатор працівника, зовнішній ключ
Year_vac	Ціле число порядку 2000	smallint	За який рік відпустка
date_vac	Дата 20-21 століття	smalldatetime	Дата виходу у відпустку
Id_man + Year_vac – унікальний ключ, що обирається як первинний			

Після визначення структури таблиць та зв'язків між ними можна написати код створення бази даних та таблиць. Як було вказано вище, команди

можуть містити логічні помилки, які не можуть бути розпізнані комп'ютером. Якщо команда з логічної помилкою вже виконана, потрібно усунути її наслідки. Якщо помилково створена таблиця не той структури, що потрібна, її можна або модифікувати, або цілком усунути і створити знову. Якщо таблиця щойно створена, простіше її усунути командою DROP TABLE, і знов створити вже правильну таблицю. Команда видалення таблиці має синтаксис:

DROP TABLE *table_name*

Команді створення бази даних Personal та відношень у базі даних Personal виглядають наступним чином (перші 4 рядки коду створюють базу даних Personal та роблять її активною):

```
CREATE DATABASE Personal
GO
USE Personal
GO
CREATE TABLE Otdel(
id_otd tinyint identity PRIMARY KEY,
name_otd varchar(25) UNIQUE NOT NULL
)
GO
CREATE TABLE Post(
Id_post tinyint IDENTITY PRIMARY KEY,
Name_post varchar(30) NOT NULL UNIQUE,
Post_money numeric(7,2) NOT NULL,
Days tinyint NOT NULL,
Count_post tinyint NOT NULL
)
GO
CREATE TABLE Persona(
Id_man smallint IDENTITY PRIMARY KEY,
PassNo char(8) NOT NULL UNIQUE,
Surname varchar(25) NOT NULL,
Name1 varchar(12) NOT NULL,
Name2 varchar(15),
Birthdate smalldatetime NOT NULL,
Sex char(1) NOT NULL,
Education varchar(15),
Married bit,
Children tinyint
)
GO
CREATE TABLE Worker(
id_man smallint REFERENCES Persona,
id_post tinyint REFERENCES Post,
id_otd tinyint REFERENCES Otdel,
main_work bit NOT NULL,
stavka numeric(4,2) NOT NULL,
date_work datetime,
PRIMARY KEY(id_man, main_work)
)
GO
```

```
CREATE TABLE Vacation(
id_man smallint REFERENCES Persona,
Year_vac smallint not null,
Date_vac smalldatetime not null,
PRIMARY KEY (id_man, Year_vac)
)
GO
```

Після створення БД потрібно внести інформацію в таблиці.

У таблицю **Відділ** занести 5 записів.

У таблицю **Посада** занести 10 записів.

У таблицю **Персона** занести дані 30 осіб різної статі (приблизно однокової кількості чоловік та жінок), різного віку (від 18 до 65 років) с різним рівнем освіти (середня, н/вища, вища), різним сімейним станом, різною кількістю дітей (2-3 працівника повинні мати більше 2 дітей, від чверті до третини працівників повинні не мати дітей, інші – мати одного або двох дітей).

У таблицю **Працівник** занести 35 рядків з даними – 20 осіб повинні працювати лише як основні працівники (main_work = 1) на ставці від 0.5 до 1.0; 5 осіб повинні працювати лише як сумісники (main_work = 0) на ставці від 0.25 до 0.5; 5 осіб повинні працювати одночасно як основні працівники (на ставку до 1.0) та як сумісники на ставки від 0.25 до 0.5.

У таблицю **Відпустка** занести 25 даних про відпустку у поточному році робітників по основній роботі.

В таблицю `Otdel` потрібно вставити 5 записів. Три записи наводяться нижче, для інших відділів студенти вибирають назву самі у залежності від того, які посади вони будуть додавати у таблицю Post. Можливі назви відділу: Комп'ютерний, Маркетингу, Виробничій, тощо.

```
INSERT Otdel VALUES('Администрация')
INSERT Otdel VALUES('Бухгалтерия')
INSERT Otdel VALUES('Технический')
GO
```

Заповнення таблиці `Post` (2-3 записи студенти додають у таблицю `Post` самі):

```
INSERT Post (Name_post,Post_money,Days, Count_post)
Values ('Директор',9000,30,1)
INSERT Post(Name_post,Post_money,Days, Count_post)
Values('Зам.директора',8000,25,1)
INSERT Post(Name_post,Post_money,Days, Count_post)
Values('Секретарь-референт',4000,20,1)
INSERT Post(Name_post,Post_money,Days, Count_post)
Values('Главний бухгалтер',6000,21,1)
INSERT Post(Name_post,Post_money,Days, Count_post)
Values('Бухгалтер',4000,20,4)
```

```
INSERT Post(Name_post,Post_money,Days,
Count_post)Values('Менеджер',3500,20,5)
INSERT Post(Name_post,Post_money,Days,
Count_post)Values('Инженер',3000,20,5)
INSERT Post(Name_post,Post_money,Days,
Count_post)Values('Секретарь',2500,20,3)
```

Для таблиці Persona наведемо частковий код заповнення таблиці (24 записи студенти додають у таблицю Persona самі):

```
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex, Married,Children)VALUES('КК111111','Иванов','Илья',
'Иванович', '12.20.1964', 'высшее', 'м',0,1)
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex,Married,Children)VALUES('КК222222','Петрова','Ольга','Ивановна
', '10.25.1972', 'высшее', 'ж',1,2)
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex,Married,Children)VALUES('КК111112','Зимина','Анна','Львовна',
'07.22.1990', 'н/высшее', 'ж',1,0)
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex,Married,Children)VALUES('КК111122','Потапов','Иван','Ильич',
'12.13.1978', 'высшее', 'м',0,2)
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex,Married,Children)VALUES('КК121212','Сидорова',
'Инна','Сергеева','08.11.1975', 'н/высшее', 'ж',0,0)
INSERT Persona(PassNo,Surname,Name1,Name2,Birthdate, Education,
Sex,Married,Children)VALUES('КК111222','Кузнецова','Ольга',
'Петровна','05.12.1985', 'среднее', 'ж',1,0)
```

Для прикладу заповнення таблиці Worker наведемо код прийому на роботу 5 перших працівників (з ідентифікаторами від 1 до 5). Перші 3 приймаються на основну роботу (main_work = 1) на повну ставку (1.0), 5-й приймається на роботу як сумісник (main_work = 0) на ставку 0.5, 4-й приймається на основну роботу (main_work = 1) на повну ставку (1.0) та на роботу за сумісництвом (main_work = 0) на ставку 0.25:

```
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(1,1,1,1,1.0,’2008-02-05’)
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(2,1,2,1,1.0,’2008-02-15’)
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(3,1,3,1,1.0,’2012-02-09’)
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(5,2,5,0,0.5,’2016-01-09’)
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(4,2,4,1,1.0,’2010-03-25’)
INSERT Worker(Id_man,Id_otd,Id_post, main_work, stavka, date_work)
Values(4,2,5,0,0.25,’2016-02-03’)
```

Для прикладу заповнення таблиці Vacation наведемо код завдання дати виходу у відпустку у 2016 році 3 перших працівників (з ідентифікаторами від 1 до 3):

```
INSERT Vacation(Id_man,year_vac,date_vac)Values(1,2016,'2016-07-15')
INSERT Vacation(Id_man,year_vac,date_vac)Values(2,2016,'2016-08-25')
INSERT Vacation(Id_man,year_vac,date_vac)Values(3,2016,'2016-09-01')
```

Студенти повинні ввести в таблицю Vacation 25 записів – для всіх працівників, що в таблиці прийняті на основну роботу, потрібно вказати дату виходу у відпустку для відпустки поточного року. У кожному місяці принаймні один працівник повинен бути у відпустці, але щонайменше половина працівників повинна відгуляти відпустку влітку.

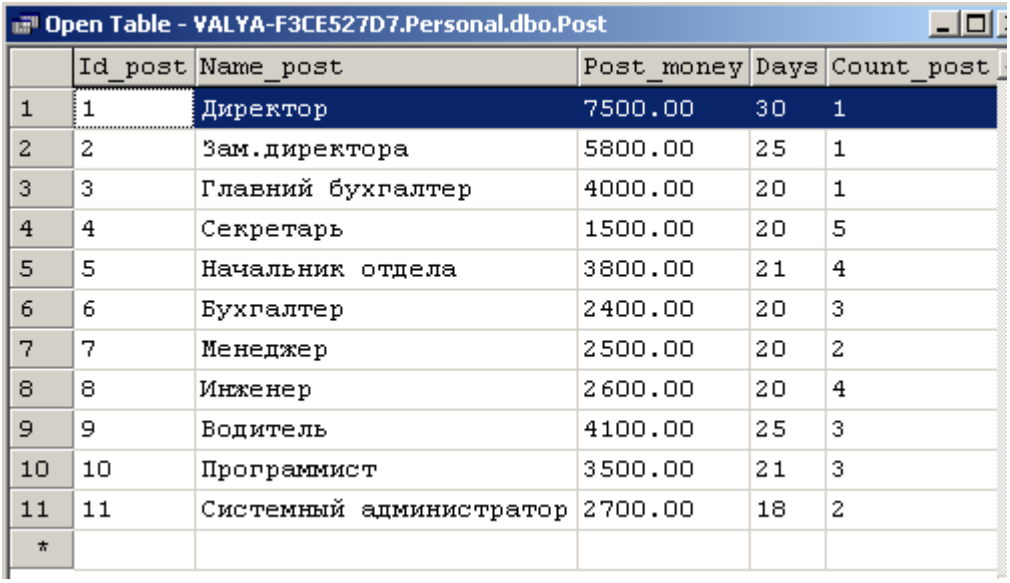
Для перегляду вмісту таблиць після їх заповнення потрібно в **Object Browser** клацнути правою кнопкою миші по імені таблиці, і вибрати пункт контекстного меню "Open".

Відкриється вміст таблиці, як видно на рисунках 3-4.

Open Table - VALYA-F3CE527D7.Personal.dbo.Persona

	Id_man	PassNo	Surname	Name1	Name2	Birthdate	Sex	Education	Married	Children
1	1	КМ111555	Королева	Алла	Сергеевна	1979-03-20 00:00:00	ж	высшее	1	2
2	2	ЛМ333222	Рябинина	Ольга	Игоревна	1982-10-07 00:00:00	ж	высшее	1	2
3	3	КМ119922	Рыбников	Олег	Петрович	1985-06-12 00:00:00	м	среднее	1	0
4	4	КК111112	Стрижов	Олег	Львович	1982-07-22 00:00:00	м	н/высшее	1	0
5	5	КК177122	Кобзарь	Иван	Иванович	1977-12-13 00:00:00	м	высшее	0	2
6	6	КК121212	Холодова	Лидия	Ивановна	1955-08-11 00:00:00	ж	н/высшее	0	0
7	7	МК118811	Трубин	Антон	Иванович	1980-01-25 00:00:00	м	высшее	1	0
8	8	МК220022	Волков	Тарас	Петрович	1990-03-25 00:00:00	м	высшее	0	2
9	9	МК110222	Туманова	Евгения	Антоновна	1990-02-12 00:00:00	ж	среднее	0	1
10	10	МК101112	Степная	Татьяна	Семеновна	1978-04-22 00:00:00	ж	высшее	0	1
11	11	МК110122	Соколов	Сергей	Ильич	1949-12-13 00:00:00	м	н/высшее	1	1
12	12	МК122112	Сидоренко	Наталья	Сергеева	1965-06-11 00:00:00	ж	высшее	0	1
13	13	ЛК110001	Зайцев	Сергей	Иванович	1980-09-05 00:00:00	м	высшее	0	1
14	14	ЛК200022	Клюквина	Ольга	Петровна	1963-03-25 00:00:00	ж	высшее	1	3
15	15	ЛК888222	Зайцева	Алла	Ивановна	1974-11-19 00:00:00	ж	высшее	0	2
16	16	ЛК777112	Воробьев	Петр	Львович	1985-07-28 00:00:00	м	н/высшее	0	0
17	17	ЛК999122	Стриженов	Олег	Ильич	1984-12-13 00:00:00	м	высшее	1	1
18	18	ЛК126612	Ланская	Карина	Андреевна	1991-07-11 00:00:00	ж	высшее	0	1

Рисунок 3 – Вміст таблиці Persona



Open Table - VALYA-F3CE527D7.Personal.dbo.Post

	Id_post	Name_post	Post_money	Days	Count_post
1	1	Директор	7500.00	30	1
2	2	Зам.директора	5800.00	25	1
3	3	Главний бухгалтер	4000.00	20	1
4	4	Секретарь	1500.00	20	5
5	5	Начальник отдела	3800.00	21	4
6	6	Бухгалтер	2400.00	20	3
7	7	Менеджер	2500.00	20	2
8	8	Инженер	2600.00	20	4
9	9	Водитель	4100.00	25	3
10	10	Программист	3500.00	21	3
11	11	Системный администратор	2700.00	18	2
*					

Рисунок 4 – Вміст таблиці Post

Формат дати ‘yyyy.mm.dd’ (ANSI формат) спрацює завжди, незалежно від вибраного поточного формату представлення дати та часу. За замовчуванням в програмі працює також американській формат запису дати, тобто ‘mm.dd.yyyy’, якщо не виконувалась команда зміни формату дати. За бажанням під час сеансу з SQL Server можна встановити зручний для користувача формат дати/часу командою SET DATEFORMAT:

SET DATEFORMAT *format*

Тут *format* може приймати одне з наступних значень: mdy, dmy, ymd, ydm, myd, dym – яке вказує в якому порядку в даті записуються цифри дня (d), місяця (m), року (y).

Для надійності можна завжди використовувати ANSI формат запису дати і часу, який в повній формі виглядає так:

```
'yyyy.mm.dd hh:mi:ss.mmm'
```

Тут yyyy – 4 цифри року;
mm – 2 цифри місяца;
dd – 2 цифри номеру дня;
hh – 2 цифри годин;
mi – 2 цифри хвилин;
ss – 2 цифри секунд;
mmm – 3 цифри мілісекунд.

Захист лабораторної роботи №1

Для захисту цієї роботи потрібно оформити звіт у електронній формі. У звіті повинні бути відповіді на всі контрольні запитання та Screenshot для кожної таблиці бази даних. Студент повинен продемонструвати викладачеві створену і заповнену базу даних, та відповісти на усні запитання.

Лабораторна робота №2 Прості запити до БД

Мета роботи:

1. Отримання практичних навичок створення простих запитів мовою SQL у СКБД MS SQL Server 2000.
2. Знайомство з функціями Transact-SQL.

Перелік тем лекційного курсу

1. Синтаксис команди SELECT.
2. Розділи команди SELECT: SELECT, FROM, WHERE, ORDER BY.

Вибірка даних в Query Analyzer MS SQL Server 2000

Команда SELECT

Вибірка даних – найбільш розповсюджена операція під час роботи з базою даних. Вона здійснюється запитами до БД – командою Transact-SQL SELECT. Команда SELECT отримує записи з бази даних та дозволяє вибірку одного чи більше рядків або стовпців з однієї або декількох таблиць. Результатом дії будь-якої команди SELECT є нова таблиця, тому можливі багаторівневі запити, в яких вкладені запити (запити нижнього рівня), повертають таблицю, що використовується у запиті вищого рівня.

Хоча мова SQL і є стандартом мови для реляційних баз даних, існує багато «діалектів» мови SQL, тобто мова SQL конкретної СКБД має деякі, властиві саме їй, відмінності. Тому розглянемо головну команду мови SQL в контексті СКБД MS SQL Server 2000 більш докладно.

Повний синтаксис цієї команди досить складний, але можна перелічити основні розділи команди, а саме:

SELECT *select_list*
[INTO *new_table*]
FROM *table_source*
[WHERE *search_condition*]
[GROUP BY *group_by_expression*]
[HAVING *search_condition*]
[ORDER BY *order_expression* [ASC | DESC]]

Розділ SELECT (обов'язковий): SELECT *select_list* – він вказує стовпці таблиці (*select_list*), яка повертається запитом.

Розділ FROM (обов'язковий): FROM *table_source* – він вказує з яких таблиць (*table_source*): базових відношень, представлень, віртуальних таблиць, – отримуються дані.

Необов'язкові розділи:

INTO *new_table*

Створює таблицю і записує до неї результат запиту (без цього розділу результат запиту є віртуальною таблицею).

WHERE *search_condition*

Містить умови пошуку рядків таблиць, які повинні брати участь у запиті.

GROUP BY *group_by_expression*

Містить перелік стовпців таблиці, за якими буде проводитись групування вибраних даних для отримання характеристик цих груп: кількості записів в групі, середнього значення, суми, максимуму або мінімуму якогось поля (чи виразу з полів).

HAVING *search_condition*

Містить умови відбору рядків результуючої таблиць, коли в запиті використовується групування; в цьому розділі, на відміну від розділу WHERE, звичайно записуються умови для груп; в одному запиті (з групуванням) можуть бути умови в розділі WHERE (на записи в таблицях, наприклад, будинки з кількістю поверхів не менше 9), та умови на групи в розділі HAVING (наприклад, групування виконується по вулицям, та вибираються вулиці, на яких не менше 10 багатоповерхових будинків).

ORDER BY *order_expression*

Містить перелік стовпців результуючої таблиці, за якими виконується упорядкування даних. Сортування може проводитись по декількох стовпцях (багаторівневе упорядкування).

Розглянемо докладніше деякі розділи команди SELECT.

Розділ SELECT

Вказує стовпці таблиці, яка повертається запитом.
Синтаксис команди:

```
SELECT [ ALL | DISTINCT ]
  [ TOP n [ PERCENT ] [ WITH TIES ] ]
  < select_list >
```

Параметри команди:

ALL

Вказує, що однакові (дублюючі) рядки повертають запитом. За замовчуванням (коли не вказано ALL або DISTINCT) використовується ALL. Однакові рядки можуть з'явитися в результату запиту з таблиць, в яких немає дублюючих записів. Наприклад, з бази даних підприємства обирається наступна інформація про працівників: прізвище, ім'я, посада. Зрозуміло, що на підприємстві можуть виявитись працівники з однаковими іменем та прізвищем на однаковій посаді. В них є характеристики, що відрізняють їх один від одного, принаймні такі, як ідентифікаційний номер та серія і номер паспорту. Тому в базі даних немає дублюючих записів, але в запиті вони з'являються за рахунок того, що вибирається часткова інформація про працівників.

DISTINCT

Вказує, що однакові (дублюючі) рядки не повертають запитом.

TOP *n* [PERCENT]

Вказує, що повертаються тільки перші *n* рядків з таблиці, що є результатом запиту. *n* є ціле число у діапазоні між 0 та 4294967295. Якщо є ключове слово PERCENT, то повертається *n* відсотків записів з результуючої таблиці. Наприклад, в результаті запиту без слова TOP повинна була повернутись таблиця з 40 рядків. Якщо додати до запиту: TOP 10 – повернеться таблиця з 10 рядків. Якщо додати до запиту: TOP 10 PERCENT – повернеться таблиця з 4 рядків. Параметр звичайно застосовують в запитах з розділом ORDER BY, тобто тих, що повертають упорядковані дані. Наприклад, якщо запит повертає прізвище, ім'я та зарплатню працівників підприємства, і є упорядкування за убуванням зарплатні, то наявність в запиті опції: TOP 10 – дозволить вибрати 10 працівників з самою високою зарплатнею.

WITH TIES

Ці ключові слова параметра TOP вказують, що в результат запиту потрібно включити всі записи з однаковим значенням атрибуту упорядкування, якщо хоч один запис входить до кількості TOP *n* (або TOP *n* PERCENT). Наприклад, є 8 працівників з самою високою зарплатнею, і 4 працівника з однаковою зарплатнею дещо нижчою, але більшою за решту працівників. Тоді запит з параметром: TOP 10 – поверне 10 записів; а запит з параметром: TOP 10 WITH TIES – поверне 12 записів, тобто всіх 4 працівників з однаковою зарплатнею.

< select_list >

Перелік стовпців таблиці, яка повертаються запитом. В переліку окремі пункти вибору (імена атрибутів базових таблиць, представлень, віртуальних таблиць, або виразів для вирахування значення в стовпці) відокремлюються один від одного комою.

Опція < select_list > має наступний синтаксис:

```
< select_list > ::=
    {   *
      | { table_name | view_name | table_alias }.*
      |    { column_name | expression | IDENTITYCOL | ROWGUIDCOL }
          [ [ AS ] column_alias ]
      | column_alias = expression
    }   [ ,...n ]
```

Параметри:

*

Вказує, що повертаються всі поля всіх таблиць, що є в розділі FROM. Тобто команда: SELECT * FROM *table_name* – поверне просто всю таблицю з іменем *table_name*, як вона є.

{*table_name* | *view_name* | *table_alias* }**.***

Обмежує зону дії параметру * одною вказаною таблицею. Тут в якості імені таблиці може використовуватись або ім'я базового відношення БД (*table_name*), або ім'я представлення (*view_name*), або псевдонім базової таблиці, представлення, віртуальної таблиці (*table_alias*). Псевдоніми визначаються в розділі FROM (розглядається нижче). Наприклад, з бази даних City потрібно вибрати інформацію про всі будинки на всіх вулицях, але для кожного будинку вказати назву вулиці. В таблиці House є тільки ідентифікатор вулиці, але назва вулиці є в таблиці Street. Отже, наступний запит вибере потрібну інформацію:

```
SELECT name_street, House.*
FROM Street, House
WHERE Street.id_street=House.id_street
```

column_name

Ім'я стовпця в таблиці, що повертається запитом. Потрібно кваліфікувати ім'я стовпця (уточнювати іменем таблиці), якщо існують стовпці з дублюючими іменами в таблицях, що перелічені в розділі FROM. Наприклад, атрибут id_street є як в таблиці Street, так і в таблиці House. Якщо запит вибирає цей атрибут, і в розділі FROM є обидві таблиці, то запис імені стовпці просто id_street викликає помилку: Ambiguous column name 'id_street'. Щоб уникнути подібної помилки, треба написати Street.id_street (або House.id_street).

expression

Вираз, що обраховує значення в стовпці. Це може бути ім'я атрибуту таблиці, константа, функція, будь-яка комбінація атрибутів, констант, і функцій у вигляді виразу, або вкладений запит (підзапит).

IDENTITYCOL

Повертає атрибут, що описаний в команді створення таблиці з властивістю IDENTITY. Якщо в розділі FROM є декілька таблиць, що мають такі атрибути, необхідно кваліфікувати поле, наприклад: Street.IDENTITYCOL

column_alias

Альтернативне ім'я стовпця (псевдонім стовпця) в результуючій таблиці. Наприклад, можна написати:

```
SELECT name_stree AS "Назва вулиці"
```

Якщо псевдонім стовпці є ідентифікатором (тобто містить тільки літери, цифри та знак підкреслення й починається є літери або підкреслення), то необов'язково включати його в обмежувачі. У наведеному прикладі псевдонім стовпця повинен бути в обмежувачах (подвійних лапках), тому що в ньому є символ пробілу (пропуску).

Псевдоніми стовпців потрібні в тих випадках, коли для обчислення значення в стовпці використовується вираз. Якщо такому стовпцю не надати псевдоніма, то його ім'я буде: (No column name).

column_alias може використовуватись в розділі ORDER BY, якщо виконується упорядкування за вказаним стовпцем. Але в розділах WHERE, GROUP BY, HAVING заборонено використовувати псевдоніми стовпців. Ключове слово AS перед псевдонімом стовпця можна пропускати.

column_alias = *expression*

Аналогічно запису: *expression* AS *column_alias*

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

В розділі SELECT можна через кому записати декілька стовпців результуючої таблиці за описаними вище правилами.

Розділ FROM команди SELECT

Вказує таблиці, представлення, віртуальні таблиці (що вираховуються вкладеним запитом) та з'єднані таблиці, що використовуються у командах DELETE, SELECT, UPDATE.

Синтаксис команди:

```
FROM { < table_source > } [ ,...n ]

< table_source > ::=
   table_name [ [ AS ] table_alias ]
   | view_name [ [ AS ] table_alias ]
   | derived_table [ AS ] table_alias
   | < joined_table >

< joined_table > ::=
   < table_source > < join_type > < table_source > ON < search_condition >
   | < table_source > CROSS JOIN < table_source >


< join_type > ::=
   [ INNER | { { LEFT | RIGHT | FULL } [ OUTER] } ]
   [ < join_hint > ]
   JOIN
```

Параметри команди:

table_name

Ім'я таблиці.

[AS] *table_alias*

Псевдонім таблиці, під яким вона буде фігурувати у інших розділах команди SELECT (DELETE, UPDATE).

< joined_table >

Результуючий набір, що є декартовим добутком двох або більше таблиць. Як відомо, добуток таблиць R і S визначає нове відношення, що є

результатом конкатенації (тобто зчеплення) кожного кортежу з відношення R з кожним кортежем з відношення S. Деякі види з'єднання таблиць R і S визначають відношення, що містять деяку підмножину декартового добутку таблиць R і S; але це не зовсім вірно щодо зовнішніх з'єднань. Розділ FROM дозволяє реалізувати як різні види з'єднань таблиць (з опцією <table_source> <join_type> <table_source> ON <search_condition>) так і операцію реляційної алгебри *перетинання* (з опцією < table_source > CROSS JOIN < table_source >).

< join_type >

Вказує тип операції з'єднання.

INNER

Вказує, що повертаються всі пари рядків, які відповідають умові з'єднання. Виключаються рядки обох таблиць, які на відповідають умові з'єднання. Цей тип зв'язку є типом зв'язку за замовчуванням, тобто конструкція:

```
table_name1 JOIN table_name2 ON search_condition
```

еквівалентна конструкції:

```
table_name1 INNER JOIN table_name2 ON search_condition
```

FULL [OUTER]

Вказує, що повертаються всі рядки обох таблиць. При цьому, ті рядки, що відповідають умові з'єднання таблиць, повертаються зв'язаними один з одним (як для зв'язку `INNER JOIN`). Але повертаються і додаткові рядки: всі рядки з обох таблиць повинні бути повернуті. Якщо для рядка з лівої таблиці (ліворуч від ключового слова FULL) немає відповідного рядка в правій таблиці, він повертається доповнений значеннями NULL для всіх атрибутів правої таблиці. Аналогічно, якщо для рядка з правої таблиці (праворуч від ключового слова FULL) немає відповідного рядка в лівій таблиці, він повертається доповнений значеннями NULL для всіх атрибутів лівої таблиці.

LEFT [OUTER]

На відміну від `FULL JOIN` не повертаються рядки правої таблиці, які не мають пари в лівій таблиці. Тобто повертаються всі рядки лівої таблиці, або зв'язані з рядками правої таблиці (якщо вони відповідають умові з'єднання), або доповнені значеннями NULL для всіх атрибутів правої таблиці (якщо вони не відповідають умові з'єднання).

RIGHT [OUTER]

На відміну від `FULL JOIN` не повертаються рядки лівої таблиці, які не мають пари в правій таблиці. Тобто повертаються всі рядки правої таблиці, або зв'язані з рядками лівої таблиці (якщо вони відповідають умові з'єднання), або доповнені значеннями NULL для всіх атрибутів лівої таблиці (якщо вони не відповідають умові з'єднання).

JOIN

Вказує, що визначений оператор зв’язку використовується до заданих таблиць або представлень.

ON <search_condition>

Вказує умову зв’язку таблиць. Умова повинна бути предикатом, в якому використовуються атрибути таблиць та операції порівняння. Наприклад, якщо в запиті використовуються таблиці `Street` і `House` (БД `City`), то умова зв’язку буде виглядати так:

```
ON Street.id_street=House.id_Street
```

Якщо в запиті використовуються таблиці `Flat` і `House` (БД `City`), то умова зв’язку буде виглядати так:

```
ON Flat.id_street=House.id_Street AND
     Flat.id_house=House.id_house
```

Розділ WHERE

Вказує умову, якій повинні відповідати рядки, що повертаються запитом.

Синтаксис команди:

WHERE < search_condition >

Параметри:

<search_condition>

Умова, якій повинні відповідати рядки, що повертаються запитом. Умова є комбінацією одного чи більше предикатів з використанням логічних операторів AND, OR, NOT. В цьому розділу крім умови пошуку рядків (умови фільтру), можна також включати умови зв’язків між таблицями, тоді в розділі FROM таблиці лише перелічуються через кому.

Формат запису умови:

< search_condition > ::=
{ [NOT] < predicate > | (< search_condition >)
[{ AND | OR } [NOT] { < predicate > | (< search_condition >) }]
}

< predicate > ::=
{ *expression* { = | < > | ! = | > | > = | ! > | < | < = | ! < } *expression*
| *string_expression* [NOT] LIKE *string_expression*
| *expression* [NOT] BETWEEN *expression* AND *expression*
| *expression* IS [NOT] NULL
| *expression* [NOT] IN **(** *subquery* | *expression* [**,***...n*] **)**
| *expression* { = | < > | ! = | > | > = | ! > | < | < = | ! < }
{ ALL | SOME | ANY} **(** *subquery* **)**
}

Параметри:

NOT

Інвертує значення логічного виразу: `NOT TRUE=FALSE, NOT FALSE=TRUE.`

AND

Комбінує дві умови та обраховує TRUE, якщо результат обох умов є TRUE.

OR

Комбінує дві умови та обраховує TRUE, якщо результат хоча б одної з умов є TRUE.

< predicate >

Вираз, який повертає значення TRUE, FALSE, або UNKNOWN.

expression

Атрибут таблиці, константа, функція, скалярний підзапит (тобто той, що повертає одне значення), або будь-яка комбінація атрибутів, констант, функцій, зв'язаних операторами.

Оператори порівняння (= , < > , ! = , > , > = , ! > , < , < = , ! <) докладно тут не розглядаються, тому що вони вже використовувались в інших дисциплінах на молодших курсах.

string_expression

Рядок символів та шаблонів символів.

[NOT] LIKE

Вказує, що символьний рядок праворуч містить шаблон символів для порівняння.

[NOT] BETWEEN

Вказує діапазон значень. Ключове слово AND відділяє початкове значення діапазону від кінцевого значення.

IS [NOT] NULL

Вказує пошук порожніх значень (NULL), або не порожніх значень (при наявності слова NOT).

[NOT] IN **(** *subquery* | *expression* [**,***...n*] **)**

Вказує на пошук рядків таблиць, для яких значення виразу ліворуч від ключового слова IN, приймає значення, яке є в множині значень, записаних у круглих дужках праворуч від ключового слова IN (або немає в множині значень, з словом NOT). Множина значень задається або переліком виразів (через кому), або вкладеним запитом, який повертає один стовпець значень.

ALL

Використовується з оператором порівняння та вкладеним запитом. Повертає TRUE для предикату, якщо всі значення, в множині значень, що повертаються вкладеним запитом, задовольняють оператору порівняння, або FALSE, якщо не всі значення з множини задовольняють оператору порівняння, або якщо вкладений запит не повертає жодного рядка. Вкладений запит повинен повертати таблицю з одного стовпця.

{ SOME | ANY }

Використовується з оператором порівняння та вкладеним запитом. Повертає FALSE для предикату, якщо всі значення, в множині значень, що повертаються вкладеним запитом, не задовольняють оператору порівняння, або якщо вкладений запит не повертає жодного рядка, та повертає TRUE, якщо деякі значення з множини задовольняють оператору порівняння, Вкладений запит повинен повертати таблицю з одного стовпця.

Приклади правильних предикатів:

```
rooms>3
YEAR(DATE())- YEAR(birthday)>=50
name_street LIKE '%басов%'
count_flat IS NOT NULL
id_house IN (SELECT id_house FROM House WHERE count_floor>5)
count_flat > ANY (SELECT count_flat FROM House H, Street S
                  WHERE H.id_street=S.id_street AND
                      name_street='Дерібасовська')
```

Розділ ORDER BY

Вказує сортування результуючого набору. Розділ ORDER BY є неприпустимим у представленнях та вкладених запитах; але може використовуватись в них, якщо в них вказаний параметр TOP.

Синтаксис команди:

ORDER BY { *order_by_expression* [ASC | DESC] } [**,**...*n*]

Параметри:

order_by_expression

Вказує стовпець, по якому проводиться упорядкування. Стовпець сортування вказується по імені, псевдоніму або виразу, що використовується для його обчислення. Також можна вказати ціле число (починаючи з 1), яке відповідає позиції стовпця упорядкування, рахуючи зліва направо. Припускається багаторівневе упорядкування. Послідовність стовпців сортування у розділі ORDER BY визначає

організацію сортування у результуючому набору. Можна сортувати за атрибутами таблиць, які не вказані в розділі SELECT. Не можна сортувати за стовпцями з даними типу **ntext**, **text**, або **image.**

ASC

Вказує, що значення в зазначеному стовпці повинні упорядковуватись в зростаючому (ascending) порядку від найменшого значення до найбільшого. Цей порядок обирається за замовчуванням.

DESC

Вказує, що значення в зазначеному стовпці повинні упорядковуватись в порядку убування (descending) від найбільшого значення до найменшого.

Функції Transact-SQL

SQL Server має велику кількість функцій, повний перелік яких можна отримати у файлі допомоги програми (Help). В цьому розділі наводяться лише функції, які потрібні для виконання варіантів завдань лабораторної роботи. Оскільки запити до БД можна писати різними способами і з використанням різних функцій, то студенти можуть під час виконання лабораторної роботи використовувати функції, які не перелічені в цьому розділі.

Функції дати та часу

Функція **GETDATE**

Повертає поточне системне значення дати і часу.

Синтаксис

GETDATE **( )**

Тип значення, що повертається: datetime

Функція DAY

Повертає ціле число, що відповідає номеру дня (в місяці) в зазначеній даті.

Синтаксис

DAY **(** *date* **)**

Аргументи:

date

Дата, з якої вибирається вказана частина дати/часу – вираз, що повертає значення типу **datetime** або **smalldatetime**, або символьний рядок у форматі дати.

***Тип значення, що повертається*:** int

Функція MONTH

Повертає ціле число, що відповідає номеру місяця в зазначеній даті.

Синтаксис

MONTH **(** *date* **)**

Аргументи:

date

Дата, з якої вибирається вказана частина дати/часу – вираз, що повертає значення типу **datetime** або **smalldatetime**, або символьний рядок у форматі дати.

***Тип значення, що повертається*:** int

Функція YEAR

Повертає ціле число, що відповідає номеру року в зазначеній даті.

Синтаксис

YEAR **(** *date* **)**

Аргументи:

date

Дата, з якої вибирається вказана частина дати/часу – вираз, що повертає значення типу **datetime** або **smalldatetime**, або символьний рядок у форматі дати.

***Тип значення, що повертається*:** int

Функція **DATEPART**

Повертає ціле значення, що відповідає зазначеній частині дати/часу в визначений даті.

Синтаксис

DATEPART **(** *datepart* **,** *date* **)**

Аргументи:

date

Дата, з якої вибирається вказана частина дати/часу – вираз, що повертає значення типу **datetime** або **smalldatetime**, або символьний рядок у форматі дати.

datepart

Цей параметр вказує, яка складова дати або часу вибирається з вказаного значення дати і часу. В таблиці 2 перелічені можливі складові дати і часу і їх абревіатура, що використовується в функції.

Таблиця 2 – Складові частини дати і часу

Складова дати/часу	Абревіатура
Year – рік	**yy, yyyy**
quarter – квартал	**qq, q**
Month – місяць	**mm, m**
dayofyear – день року	**dy, y**
Day – день (місяца)	**dd, d**
Week – тиждень	**wk, ww**
Hour – година	**hh**
minute – хвилина	**mi, n**
second – секунда	**ss, s**
millisecond – мілісекунда	**ms**

***Тип значення, що повертається*:** int

Зауваження

Функції DAY(*date*), MONTH(*date*), та YEAR(*date*) є синонімами функцій DATEPART(**dd**, *date*), DATEPART(**mm**, *date*), and DATEPART(**yy**, *date*), відповідно.

Приклад застосування функції DATEPART: нехай сьогодні 1 вересня 2012 року. Тоді:

```
DATEPART (yyyy, GETDATE())   поверне 2012  – рік
DATEPART (mm, GETDATE())     поверне 9     – номер місяца
DATEPART (dd, GETDATE())     поверне 1     – номер дня в місяці
DATEPART (qq, GETDATE())     поверне 3     – номер кварталу
```

Функція **DATEADD**

Повертає нове значення дати і часу, основане на додаванні заданого інтервалу до визначеного значення.

Синтаксис

DATEADD **(** *datepart* **,** *number***,** *date* **)**

Аргументи:

datepart

Цей параметр вказує, яка складова дати або часу додається до початкового значення дати і часу. В таблиці 2 перелічені можливі складові дати і часу і їх абревіатура, що використовується в функції.

number

Ціле значення, що використовується для збільшення дати. До дати додається *number* складових часток дати/часу (*datepart*). *number* може бути негативним значенням.

date

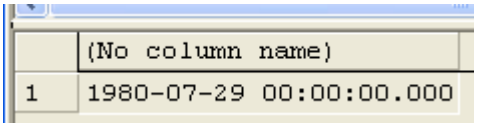
Дата, до якої додається задане значення – вираз, що повертає значення типу **datetime** або **smalldatetime**, або символьний рядок у форматі дати.

***Тип значення, що повертається*:** тип **datetime**, але якщо аргумент *date* має тип **smalldatetime**, то повертається **smalldatetime**.

Приклад застосування функції: нехай особа народилась 15 липня 1980 року. Дата народження записана в атрибуті `birthdate` таблиці Persona. Тоді команда:

```
SELECT DATEADD(ww,2,birthdate) FROM Persona
```

Поверне:



	(No column name)
1	1980-07-29 00:00:00.000

Тобто повернулась дата через два тижня після дати народження особи.

Функція **DATEDIFF**

Визначає, скільки зазначених часток дати/часу вміщується у діапазоні між двома вказаними датами.

Синтаксис

DATEDIFF **(** *datepart* **,** *startdate* **,** *enddate* **)**

Аргументи:

datepart

Складова частина дати/часу (див.таблицю 2).

startdate

Початкове значення дати.

enddate

Кінцеве значення дати.

***Тип значення, що повертається*:** int

Зауваження: Якщо значення *startdate* перевищує *enddate*, повертається негативне число.

Приклад застосування функції:

```
DATEDIFF (dd, '2010.12.25', '2011.02.11')
```

Поверне значення 48, тобто стільки днів пройшло від першої дати до другої.

Функції символьних рядків

Функція LEFT

Повертає частину символьного рядка, починаючи с початку заданого рядка і довжиною в указану кількість символів

Синтаксис

LEFT **(** *character_expression***,** *integer_expression* **)**

Аргументи:

character_expression

Вираз символьного типу. *character_expression* може бути константою, змінною, або атрибутом таблиці.

integer_expression

Позитивне ціле число.

Тип значення, що повертається: Character

Приклад функції LEFT: `LEFT('приклад',3) = 'при'`

Функція RIGHT

Повертає частину символьного рядка, починаючи с вказаного числом *integer_expression* символу від кінця рядка.

Синтаксис

RIGHT **(** *character_expression* **,** *integer_expression* **)**

Аргументи і тип значення, що повертається, аналогічні аргументам і типу функції LEFT.

Приклад функції RIGHT: `RIGHT('контрольна',4) = 'льна'`

Контрольні питання

1. Яка команда Transact-SQL використовується для вибору даних з таблиць бази даних? Навести синтаксис команди з усіма можливими розділами, але не розкриваючи докладно кожний розділ.
2. Для чого призначений розділ SELECT команди Transact-SQL SELECT? Навести синтаксис розділу.
3. Для чого призначений розділ FROM команди Transact-SQL SELECT? Навести синтаксис розділу.
4. Для чого призначений розділ WHERE команди Transact-SQL SELECT? Навести синтаксис розділу.
5. Для чого призначений розділ ORDER BY команди Transact-SQL SELECT? Навести синтаксис розділу.
6. Які є функції Transact-SQL для роботи з датами? Навести синтаксис функцій з поясненням параметрів.

Приклади запитів:

1. Для працівників без вищої освіти, вибрати: прізвище, ім’я, вік, назву посади, зарплату (зарплата розраховується за формулою `post_money*Stavka`). Упорядкувати за спаданням зарплати.

Атрибути: прізвище, ім’я та дата народження (за яким розраховується вік), – належать таблиці `Persona`; атрибут посада – таблиці `Post`; зв'язок між ними можна встановити тільки через таблицю `Worker`: таблиця `Persona` зв’язується з таблицею `Worker` по ключу `Id_man`, таблиця `Worker` зв’язується з таблицею `Post` по ключу `Id_post`. В розділі `WHERE` буде умова фільтру (немає вищої освіти):

```
SELECT Surname as Фамилия, Name1 as Имя,
       YEAR(GETDATE())-YEAR(birthdate) as Возраст,
       Name_post Должность, post_money*Stavka as Зарплата
FROM Persona p join Worker w on p.id_man=w.id_man join
 Post on w.id_post=Post.id_post
WHERE Education !='высшее'
ORDER BY Зарплата DESC
```

	Фамилия	Имя	Возраст	Должность	Зарплата
1	Воробьев	Петр	31	Начальник отдела	3800.0000
2	Стрижов	Олег	34	Водитель	3075.0000
3	Холодова	Лидия	61	Бухгалтер	2400.0000
4	Рыбников	Олег	31	Секретарь	1500.0000
5	Туманова	Евгения	26	Секретарь	1350.0000
6	Сергеев	Андрей	36	Бухгалтер	1200.0000

Рисунок 5 – Результат виконання запиту 1

2. Для працівників з вищою освітою, які працюють не на повну ставку, вибрати: прізвище, ім’я, вік, назву відділу, назву посади, ставку.

Атрибути: прізвище, ім’я та дата народження (за яким розраховується вік), – належать таблиці `Persona`; атрибут посада – таблиці `Post`; зв'язок між ними можна встановити тільки через таблицю `Worker`: таблиця `Persona` зв’язується з таблицею `Worker` по ключу `Id_man`, таблиця `Worker` зв’язується з таблицею `Post` по ключу `Id_post`. Встановимо зв'язок в розділі `FROM`; в розділі `WHERE` будуть умови фільтру (вища освіта та неповна ставка):

```
SELECT Surname as Фамилия, Name1 as Имя,
       Возраст = YEAR(GETDATE())-YEAR(birthdate),
       Name_otd Отдел, Name_post Должность,
       Stavka as Ставка
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post inner join
Otdel ON Otdel.id_otd=w.id_otd
WHERE Education ='высшее' and stavka<1
```

	Фамилия	Имя	Возраст	Отдел	Должность	Ставка
1	Трубин	Антон	36	Технический	Инженер	.75
2	Волков	Тарас	26	Компьютерный	Системный администратор	.50
3	Волков	Тарас	26	Компьютерный	Программист	.85
4	Степная	Татьяна	38	Бухгалтерия	Главний бухгалтер	.85
5	Зайцев	Сергей	36	Компьютерный	Программист	.25
6	Клюквина	Ольга	53	Производственный	Менеджер	.40
7	Зайцева	Алла	42	Производственный	Секретарь	.50
8	Стриженов	Олег	32	Производственный	Водитель	.07
9	Ланская	Карина	25	Технический	Секретарь	.50
10	Ланская	Карина	25	Технический	Инженер	.75
11	Никитин	Сергей	33	Компьютерный	Начальник отдела	.50

Рисунок 6 – Результат виконання запиту 2

Завдання до лабораторної роботи №2

1. Для чоловіків з вищою освітою та жінок з незакінченою вищою освітою вибрати: прізвище, ім’я, освіту, назву посади, зарплату (зарплата розраховується за формулою `money*Stavka`.
2. Для чоловіків одружених або з дітьми та жінок з вищою або незакінченою вищою освітою вибрати: прізвище, ім’я, освіту, сімейний стан, кількість дітей, назву посади, зарплату.
3. Вибрати працівників молодше 35 років, в яких є більше 1 дитини. Вибираються прізвище працівника, ім’я, вік (на кінець року), кількість дітей, назва посади, зарплата.
4. Вибрати працівників, в яких цього року ще не було дня народження (тобто номер місяця їх народження більше номеру місяця поточної дати, або номер місяця їх народження дорівнює номеру місяця поточної дати та номер дня їх народження більше номеру дня поточної дати). Вибираються прізвище працівника, ім’я, вік (на кінець року), дата народження, назва посади.
5. Вибрати працівників віком більше 60 років, та зарплатою більше 3000. Вибираються прізвище працівника та його ініціали (в одному стовпчику), вік, освіта, назва посади, зарплата. Упорядкувати за абеткою працівників, тобто за зростанням значень у першому стовпчику.
6. Вибрати співробітників-жінок, що мають вищу освіту. Вибираються прізвище, ім’я, кількість дітей, сімейний стан, зарплата. Упорядкувати за зростанням кількості дітей; в другу чергу – за убуванням зарплати.

7. Вибрати неодружених співробітників-чоловіків, що народились у другому півріччі. Вибираються прізвище, ім’я, дата народження, освіта, назва посади, зарплата. Упорядкувати за зростанням зарплати.
8. Вибрати працівників молодше 25 років з вищою освітою і зарплатнею більше 4000, які неодружені, або не мають дітей. Вибираються прізвище працівника, ім’я, вік (на кінець року), освіта, сімейний стан, назва посади, зарплата.
9. Для працівників з вищою освітою, не сумісників (працюють як основні працівники), вибрати: прізвище, ім’я, вік, освіту, зарплату).
10. Для працівників, що йдуть у відпустку влітку, вибрати: прізвище з ініціалами, назву відділу, назву посади, освіту, дату початку відпустки.
11. Для працівників, що йдуть у відпустку не влітку, вибрати: прізвище з ініціалами, назву відділу, кількість дітей, дату початку відпустки, кількість днів відпустки.
12. Для працівників, що йдуть у відпустку в другому кварталі, вибрати: прізвище з ініціалами, назву відділу, назву посади, дату закінчення відпустки.

Звіт з лабораторної роботи №2

Звіт з цієї роботи повинен містити відповіді на контрольні запитання, роздруківку вмісту таблиць (Screenshot відкритої таблиці).

Для кожного запиту потрібно навести SQL-код запиту та Screenshot результату запиту.

Якщо для деяких запитів результат виявиться порожнім, тобто запит не вибере ніяких записів з таблиць, щоб відповідали потрібній умові, то треба внести зміни в зміст таблиць. Особливо це стосується таблиці `Persona`.

Лабораторна робота №3 Складні запити до БД

Мета роботи:

1. Отримання практичних навичок створення складних запитів мовою SQL у СКБД MS SQL Server 2000.
2. Знайомство з агрегатними функціями;
3. Отримання практичних навичок створення запитів з групуванням даних мовою SQL у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

1. Функції CAST, CONVERT, CASE.
2. Вкладені запити.
3. Групування даних у запитах

Функції Transact-SQL

SQL Server має велику кількість функцій, повний перелік яких можна отримати у файлі допомоги програми (Help). В цьому розділі наводяться лише функції, які потрібні для виконання варіантів завдань лабораторної роботи. Оскільки запити до БД можна писати різними способами і з використанням різних функцій, то студенти можуть під час виконання лабораторної роботи використовувати функції, які не перелічені в цьому розділі, якщо вони самостійно ознайомились з форматом запису та використання цих функцій. Деякі необхідні в запитах функції вже описані вище (лаб.робота №2).

Функції перетворення типу даних

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

Функції **CAST** і **CONVERT**

Конвертують вираз одного типу даних в інший. Функції виконують аналогічні дії, але CONVERT має трохи більше можливостей.

Синтаксис

CAST **(** *expression* AS *data_type* **)**

CONVERT **(** *data_type* [**(** *length* **)**] **,** *expression* [**,** *style*] **)**

Аргументи:

expression

Будь-який припустимий для синтаксису Microsoft SQL Server вираз.

data_type

Це тип даних, в який потрібно конвертувати *expression.* Це потрібен бути системний тип даних; типи даних, що визначені користувачем, не припустимі.

length

Довжина символьного рядка. Необов'язковий параметр, що використовується при перетворенні у типи: **nchar**, **nvarchar**, **char**, **varchar**, **binary**, або **varbinary.**

style

Це є стиль завдання формату дати/часу під час перетворення типів **datetime** або **smalldatetime** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**), або формат символьного рядка під час перетворення типів **float**, **real**, **money**, або **smallmoney** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**).

У таблиці 3 перелічені основні формати запису дати. Повний перелік форматів запису дати/часу можна отримати у файлі допомоги (Help) програми Query Analyzer SQL Server.

Таблиця 3 – Нумерація стилів формату дати/часу

№ стилю без століття (yy)	№ стилю зі століттям (yyyy)	Вхідне/вихідне значення
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
8	108	hh:mm:ss
14	114	hh:mi:ss:mmm(24h)
	20 або 120	yyyy-mm-dd hh:mi:ss(24h)
	21 або 121	yyyy-mm-dd hh:mi:ss.mmm(24h)

У таблиці 3 третій стовпчик називається "Вхідне/вихідне значення", тобто можна конвертувати символьний рядок у дату/час (вхідне значення символьного рядка для обраного стилю запису дати/часу повинного відповідати формату запису, представленому у 3-му стовпчику таблиці 3), або конвертувати дату/час у символьний рядок – тоді 3-й стовпчик таблиці 3 демонструє, як буде виглядати символьний рядок з записом дати/часу.

Зауваження: Якщо використовується дата без століття, то SQL Server інтерпретує двоцифровий запис року (yy) як рік двадцятого століття, якщо він знаходиться в діапазоні від 50 до 99, і як рік 21 століття – якщо двоцифрове число в діапазоні від 0 до 49.

Таким чином функція:

```
CONVERT(datetime,'15/11/40',3)
```

поверне дату: 15 листопаду 2040 року; а функція:

```
CONVERT(datetime,'07.20.70',1)
```

поверне дату: 20 липня 1970 року.

Для впевненості правильності завдання дати краще завжди використовувати формат дати з століттям.

Для перетворення числових типів в рядкові (символьні) можна також використовувати функцію STR.

Функція **STR**

Повертає символьні дані, конвертовані з числових даних.

Синтаксис

STR **(** *float_expression* [**,** *length* [**,** *decimal*]] **)**

Аргументи:

float_expression

Вираз приблизного числового типу даних з десятковою крапкою.

length

Загальна довжина рядка символів, що містить десяткову крапку, знак числа (для від'ємних чисел), цифри та пробіли (пропуски). За замовчуванням *length* = 10.

decimal

Кількість цифр дробової частини числа, тобто кількість цифр праворуч десяткової крапки.

Тип значення, що повертається

char

Зауваження: Для переводу цілого числа у рядок символів можна використовувати як функцію CONVERT, так і функцію STR, але вони по різному додають пробіли, якщо число потребує менш символів, ніж вказано в форматі переведення його у рядок символів: функція CONVERT, додає пробіли праворуч числа (хвостові пробіли), а функцію STR – ліворуч числа.

Це можна побачити з наступного прикладу, в якому номер поточного місяця (у даному випадку це березень) перетворюється у рядок з 4-х символів спочатку функцією `CONVERT`, потім функцією `STR`. Тому запит:

```
select conv=convert(char(4),MONTH(getdate()))+'/',
       st=STR(MONTH(getdate()),4)+'/'
```

Поверне результат:

	conv	st
1	3 /	3/

Якщо аргумент *length* містить меншу довжину, ніж потрібно для числа, що конвертується у рядок, то функція `CONVERT`, так і функція `STR` повернуть рядок з символів '*', але функція `CONVERT` поверне рядок з одного символу '*', а

функція `STR` поверне рядок з *length* символів ‘*’, як видно з результату наступного запита:

```
select conv=convert(char(4),10000+MONTH(getdate()))+'/',
    st=STR(10000+MONTH(getdate()),4)+'/'
```

Результат запиту:

	conv	st
1	*/	****/

Якщо в функції `STR` є аргумент *decimal* і аргумент *length* містить меншу довжину, ніж потрібно для числа, то число конвертується у рядок з меншим числом цифр дробової частини, ніж вказано в аргументі , і тільки коли аргумент *length* менше цифр цілої частини числа (разом зі знаком числа для від’ємних чисел), то повертається рядок з символів ‘*’. Тобто запит:

```
select max_money=STR((select max(post_money) from post),7,2)
```

поверне:

	max_money
1	9000.00

Запит:

```
select max_money=STR((select max(post_money) from post),6,2)
```

поверне:

	max_money
1	9000.0

Запит:

```
select max_money=STR((select max(post_money) from post),5,2)
```

поверне:

	max_money
1	9000

А запит:

```
select max_money=STR((select max(post_money) from post),3,2)
```

поверне:

	max_money
1	***

Функція **CASE**

Обчислює список умов та повертає результат одного з множини можливих виразів.

Функція CASE має два формати:

- Проста CASE функція порівнює значення вхідного виразу з набором простих виразів і визначає результат.
- Пошукова CASE функція обчислює набор виразів логічного типу для визначення результату.

Обидва формати підтримують необов'язковий аргумент ELSE.

Синтаксис

Проста CASE функція:

```
CASE input_expression
   WHEN when_expression THEN result_expression
      [ ...n ]
   [       ELSE else_result_expression      ]
END
```

Пошукова CASE функція:

```
CASE
   WHEN Boolean_expression THEN result_expression
      [ ...n ]
   [       ELSE else_result_expression      ]
END
```

Аргументи:

input_expression

Будь-який припустимий вираз Microsoft SQL Server.

WHEN *when_expression*

Для простої CASE функції *when_expression* повинен бути виразом, що сумісний за порівнянням з виразом *input_expression*.

[...*n*]

Це означає, що конструкцій WHEN *when_expression* THEN *result_expression* (або WHEN *Boolean_expression* THEN *result_expression*) може бути декілька.

THEN *result_expression*

result_expression – це вираз, значення якого повертається, коли значення *input_expression* дорівнює значенню *when_expression* (або коли значення *Boolean_expression* дорівнює TRUE). *result_expression* – будь-який припустимий вираз Microsoft SQL Server.

ELSE *else_result_expression*

else_result_expression – це вираз, значення якого повертається, якщо для простої CASE функції немає жодного *when_expression*, значення якого дорівнює значенню *input_expression*; або якщо для пошукової CASE функції немає жодного *Boolean_expression*, значення якого дорівнює TRUE. Якщо цей аргумент відсутній, але жодна з гілок WHEN не задовольняє умові пошуку, CASE функція повертає NULL.

WHEN *Boolean_expression*

Для пошукової CASE функції. *Boolean_expression* – довільний припустимий вираз логічного типу.

Тип значення, що повертається

Аналогічний типу виразів *result_expression* ***та*** *else_result_expression.*

Приклад використання функцій CASE I CONVERT.

Нехай є БД визначних історичних осіб минулого і сьогодення. Нехай в таблиці `Persona` є атрибути: `Surname` (призвіще, тип varchar), `Name1` (ім'я, тип varchar), `BirthDate` (дата народження, тип datetime), `DeathDate` (дата смерті, тип datetime). Потрібно вивести інформацію о кожній особі у наступному форматі форматі:

```
Олександр Пушкін            17/10/1799 – 25/05/1837
Леонід Кравчук              25/04/1948 –
```

В цьому прикладі атрибут `DeathDate` може мати порожні значення (для живих осіб). Для перевірки, чи має цей атрибут порожнє значення, і виведення або пропуску, або рядка з датою, використовується функція CASE. Функція CONVERT використовується для переведення у символьний рядок дат (народження і смерті). Таким чином:

```
SELECT Особа=Surname+' '+Name1, "Дати життя"=
    CONVERT(char(10), BirthDate,103)+ ' - '+
    CASE
        WHEN DeathDate IS NULL THEN ''
        ELSE CONVERT(char(10), DeathDate,103)
        END
FROM Persona
```

Складна вибірка даних в Query Analyzer MS SQL Server 2000

Основні теоретичні відомості

Агрегатні функції

Крім пошуку інформації, яка зберігається в таблицях, за допомогою запитів також можна отримувати деяку статистичну інформацію: максимальне значення деякого атрибута (або вираза з використанням атрибута), мінімальне, середнє значення. Ці можливості надають агрегатні функції. До агрегатних функцій відносяться функції COUNT, SUM, AVG, MAX, MIN. Зазвичай агрегатні функції використовуються у запитах з групуванням і обчислюють кількість записів в групі, суму деякого атрибута для групи, середнє, мінімальне або максимальне значення для групи деякого атрибута. Агрегатні функції можна використовувати не тільки для одного атрибута, но і для виразу з атрибутом, наприклад, для розрахунку середнього (мінімального, максимального) віку групи осіб за їх датою народження.

Розділ GROUP BY

Вказує групи, для яких розраховуються значення за допомогою агрегатних функцій: суму, кількість записів, середнє значення, максимальне значення, мінімальне значення. Якщо є розділ GROUP BY, в розділі SELECT крім виразів з агрегатними функціями можуть бути тільки стовпці, перелічені в розділі GROUP BY. Якщо в команді SELECT є розділ GROUP BY і немає розділу ORDER BY, упорядкування рядків результуючого набору проводиться за пунктами групування.

Синтаксис команди:

GROUP BY [ALL] *group_by_expression* [**,***...n*]

Параметри:

ALL

Вказує, що всі групи повинні бути представлені у результуючому наборі, навіть якщо в них немає жодного запису. Коли вказаний параметр ALL, для групи без записів (для якої умова фільтру в розділі WHERE не знайде жодного запису) функція COUNT повертає значення арифметичного нуля, інші агрегатні функції (MAX, MIN, SUM, AVG) повертають значення NULL. Якщо параметр не вказаний, група без записів не включається в результат запиту.

group_by_expression

Вказує вираз, за яким групуються запити. Найчастіше вираз складається з імені стовпця, за яким проводиться групування, але це може бути неагрегатний вираз з використанням атрибутів стовпців. Якщо в розділі SELECT для цього виразу був наданий псевдонім стовпця, то в розділі GROUP BY повинен використовуватись вираз, а не псевдонім.

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

Групування, як і упорядкування, може бути багаторівневим.

Приклади використання розділу GROUP BY.

Вирахувати кількість будинків на кожній вулиці, при цьому включити в результат вулиці, на яких немає будинків (якщо в цьому запиті описати зв'язок між таблицями в розділі FROM, то вулиці без будинків не будуть вибрані не зважаючи на параметр ALL):

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
```

Вирахувати загальну кількість кімнат по всіх квартирах будинку, та середню площу квартири для кожного будинку на кожній вулиці, при цьому виключити з результату будинки, в яких немає квартир:

```
SELECT name_street+' д.'+h.number AS house_N,
          SUM(rooms) as kol_vo, AVG(all_square) as AVG_Square
FROM street s, house h,flat f
WHERE s.id_street=h.id_street AND h.id_street=f.id_street AND
      f.number=h.number
GROUP BY name_street+' д.'+h.number
```

Розділ HAVING

Вказує умову пошуку для груп або агрегатних функцій.

Синтаксис команди:

HAVING < search_condition >

Параметр:

< search_condition >

Вказує умову пошуку для груп з використанням агрегатних функцій. Наприклад, якщо в останньому запиті потрібно вибрати тільки будинки з загальною кількістю кімнат не менше 20, то після розділу: GROUP BY … – буде розділ HAVING:

```
HAVING COUNT(*)>=20
```

Приклад: вирахувати кількість будинків на кожній вулиці, при цьому включити в результат тільки ті вулиці, на яких більше 5 будинків:

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
HAVING COUNT(*)>=5
```

В одному запиті можуть бути як розділ WHERE, так і розділ HAVING. Звичайно в розділі WHERE записуються умови пошуку даних в таблицях бази даних, а в розділі HAVING – умови пошуку груп даних, отриманих в результату виконання запиту.

Звіт по лабораторній роботі повинен містити:

- Відповіді на контрольні питання.
- Для кожного пункту завдання: текст завдання, SQL-код виконаного завдання, копію екрану (Screenshot) з результатом виконання завдання.

Перелік запитань до лабораторної роботи №3

1. Які є функції перетворення типів у мові Transact-SQL? Навести синтаксис функцій з описом параметрів.
2. Для чого використовується функція Transact-SQL CASE? Навести можливі формати синтаксису функції та пояснити параметри.

3. Для чого призначений розділ GROUP BY команди Transact-SQL SELECT? Наведіть синтаксис.
4. Для чого призначений розділ HAVING команди Transact-SQL SELECT? Наведіть синтаксис.
5. Наведіть синтаксис та поясніть параметри всіх агрегатних функцій.

Приклади запитів

Запити з використанням функцій CASE та CONVERT

3. Для працівників відділу «Адміністрація», які ще не пішли у відпустку поточного року, вибрати: прізвище з ініціалами, назву посади, сезон виходу у відпустку: зима, весна, літо, осінь, – і дату виходу у відпустку у форматі dd/mm/yyyy:

```
SELECT RTRIM(Surname)+' '+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.' as
[Фамилия И.О.],     Name_post Должность,  [Сезон отпуска]= case
     when month(date_vac) between 3 and 4 then 'весна'
     when month(date_vac) between 6 and 8 then 'лето'
     when month(date_vac) between 9 and 11 then 'осень'
     else 'зима' end,
[Начало отпуска]=convert(char(10),date_vac,103)
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
    INNER JOIN vacation v ON p.id_man=v.id_man
    INNER JOIN Otdel ON otdel.id_otd=w.id_otd
WHERE  name_otd='Администрация' and year_vac=year(getdate())
       and date_vac>getdate() and main_work=1
```

Результат виконання команди видно на рисунку 7.

	Фамилия И.О.	Должность	Сезон отпуска	Начало отпуска
1	Иванов И.И.	Директор	лето	15/07/2012
2	Сорокина С.П.	Секретарь	лето	01/06/2012
3	Иванова И.И.	Зам.директора	лето	25/07/2012
4	Воробьева С.П.	Системный администратор	осень	02/10/2012
5	Иванникова О.И.	Прибиральниця	весна	25/04/2012

Рисунок 7 – Результат команди SELECT прикладу 3

Використання вкладених запитів

4. Для жінок, які не працюють за сумісництвом (не входять в множину працівників-сумісників), вибрати: прізвище з ініціалами, кількість дітей, освіту, відділ, зарплату (зарплата розраховується за формулою `post_money*Stavka`).

```
select ФИО=surname+' '+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.',
 "Кол-во детей"=Children, Образование=Education,
 Отдел=Name_otd, Зарплата=Post_money*stavka
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
INNER JOIN Otdel ON w.Id_Otd=Otdel.Id_otd
WHERE Sex='ж' and w.Id_man NOT IN
(SELECT Id_man FROM Worker WHERE Main_work=0)
```

Результат виконання команди видно на рисунку 8.

	ФИО	Кол-во детей	Образование	Отдел	Зарплата
1	Петрова О.И.	2	высшее	Бухгалтерия	6000.0000
2	Зимина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
3	Сидорова И.С.	0	н/высшее	Маркетинга	2625.0000
4	Сорокина С.П.	0	среднее	Администрация	2500.0000
5	Иванова И.И.	1	высшее	Администрация	8000.0000
6	Замятина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
7	Антонюк О.П.	2	высшее	Маркетинга	2625.0000
8	Лисицына А.Л.	0	н/высшее	Маркетинга	3500.0000
9	Алексеев О.П.	2	высшее	Маркетинга	2625.0000
10	Ольгина С.П.	0	среднее	Производства	4000.0000
11	Хрущ И.И.	1	высшее	Производства	5000.0000
12	Соколова Л.Л.	0	н/высшее	Производства	4000.0000
13	Медведь И.С.	0	н/высшее	Маркетинга	3500.0000
14	Семенов О.П.	0	высшее	Маркетинга	4000.0000
15	Хрущева С.П.	2	среднее	Производства	3000.0000

Рисунок 6 – Результат команди SELECT прикладу 4

Приклади запитів з групуванням

5. Для кожного відділу розрахувати кількість чоловіків та кількість жінок. Вибрати: назву відділу, стать, кількість осіб цієї статі в цьому відділі.

```
SELECT Name_otd as Отдел, sex as Пол, count(*)  Количество
FROM Persona p INNER JOIN worker w ON p.id_man=w.id_man
     INNER JOIN Otdel ON w.Id_Otd=Otdel.Id_otd
GROUP BY Name_otd, sex
```

В цьому запиті є два рівня групування: за статтю та за відділом. Якщо поміняти місцями атрибути в розділі `GROUP BY` (тобто написати `GROUP BY sex, Name_otd`), то розрахунки залишаться тими самими, але зміниться порядок виводу записів в результаті запиту: або для кожного відділу виводиться спочатку кількість жінок, потім – кількість чоловіків; або спочатку виводиться кількість жінок для кожного відділу, а потім – кількість чоловіків

для кожного відділу). Спробуйте виконати цей запит з обома порядками групування та отримайте результат.

6. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальну зарплатню з урахуванням ставки працівника. Включаються відділи, де є працівники з дітьми.

```
SELECT Отдел=Name_otd, [Количество сотрудников]= count(*),
  AVG(YEAR(GETDATE())-YEAR(birthdate)) as [Средний Возраст],
STR(MIN(post_money*Stavka),7,2)) as MIN_Зарплата
FROM Persona p, Worker w , Post, otdel
WHERE p.id_man=w.id_man AND w.id_post=Post.id_post AND
  Otdel.id_otd=w.id_otd AND w.id_otd IN
(SELECT id_otd FROM Persona p join Worker w ON p.id_man=w.id_man
     WHERE children>0)
   GROUP BY Name_otd
```

Результат виконання команди видно на рисунку 9.

	Отдел	Количество сотрудников	Средний Возраст	MIN_Зарплата
1	Администрация	8	41	2000.00
2	Бухгалтерия	8	35	1000.00
3	Маркетинга	15	36	1000.00
4	Производства	15	38	1100.00

Рисунок 9 – Результат команди SELECT прикладу 6

Завдання для виконання лабораторної роботи

1. Для працівників, що йдуть у відпустку в другому кварталі, вибрати: прізвище з ініціалами, назву відділу, назву посади, дату початку відпустки у форматі dd.mm.yyyy, кількість днів відпустки.
2. Для працівників, що йдуть у відпустку восени, вибрати: прізвище з ініціалами, назву відділу, назву посади, дату початку відпустки dd/mm/yy, ставку.
3. Для основних працівників (не сумісників) з вищою освітою вибрати: прізвище з ініціалами, вік (на кінець року), вікову групу, назву посади, зарплату. Вікову групу визначаємо наступним чином:
   - менше 35 років – молодіж;
   - від 35 до 60 років – зрілі;
   - більше 60 – похилі (пожилые).
4. Для працівників, що прийняті на роботу раніше минулого року на повну ставку (ставка = 1), вибрати: прізвище з ініціалами, вік (на кінець року), освіту, назву відділу, назву посади, стаж роботи (повних років). Стаж

роботи (повних років) вираховується за допомогою функції CASE, з урахуванням того, якого місяця і числа робітник був прийнятий на роботу.

5. Для працівників без вищої освіти, які не працюють за сумісництвом (не входять в множину працівників-сумісників), вибрати: прізвище з ініціалами, назву посади, освіту, відділ, зарплату (зарплата розраховується за формулою `money*Stavka`).
6. Для працівників-чоловіків, що працюють на посадах, на яких немає сумісників (не входять в множину посад працівників-сумісників), вибрати: прізвище з ініціалами, назву посади, освіту, відділ, стаж роботи (розраховується за формулою: від поточного року відняти рік прийому на роботу).
7. Для працівників, що працюють як сумісники (входять в множину працівників-сумісників) і як основні працівники, вибрати: прізвище з ініціалами, назву посади по основній роботі, освіту, відділ (де працює по основній роботі), оклад (по основній роботі). (Основна робота буде у якості умови фільтру у основному запиті; у вкладеному запиті буде розраховуватися множина ідентифікаторів працівників, що працюють за сумісництвом).
8. Для працівників з дітьми, що не працюють як основні працівникі (не входять в множину основних працівників), вибрати: прізвище з ініціалами, назву посади, кількість дітей, відділ, зарплату (зарплата розраховується за формулою `money*Stavka`).
9. Для працівників з відділів, в яких немає сумісників, вибрати: прізвище з ініціалами, вік, назву посади, відділ, освіту, оклад.
10. Для працівників з відділів, в яких немає жінок, вибрати: прізвище з ініціалами, назву посади, відділ, освіту, сімейний стан.

Завдання

В запитах з номерами 6-10 потрібно використати вкладений запит, якій розраховує деяку множину номерів відділів. Основний запит обирає записи, для яких номер відділу входить (або не входить) у розраховану множину номерів відділів. Як і в лабораторній роботі №4 слід продумати логіку при написанні вкладеного запиту. Наприклад, не є одноковими множинами множини номерів відділів, в яких немає жінок, та номерів відділів, в яких є чоловіки (тому що є відділи, в яких працюють й чоловіки, й жінки).

Запити для виконання:

1. Розрахувати кількість чоловіків та кількість жінок, що працюють на кожній посаді.
2. Розрахувати кількість осіб з кожним рівнем освіти, що працюють в кожному відділі (Тобто, скільки осіб є з вищою освітою, скільки – з незакінченою вищою, і т.д. Групувати треба по атрибуту `education`).
3. Розрахувати кількість осіб з дітьми, що працюють в кожному відділі. Залишити тільки відділи, де таких осіб більше 2.
4. Розрахувати кількість сумісників в кожному відділі. Залишити тільки відділи, де таких осіб більше 3.
5. Розрахувати середню та максимальну зарплатню (з урахуванням ставки) в кожному відділі. Залишити тільки відділи з середньою зарплатнею більше 2000, в яких працює не менше 6 осіб.
6. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальний оклад. Включаються відділи, де немає працівників з дітьми.
7. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, максимальний вік працівників, середню зарплату (з урахуванням ставки). Включаються відділи, де приймали працівників у 2010 році.
8. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, мінімальний вік працівників, середню зарплату (з урахуванням ставки). Включаються відділи, де є працівники-жінки.
9. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальну ставку. Включаються відділи, в яких не приймали на роботу працівників поточного року.
10. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, максимальний вік працівників, середній оклад. Включаються відділи, де немає працівників з начальною освітою.

Лабораторна робота №4 Маніпулювання даними

Мета роботи:

1. Знайомство з мовою DML, як частиною мови Transact-SQL;
2. Отримання практичних навичок змінення даних в таблицях та усунення записів з таблиць у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

1. Команди UPDATE.
2. Команди DELETE.

Основні теоретичні відомості

Редагування даних в таблицях

Для змінення значення полів записів таблиць використовується команда Transact-SQL UPDATE, яку найчастіше використовують у наступному форматі:

UPDATE *table_name* SET *column_name* = { *expression* | DEFAULT | NULL } [**,***...n*]
[FROM < table_source >]
[WHERE < search_condition >]

Параметри команди:

table_name

Ім'я таблиці, в якій змінюють значення атрибутів.

column_name

Ім'я атрибуту (поля таблиці), значення якого змінюється.

expression

Вираз, значення якого привласнюється полю *column_name*. Тип виразу *expression* повинен бути сумісним по присвоюванню з типом даних *column_name.* В якості виразу для вирахування значення атрибуту може використовуватись запит, тобто команда SELECT (дивись розділ "Вибірка даних в Query Analyzer MS SQL Server 2000"). В даному випадку такий запит повинен повертати таблицю з одного значення (один стовпець, один рядок).

DEFAULT | NULL

Ці параметри розглядаються у наступному семестрі.

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

Одною командою можна змінити значення декільком атрибутам (полям таблиці); тобто конструкцію:

column_name = *expression* |

можна записувати кілька разів, відокремлюючи привласнення кожного атрибута комою.

FROM < table_source >

Вказує, що використовуються інші таблиці (базові таблиці, представлення, віртуальні таблиці) для визначення кортежів таблиці *table_name*, в яких будуть виконуватись зазначені зміни полів. Ця конструкція розглядалася вище, у розділі "Вибірка даних в Query Analyzer MS SQL Server 2000" для команди SELECT.

WHERE < search_condition >

Вказує умови пошуку рядків (кортежів) таблиці, в яких будуть змінюватись значення вказаних атрибутів. В умові після ключового слова WHERE можуть використовуватись атрибути той таблиці, для якої змінюються значення полів, і зв'язаних з нею таблиць, вказаних в розділі FROM < table_source >. Зв'язки між таблицями можна задавати і в розділі FROM, і в розділі WHERE (дивись команду SELECT).

Приклади використання команди UPDATE.

Для демонстрації команд редагування даних та видалення записів з бази даних, розглянемо схематичну базу даних житлового фонду міста (`City`) з трьох таблиць: `Street` (Вулиця), `House` (Будинок), `Flat` (Квартира). В таблиці `Street` буде лише два атрибута (стовпця): назва вулиці (`name_street`) та ідентифікатор вулиці в БД (`id_street`). В таблиці `House` будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці (`number`), кількість поверхів в будинку (`count_floor`), кількість квартир (`count_flat`). Первинний ключ таблиці `House` буде складовим: складається з атрибутів `id_street` та `number`. В таблиці `Flat` будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці, номер квартири (`flat_number`), кількість кімнат у квартирі (`rooms`), загальна площа квартири (`all_square`).

Розглянемо декілька прикладів змінення даних в БД `City` командою UPDATE.

1. Змінити назву вулиці Пушкінська на Лермонтівська:

```
UPDATE Street SET name_street='Лермонтівська'
WHERE name_street='Пушкінська'
```

2. Привласнити значення «три» для кількості кімнат у квартирі номер 4 в будинку №1 на вулиці Дерібасівській:

```
UPDATE Flat SET rooms=3 WHERE flat_number=4 and number='1'
and  id_street=
            (SELECT id_street FROM  Street
             WHERE name_street='Дерібасовська')
```

Видалення записів в таблиці

Для видалення записів з таблиці використовується команда Transact-SQL DELETE, яку найчастіше використовують у наступному форматі:

DELETE [FROM] *table_name*
[FROM < table_source >]
[WHERE < search_condition >]

Параметри команди:

table_name

Ім'я таблиці, в якій видаляються записи. Ключове слово FROM перед іменем таблиці можна пропускати. Тобто команди:

```
DELETE Flat
```

та

```
DELETE FROM Flat
```

є ідентичними. До речі, вказані команди повністю видалять всі записи з вказаної таблиці (`Flat`). Щоб видалити не всі записи таблиці, а лише деякі, які відповідають заданій умові, потрібно використовувати параметр WHERE < search_condition >. Цей параметр, а також параметр FROM < table_source > є аналогічними цим же параметрам розглянутої команди UPDATE, і більш докладно розглядаються в команді SELECT.

Якщо командою DELETE видаляють записи з таблиці, первинний ключ якої використовується як зовнішній ключ іншої (підлеглої) таблиці, то SQL Server спочатку перевіряє, чи є в підлеглій таблиці рядки, зовнішній ключ яких дорівнює первинному ключу в тих рядках основної таблиці, що видаляються. Якщо такі рядки є, то може бути два варіанта подій:

- якщо для підлеглої таблиці в описі зовнішнього ключа вказане обмеження цілісності CASCADE, то разом з рядками основної таблиці будуть видалені всі зв'язані з ними зовнішнім ключем рядки підлеглої таблиці;
- якщо для підлеглої таблиці в описі зовнішнього ключа вказане обмеження цілісності NO ACTION (або не вказане нічого), то SQL Server заборонить видаляти рядки основної таблиці, які зв'язані з рядками підлеглої таблиці.

Наприклад, для бази даних `City`, що розглядається у прикладах вище, для будинку №1 по вулиці Дерібасівській в базу даних записані 3 квартири. Нехай каскадного видалення записів не передбачено. Тому наступна команда (видалення вказаного будинку):

```
DELETE house WHERE number='1' AND id_street=
(SELECT id_street FROM street WHERE
 name_street='Дерібасовська')
```

не буде виконана, і вона викличе помилку:

```
DELETE statement conflicted with TABLE REFERENCE constraint
'FK__flat__7F60ED59'. The conflict occurred in database 'city',
table 'flat'.
```

База даних `Personal` не має каскадного видалення даних, тому при спробі, наприклад, усунути з таблиці `Post` посаду, до якої в базі даних приписані працівники (в таблиці `Worker`), SQL Server заборонить видаляти рядки таблиці `Post` і виведе повідомлення про помилку.

Звіт по лабораторній роботі повинен містити:

- Відповіді на контрольні питання.
- Текст завдання, SQL-код виконаного завдання, копію екрану (Screenshot) з результатом виконання завдання.

Перелік запитань до лабораторної роботи №4

1. Яка команда Transact-SQL використовується для видалення записів з таблиці? Наведіть синтаксис команди с поясненням її параметрів.
2. Яка команда Transact-SQL використовується для змінення записів в таблиці? Наведіть синтаксис команди с поясненням її параметрів.

Приклади використання команд UPDATE і DELETE до БД Personal:

1. Працівника з прізвищем Ільїн, який працює в відділу Маркетингу на посаді економіста, перевести на посаду менеджера.
2. Всіх працівників, які працюють як основні працівники, і прийняті на роботу до 2008 року, перевести на повну ставку (тобто 1).
3. Звільнити (усунути з таблиці Worker) усіх сумісників відділу виробництва.
4. Звільнити всіх працівників, яким на кінець року буде не менше 61 року.

Перші два пункти завдання виконуються командою UPDATE, останні два – командою DELETE.

1. Працівника з прізвищем Ільїн, який працює в відділу Маркетингу на посаді економіста, перевести на посаду менеджера.

```
UPDATE Worker SET Id_post=
(SELECT Id_post FROM Post WHERE Name_post='менеджер')
WHERE Id_man IN
(SELECT Id_man FROM Persona WHERE Surname='Ильин')
```

```
AND Id_post=
(SELECT Id_post FROM Post WHERE Name_post='экономист')
AND Id_otdel=
(SELECT Id_otdel FROM Otdel WHERE Name_otdel='Маркетинга')
```

Як видно з наведеного SQL-коду, в команді не використовувалась необов'язкова опція команди FROM (в якій перелічуються додаткові таблиці для пошуку рядків, поля в яких змінюються). Замість цього використані 3 вкладені запити, які локалізують необхідний рядок таблиці `Worker`.

Результат виконання команди UPDATE видно з порівняння рисунків 8 і 9.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	.75
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50
11	9	3	7	2008-11-23 00:00:00.000	1	1.00
12	10	1	4	2008-09-11 00:00:00.000	1	1.00

Рисунок 8 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50
11	9	3	8	2008-11-23 00:00:00.000	1	1.00

Рисунок 9 – Зміна даних в таблиці `Worker` після команди UPDATE

2. Всіх працівників, які працюють як основні працівники, і прийняті на роботу до 2008 року, перевести на повну ставку.

```
UPDATE Worker SET stavka=1
WHERE stavka<1 AND Main_work=1 AND YEAR(date_work)<2008
```

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	1	1	1	2000-01-10 00:00:00.000	1	1.00
2	2	2	5	2000-01-20 00:00:00.000	1	1.00
3	3	4	10	2009-12-15 00:00:00.000	0	.50
4	4	2	6	2010-05-11 00:00:00.000	1	1.00
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	.75
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50

Рисунок 10 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	1	1	1	2000-01-10 00:00:00.000	1	1.00
2	2	2	5	2000-01-20 00:00:00.000	1	1.00
3	3	4	10	2009-12-15 00:00:00.000	0	.50
4	4	2	6	2010-05-11 00:00:00.000	1	1.00
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	1.00
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50

Рисунок 11 – Зміна даних в таблиці `Worker` після команди UPDATE

3. Звільнити (усунути з таблиці Worker) усіх сумісників відділу виробництва.

```
DELETE Worker WHERE Main_work=0 AND Id_otdel=
(SELECT id_otdel FROM Otdel WHERE Name_otdel='Производства')
```

Як видно з порівняння рисунків 12 і 13, два записи з таблиці `Worker` видалились.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	3	4	10	2009-12-15 00:00:00.000	0	.50
2	7	4	10	2003-11-12 00:00:00.000	1	1.00
3	8	4	9	2006-08-18 00:00:00.000	1	1.00
4	16	4	12	2009-12-15 00:00:00.000	1	1.00
5	19	4	9	2006-08-18 00:00:00.000	1	1.00
6	23	4	4	2006-09-11 00:00:00.000	1	1.00
7	25	4	10	2010-05-11 00:00:00.000	1	1.00
8	26	4	13	2010-12-25 00:00:00.000	0	.50
9	29	4	9	2006-08-18 00:00:00.000	1	1.00
10	30	4	10	2003-11-12 00:00:00.000	1	1.00
11	31	4	14	2008-05-10 00:00:00.000	1	1.00
12	32	4	10	2010-03-12 00:00:00.000	1	1.00
13	33	4	13	2009-12-15 00:00:00.000	1	1.00
14	34	4	10	2006-09-11 00:00:00.000	1	1.00
15	40	4	9	2006-08-18 00:00:00.000	1	1.00

Рисунок 12 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	7	4	10	2003-11-12 00:00:00.000	1	1.00
2	8	4	9	2006-08-18 00:00:00.000	1	1.00
3	16	4	12	2009-12-15 00:00:00.000	1	1.00
4	19	4	9	2006-08-18 00:00:00.000	1	1.00
5	23	4	4	2006-09-11 00:00:00.000	1	1.00
6	25	4	10	2010-05-11 00:00:00.000	1	1.00
7	29	4	9	2006-08-18 00:00:00.000	1	1.00
8	30	4	10	2003-11-12 00:00:00.000	1	1.00
9	31	4	14	2008-05-10 00:00:00.000	1	1.00
10	32	4	10	2010-03-12 00:00:00.000	1	1.00
11	33	4	13	2009-12-15 00:00:00.000	1	1.00
12	34	4	10	2006-09-11 00:00:00.000	1	1.00
13	40	4	9	2006-08-18 00:00:00.000	1	1.00

Рисунок 13 – Видалення записів з таблиці
(звільнення працівників з номерами 3 і 26)

4. Звільнити всіх працівників, яким на кінець року буде не менше 61 року.

```
DELETE Worker WHERE Id_man IN
(SELECT Id_man FROM Persona
WHERE YEAR(GETDATE())-YEAR(birthdate)>=61)
```

Як видно з порівняння рисунків 14 і 15, два записи з таблиці `Worker` видалились.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	7	4	10	2003-11-12 00:00:00.000	1	1.00
2	8	4	9	2006-08-18 00:00:00.000	1	1.00
3	16	4	12	2009-12-15 00:00:00.000	1	1.00
4	19	4	9	2006-08-18 00:00:00.000	1	1.00
5	23	4	4	2006-09-11 00:00:00.000	1	1.00
6	25	4	10	2010-05-11 00:00:00.000	1	1.00
7	29	4	9	2006-08-18 00:00:00.000	1	1.00
8	30	4	10	2003-11-12 00:00:00.000	1	1.00
9	31	4	14	2008-05-10 00:00:00.000	1	1.00
10	32	4	10	2010-03-12 00:00:00.000	1	1.00
11	33	4	13	2009-12-15 00:00:00.000	1	1.00
12	34	4	10	2006-09-11 00:00:00.000	1	1.00
13	40	4	9	2006-08-18 00:00:00.000	1	1.00
14	39	3	8	2002-02-11 00:00:00.000	1	1.00
15	37	3	7	2003-11-12 00:00:00.000	1	1.00
16	38	3	7	2008-11-23 00:00:00.000	1	1.00
17	35	3	8	2004-01-20 00:00:00.000	1	1.00
18	36	3	8	2010-05-11 00:00:00.000	1	1.00

Рисунок 14 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	8	4	9	2006-08-18 00:00:00.000	1	1.00
2	16	4	12	2009-12-15 00:00:00.000	1	1.00
3	19	4	9	2006-08-18 00:00:00.000	1	1.00
4	23	4	4	2006-09-11 00:00:00.000	1	1.00
5	25	4	10	2010-05-11 00:00:00.000	1	1.00
6	29	4	9	2006-08-18 00:00:00.000	1	1.00
7	30	4	10	2003-11-12 00:00:00.000	1	1.00
8	31	4	14	2008-05-10 00:00:00.000	1	1.00
9	32	4	10	2010-03-12 00:00:00.000	1	1.00
10	33	4	13	2009-12-15 00:00:00.000	1	1.00
11	34	4	10	2006-09-11 00:00:00.000	1	1.00
12	40	4	9	2006-08-18 00:00:00.000	1	1.00
13	39	3	8	2002-02-11 00:00:00.000	1	1.00
14	35	3	8	2004-01-20 00:00:00.000	1	1.00
15	36	3	8	2010-05-11 00:00:00.000	1	1.00

Рисунок 15 – Видалення записів з таблиці
(звільнення працівників з номерами 7 і 38)

Виконання лабораторної роботи

Оскільки всі студенти працюють з одною базою даних, то після виконання одним студентом команди на змінення або усунення записів в таблиці, інші студенти не зможуть виконати ті самі дії. Тому викладач вирішує,

у якому порядку який студент виконує команди (наприклад, з першої до останньої підряд; з другої до останньої, а потім – першу; і т.д.). Після виконання кожним студентом одної команди (UPDATE або DELETE) викладач видаляє всі записи з таблиці `Worker` і потім з файлу вводить знову всі попередні записи в цю таблицю.

Запити на оновлення:

1. Для працівників молодше 40 років, що працюють по основній роботі не на повну ставку і мають оклад менше 3000, встановити повну ставку.
2. Для працівників з вищою освітою, що працюють за сумісництвом на ставку менше 0.5 і мають оклад менше 2500, встановити ставку 0.5.
3. Для неодружених працівників з дітьми, що працюють по основній роботі на ставку менше 0.7 і мають оклад менше 2800, додати 0.15 частину до ставки.
4. Для працівників відділів «Комп'ютерний» та «Бухгалтерія», що мають вищу освіту та працюють по основній роботі на ставку менше 0.8, додати 0.1 частину до ставки.
5. Для працівників молодше 35 років, що мають стаж більше 10 років, працюють не на повну ставку і мають оклад менше 3000, встановити повну ставку.
6. Для працівників молодше 35 років, що працюють на посадах «Системний адміністратор» або «Програміст» по основній роботі на ставку менше 0.75, додати 0.2 частини до ставки.

Запити на усунення:

1. Звільнити (видалити з таблиці Worker) сумісників, які старші за 65 років та працюють на посадах «Директор» або «Головний бухгалтер».
2. Звільнити (видалити з таблиці Worker) працівників зі стажем роботи менше 5 років, які старші за 60 років та працюють на посадах з окладом більше 4000.
3. Звільнити (видалити з таблиці Worker) сумісників без вищої освіти, що працюють на посадах з окладом більше 3500, якщо вони працюють не в відділі виробництва.
4. Звільнити (видалити з таблиці Worker) основних працівників з рівнем освіти «Середня» з посад, які вони займають як сумісники.
5. Звільнити (видалити з таблиці Worker) працівників зі стажем роботи менше 4 років, які старші за 65 років та працюють на посадах з кількістю ставок (`count_post`) у відділі не більше 2.
6. Звільнити (видалити з таблиці Worker) сумісників з рівнем освіти «Середня», що працюють на посадах «Бухгалтер» або «Системний адміністратор».

Лабораторна робота №3 Складні запити до БД

Мета роботи:

1. Отримання практичних навичок створення складних запитів мовою SQL у СКБД MS SQL Server 2000.
2. Знайомство з агрегатними функціями;
3. Отримання практичних навичок створення запитів з групуванням даних мовою SQL у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

4. Функції CAST, CONVERT, CASE.
5. Вкладені запити.
6. Групування даних у запитах

Функції Transact-SQL

SQL Server має велику кількість функцій, повний перелік яких можна отримати у файлі допомоги програми (Help). В цьому розділі наводяться лише функції, які потрібні для виконання варіантів завдань лабораторної роботи. Оскільки запити до БД можна писати різними способами і з використанням різних функцій, то студенти можуть під час виконання лабораторної роботи використовувати функції, які не перелічені в цьому розділі, якщо вони самостійно ознайомились з форматом запису та використання цих функцій. Деякі необхідні в запитах функції вже описані вище (лаб.робота №2).

Функції перетворення типу даних

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

Функції **CAST** і **CONVERT**

Конвертують вираз одного типу даних в інший. Функції виконують аналогічні дії, але CONVERT має трохи більше можливостей.

Синтаксис

CAST **(** *expression* AS *data_type* **)**

CONVERT **(** *data_type* [**(** *length* **)**] **,** *expression* [**,** *style*] **)**

Аргументи:

expression

Будь-який припустимий для синтаксису Microsoft SQL Server вираз.

data_type

Це тип даних, в який потрібно конвертувати *expression.* Це потрібен бути системний тип даних; типи даних, що визначені користувачем, не припустимі.

length

Довжина символьного рядка. Необов'язковий параметр, що використовується при перетворенні у типи: **nchar**, **nvarchar**, **char**, **varchar**, **binary**, або **varbinary.**

style

Це є стиль завдання формату дати/часу під час перетворення типів **datetime** або **smalldatetime** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**), або формат символьного рядка під час перетворення типів **float**, **real**, **money**, або **smallmoney** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**).

У таблиці 3 перелічені основні формати запису дати. Повний перелік форматів запису дати/часу можна отримати у файлі допомоги (Help) програми Query Analyzer SQL Server.

Таблиця 3 – Нумерація стилів формату дати/часу

№ стилю без століття (yy)	**№ стилю зі століттям (yyyy)**	**Вхідне/вихідне значення**
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
8	108	hh:mm:ss
14	114	hh:mi:ss:mmm(24h)
	20 або 120	yyyy-mm-dd hh:mi:ss(24h)
	21 або 121	yyyy-mm-dd hh:mi:ss.mmm(24h)

У таблиці 3 третій стовпчик називається "Вхідне/вихідне значення", тобто можна конвертувати символьний рядок у дату/час (вхідне значення символьного рядка для обраного стилю запису дати/часу повинного відповідати формату запису, представленому у 3-му стовпчику таблиці 3), або конвертувати дату/час у символьний рядок – тоді 3-й стовпчик таблиці 3 демонструє, як буде виглядати символьний рядок з записом дати/часу.

Зауваження: Якщо використовується дата без століття, то SQL Server інтерпретує двоцифровий запис року (yy) як рік двадцятого століття, якщо він знаходиться в діапазоні від 50 до 99, і як рік 21 століття – якщо двоцифрове число в діапазоні від 0 до 49.

Таким чином функція:

```
CONVERT(datetime,'15/11/40',3)
```

поверне дату: 15 листопаду 2040 року; а функція:

```
CONVERT(datetime,'07.20.70',1)
```

поверне дату: 20 липня 1970 року.

Для впевненості правильності завдання дати краще завжди використовувати формат дати з століттям.

Для перетворення числових типів в рядкові (символьні) можна також використовувати функцію STR.

Функція **STR**

Повертає символьні дані, конвертовані з числових даних.

Синтаксис

STR **(** *float_expression* [**,** *length* [**,** *decimal*]] **)**

Аргументи:

float_expression

Вираз приблизного числового типу даних з десятковою крапкою.

length

Загальна довжина рядка символів, що містить десяткову крапку, знак числа (для від'ємних чисел), цифри та пробіли (пропуски). За замовчуванням *length* = 10.

decimal

Кількість цифр дробової частини числа, тобто кількість цифр праворуч десяткової крапки.

Тип значення, що повертається

char

Зауваження: Для переводу цілого числа у рядок символів можна використовувати як функцію CONVERT, так і функцію STR, але вони по різному додають пробіли, якщо число потребує менш символів, ніж вказано в форматі переведення його у рядок символів: функція CONVERT, додає пробіли праворуч числа (хвостові пробіли), а функцію STR – ліворуч числа.

Це можна побачити з наступного прикладу, в якому номер поточного місяця (у даному випадку це березень) перетворюється у рядок з 4-х символів спочатку функцією `CONVERT`, потім функцією `STR`. Тому запит:

```
select conv=convert(char(4),MONTH(getdate()))+'/',
       st=STR(MONTH(getdate()),4)+'/'
```

Поверне результат:

	conv	st
1	3 /	3/

Якщо аргумент *length* містить меншу довжину, ніж потрібно для числа, що конвертується у рядок, то функція `CONVERT`, так і функція `STR` повернуть рядок з символів '*', але функція `CONVERT` поверне рядок з одного символу '*', а

функція STR поверне рядок з *length* символів '*', як видно з результату наступного запита:

```
select conv=convert(char(4),10000+MONTH(getdate()))+'/',
    st=STR(10000+MONTH(getdate()),4)+'/'
```

Результат запиту:

	conv	st
1	*/	****/

Якщо в функції STR є аргумент *decimal* і аргумент *length* містить меншу довжину, ніж потрібно для числа, то число конвертується у рядок з меншим числом цифр дробової частини, ніж вказано в аргументі , і тільки коли аргумент *length* менше цифр цілої частини числа (разом зі знаком числа для від'ємних чисел), то повертається рядок з символів '*'. Тобто запит:

```
select max_money=STR((select max(post_money) from post),7,2)
```

поверне:

	max_money
1	9000.00

Запит:

```
select max_money=STR((select max(post_money) from post),6,2)
```

поверне:

	max_money
1	9000.0

Запит:

```
select max_money=STR((select max(post_money) from post),5,2)
```

поверне:

	max_money
1	9000

А запит:

```
select max_money=STR((select max(post_money) from post),3,2)
```

поверне:

	max_money
1	***

Функція **CASE**

Обчислює список умов та повертає результат одного з множини можливих виразів.

Функція CASE має два формати:

- Проста CASE функція порівнює значення вхідного виразу з набором простих виразів і визначає результат.
- Пошукова CASE функція обчислює набор виразів логічного типу для визначення результату.

Обидва формати підтримують необов'язковий аргумент ELSE.

Синтаксис

Проста CASE функція:

```
CASE input_expression
   WHEN when_expression THEN result_expression
      [ ...n ]
   [       ELSE else_result_expression      ]
END
```

Пошукова CASE функція:

```
CASE
   WHEN Boolean_expression THEN result_expression
      [ ...n ]
   [       ELSE else_result_expression      ]
END
```

Аргументи:

input_expression

Будь-який припустимий вираз Microsoft SQL Server.

WHEN *when_expression*

Для простої CASE функції *when_expression* повинен бути виразом, що сумісний за порівнянням з виразом *input_expression*.

[...*n*]

Це означає, що конструкцій WHEN *when_expression* THEN *result_expression* (або WHEN *Boolean_expression* THEN *result_expression*) може бути декілька.

THEN *result_expression*

result_expression – це вираз, значення якого повертається, коли значення *input_expression* дорівнює значенню *when_expression* (або коли значення *Boolean_expression* дорівнює TRUE). *result_expression* – будь-який припустимий вираз Microsoft SQL Server.

ELSE *else_result_expression*

else_result_expression – це вираз, значення якого повертається, якщо для простої CASE функції немає жодного *when_expression*, значення якого дорівнює значенню *input_expression*; або якщо для пошукової CASE функції немає жодного *Boolean_expression*, значення якого дорівнює TRUE. Якщо цей аргумент відсутній, але жодна з гілок WHEN не задовольняє умові пошуку, CASE функція повертає NULL.

WHEN *Boolean_expression*

Для пошукової CASE функції. *Boolean_expression* – довільний припустимий вираз логічного типу.

Тип значення, що повертається

Аналогічний типу виразів *result_expression* ***та*** *else_result_expression.*

Приклад використання функцій CASE I CONVERT.

Нехай є БД визначних історичних осіб минулого і сьогодення. Нехай в таблиці `Persona` є атрибути: `Surname` (призвіще, тип varchar), `Name1` (ім'я, тип varchar), `BirthDate` (дата народження, тип datetime), `DeathDate` (дата смерті, тип datetime). Потрібно вивести інформацію о кожній особі у наступному форматі форматі:

```
Олександр Пушкін            17/10/1799 – 25/05/1837
Леонід Кравчук              25/04/1948 –
```

В цьому прикладі атрибут `DeathDate` може мати порожні значення (для живих осіб). Для перевірки, чи має цей атрибут порожнє значення, і виведення або пропуску, або рядка з датою, використовується функція CASE. Функція CONVERT використовується для переведення у символьний рядок дат (народження і смерті). Таким чином:

```
SELECT Особа=Surname+' '+Name1, "Дати життя"=
    CONVERT(char(10), BirthDate,103)+ ' - '+
    CASE
        WHEN DeathDate IS NULL THEN ''
        ELSE CONVERT(char(10), DeathDate,103)
        END
FROM Persona
```

Складна вибірка даних в Query Analyzer MS SQL Server 2000

Основні теоретичні відомості

Агрегатні функції

Крім пошуку інформації, яка зберігається в таблицях, за допомогою запитів також можна отримувати деяку статистичну інформацію: максимальне значення деякого атрибута (або вираза з використанням атрибута), мінімальне, середнє значення. Ці можливості надають агрегатні функції. До агрегатних функцій відносяться функції COUNT, SUM, AVG, MAX, MIN. Зазвичай агрегатні функції використовуються у запитах з групуванням і обчислюють кількість записів в групі, суму деякого атрибута для групи, середнє, мінімальне або максимальне значення для групи деякого атрибута. Агрегатні функції можна використовувати не тільки для одного атрибута, но і для виразу з атрибутом, наприклад, для розрахунку середнього (мінімального, максимального) віку групи осіб за їх датою народження.

Розділ GROUP BY

Вказує групи, для яких розраховуються значення за допомогою агрегатних функцій: суму, кількість записів, середнє значення, максимальне значення, мінімальне значення. Якщо є розділ GROUP BY, в розділі SELECT крім виразів з агрегатними функціями можуть бути тільки стовпці, перелічені в розділі GROUP BY. Якщо в команді SELECT є розділ GROUP BY і немає розділу ORDER BY, упорядкування рядків результуючого набору проводиться за пунктами групування.

Синтаксис команди:

GROUP BY [ALL] *group_by_expression* [**,***...n*]

Параметри:

ALL

Вказує, що всі групи повинні бути представлені у результуючому наборі, навіть якщо в них немає жодного запису. Коли вказаний параметр ALL, для групи без записів (для якої умова фільтру в розділі WHERE не знайде жодного запису) функція COUNT повертає значення арифметичного нуля, інші агрегатні функції (MAX, MIN, SUM, AVG) повертають значення NULL. Якщо параметр не вказаний, група без записів не включається в результат запиту.

group_by_expression

Вказує вираз, за яким групуються запити. Найчастіше вираз складається з імені стовпця, за яким проводиться групування, але це може бути неагрегатний вираз з використанням атрибутів стовпців. Якщо в розділі SELECT для цього виразу був наданий псевдонім стовпця, то в розділі GROUP BY повинен використовуватись вираз, а не псевдонім.

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

Групування, як і упорядкування, може бути багаторівневим.

Приклади використання розділу GROUP BY.

Вирахувати кількість будинків на кожній вулиці, при цьому включити в результат вулиці, на яких немає будинків (якщо в цьому запиті описати зв'язок між таблицями в розділі FROM, то вулиці без будинків не будуть вибрані не зважаючи на параметр ALL):

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
```

Вирахувати загальну кількість кімнат по всіх квартирах будинку, та середню площу квартири для кожного будинку на кожній вулиці, при цьому виключити з результату будинки, в яких немає квартир:

```
SELECT name_street+' д.'+h.number AS house_N,
          SUM(rooms) as kol_vo, AVG(all_square) as AVG_Square
FROM street s, house h,flat f
WHERE s.id_street=h.id_street AND h.id_street=f.id_street AND
      f.number=h.number
GROUP BY name_street+' д.'+h.number
```

Розділ HAVING

Вказує умову пошуку для груп або агрегатних функцій.

Синтаксис команди:

HAVING < search_condition >

Параметр:

< search_condition >

Вказує умову пошуку для груп з використанням агрегатних функцій. Наприклад, якщо в останньому запиті потрібно вибрати тільки будинки з загальною кількістю кімнат не менше 20, то після розділу: GROUP BY … – буде розділ HAVING:

```
HAVING COUNT(*)>=20
```

Приклад: вирахувати кількість будинків на кожній вулиці, при цьому включити в результат тільки ті вулиці, на яких більше 5 будинків:

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
HAVING COUNT(*)>=5
```

В одному запиті можуть бути як розділ WHERE, так і розділ HAVING. Звичайно в розділі WHERE записуються умови пошуку даних в таблицях бази даних, а в розділі HAVING – умови пошуку груп даних, отриманих в результату виконання запиту.

Звіт по лабораторній роботі повинен містити:

- Відповіді на контрольні питання.
- Для кожного пункту завдання: текст завдання, SQL-код виконаного завдання, копію екрану (Screenshot) з результатом виконання завдання.

Перелік запитань до лабораторної роботи №3

1. Які є функції перетворення типів у мові Transact-SQL? Навести синтаксис функцій з описом параметрів.
2. Для чого використовується функція Transact-SQL CASE? Навести можливі формати синтаксису функції та пояснити параметри.

3. Для чого призначений розділ GROUP BY команди Transact-SQL SELECT? Наведіть синтаксис.
4. Для чого призначений розділ HAVING команди Transact-SQL SELECT? Наведіть синтаксис.
5. Наведіть синтаксис та поясніть параметри всіх агрегатних функцій.

Приклади запитів

Запити з використанням функцій CASE та CONVERT

1. Для працівників відділу «Адміністрація», які ще не пішли у відпустку поточного року, вибрати: прізвище з ініціалами, назву посади, сезон виходу у відпустку: зима, весна, літо, осінь, – і дату виходу у відпустку у форматі dd/mm/yyyy:

```
SELECT RTRIM(Surname)+' '+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.' as
[Фамилия И.О.],     Name_post Должность,  [Сезон отпуска]= case
     when month(date_vac) between 3 and 4 then 'весна'
     when month(date_vac) between 6 and 8 then 'лето'
     when month(date_vac) between 9 and 11 then 'осень'
     else 'зима' end,
[Начало отпуска]=convert(char(10),date_vac,103)
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
    INNER JOIN vacation v ON p.id_man=v.id_man
    INNER JOIN Otdel ON otdel.id_otd=w.id_otd
WHERE  name_otd='Администрация' and year_vac=year(getdate())
       and date_vac>getdate() and main_work=1
```

Результат виконання команди видно на рисунку 5.

	Фамилия И.О.	Должность	Сезон отпуска	Начало отпуска
1	Иванов И.И.	Директор	лето	15/07/2012
2	Сорокина С.П.	Секретарь	лето	01/06/2012
3	Иванова И.И.	Зам.директора	лето	25/07/2012
4	Воробьева С.П.	Системный администратор	осень	02/10/2012
5	Иванникова О.И.	Прибиральниця	весна	25/04/2012

Рисунок 5 – Результат команди SELECT

Використання вкладених запитів

2. Для жінок, які не працюють за сумісництвом (не входять в множину працівників-сумісників), вибрати: прізвище з ініціалами, кількість дітей, освіту, відділ, зарплату (зарплата розраховується за формулою `post_money*Stavka`).

```
select ФИО=surname+' '+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.',
 "Кол-во детей"=Children, Образование=Education,
```

```
 Отдел=Name_otd, Зарплата=Post_money*stavka
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
INNER JOIN Otdel ON w.Id_Otd=Otdel.Id_otd
WHERE Sex='ж' and w.Id_man NOT IN
(SELECT Id_man FROM Worker WHERE Main_work=0)
```

Результат виконання команди видно на рисунку 6.

	ФИО	Кол-во детей	Образование	Отдел	Зарплата
1	Петрова О.И.	2	высшее	Бухгалтерия	6000.0000
2	Зимина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
3	Сидорова И.С.	0	н/высшее	Маркетинга	2625.0000
4	Сорокина С.П.	0	среднее	Администрация	2500.0000
5	Иванова И.И.	1	высшее	Администрация	8000.0000
6	Замятина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
7	Антонюк О.П.	2	высшее	Маркетинга	2625.0000
8	Лисицына А.Л.	0	н/высшее	Маркетинга	3500.0000
9	Алексеев О.П.	2	высшее	Маркетинга	2625.0000
10	Ольгина С.П.	0	среднее	Производства	4000.0000
11	Хрущ И.И.	1	высшее	Производства	5000.0000
12	Соколова Л.Л.	0	н/высшее	Производства	4000.0000
13	Медведь И.С.	0	н/высшее	Маркетинга	3500.0000
14	Семенов О.П.	0	высшее	Маркетинга	4000.0000
15	Хрущева С.П.	2	среднее	Производства	3000.0000

Рисунок 6 – Результат команди SELECT

Приклади запитів з групуванням

3. Для кожного відділу розрахувати кількість чоловіків та кількість жінок. Вибрати: назву відділу, стать, кількість осіб цієї статі в цьому відділі.

```
SELECT Name_otd as Отдел, sex as Пол, count(*)  Количество
FROM Persona p INNER JOIN worker w ON p.id_man=w.id_man
     INNER JOIN Otdel ON w.Id_Otd=Otdel.Id_otd
GROUP BY Name_otd, sex
```

В цьому запиті є два рівня групування: за статтю та за відділом. Якщо поміняти місцями атрибути в розділі `GROUP BY` (тобто написати `GROUP BY sex, Name_otd`), то розрахунки залишаться тими самими, але зміниться порядок виводу записів в результаті запиту: або для кожного відділу виводиться спочатку кількість жінок, потім – кількість чоловіків; або спочатку виводиться кількість жінок для кожного відділу, а потім – кількість чоловіків для кожного відділу). Спробуйте виконати цей запит з обома порядками групування та отримайте результат.

4. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальну зарплатню з урахуванням ставки працівника. Включаються відділи, де є працівники з дітьми.

```
SELECT Отдел=Name_otd, [Количество сотрудников]= count(*),
  AVG(YEAR(GETDATE())-YEAR(birthdate)) as [Средний Возраст],
STR(MIN(post_money*Stavka),7,2)) as MIN_Зарплата
FROM Persona p, Worker w , Post, otdel
WHERE p.id_man=w.id_man AND w.id_post=Post.id_post AND
  Otdel.id_otd=w.id_otd AND w.id_otd IN
(SELECT id_otd FROM Persona p join Worker w ON p.id_man=w.id_man
     WHERE children>0)
   GROUP BY Name_otd
```

Результат виконання команди видно на рисунку 7.

	Отдел	Количество сотрудников	Средний Возраст	MIN_Зарплата
1	Администрация	8	41	2000.00
2	Бухгалтерия	8	35	1000.00
3	Маркетинга	15	36	1000.00
4	Производства	15	38	1100.00

Рисунок 7 – Результат команди SELECT

Завдання для виконання лабораторної роботи

1. Для основних працівників (не сумісників) з вищою освітою вибрати: прізвище з ініціалами, вік (на кінець року), вікову групу, назву посади, зарплату. Вікову групу визначаємо наступним чином:

- менше 35 років – молодіж;
- від 35 до 60 років – зрілі;
- більше 60 – похилі (пожилые).

2. Для працівників, що прийняті на роботу раніше минулого року на повну ставку (ставка = 1), вибрати: прізвище з ініціалами, вік (на кінець року), освіту, назву відділу, назву посади, стаж роботи (повних років). Стаж роботи (повних років) вираховується за допомогою функції CASE, з урахуванням того, якого місяця і числа робітник був прийнятий на роботу.
3. Для працівників без вищої освіти, які не працюють за сумісництвом (не входять в множину працівників-сумісників), вибрати: прізвище з ініціалами, назву посади, освіту, відділ, зарплату (зарплата розраховується за формулою `money*Stavka`).
4. Для працівників-чоловіків, що працюють на посадах, на яких немає сумісників (не входять в множину посад працівників-сумісників),

вибрати: прізвище з ініціалами, назву посади, освіту, відділ, стаж роботи (розраховується за формулою: від поточного року відняти рік прийому на роботу).

5. Для працівників, що працюють як сумісники (входять в множину працівників-сумісників) і як основні працівники, вибрати: прізвище з ініціалами, назву посади по основній роботі, освіту, відділ (де працює по основній роботі), оклад (по основній роботі). (Основна робота буде у якості умови фільтру у основному запиті; у вкладеному запиті буде розраховуватися множина ідентифікаторів працівників, що працюють за сумісництвом).
6. Для працівників з дітьми, що не працюють як основні працівникі (не входять в множину основних працівників), вибрати: прізвище з ініціалами, назву посади, кількість дітей, відділ, зарплату (зарплата розраховується за формулою `money*Stavka`).
7. Для працівників з відділів, в яких немає сумісників, вибрати: прізвище з ініціалами, вік, назву посади, відділ, освіту, оклад.
8. Для працівників з відділів, в яких немає жінок, вибрати: прізвище з ініціалами, назву посади, відділ, освіту, сімейний стан.
9. Розрахувати кількість осіб з кожним рівнем освіти, що працюють в кожному відділі (Тобто, скільки осіб є з вищою освітою, скільки – з незакінченою вищою, і т.д. Групувати треба по атрибуту education).
10. Розрахувати кількість осіб з дітьми, що працюють в кожному відділі. Залишити тільки відділи, де таких осіб більше 2.
11. Розрахувати середню та максимальну зарплатню (з урахуванням ставки) в кожному відділі. Залишити тільки відділи з середньою зарплатнею більше 2000, в яких працює не менше 6 осіб.
12. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальний оклад. Включаються відділи, де немає працівників з дітьми.
13. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, максимальний вік працівників, середню зарплату (з урахуванням ставки). Включаються відділи, де приймали працівників у 2010 році.
14. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальну ставку. Включаються відділи, в яких не приймали на роботу працівників поточного року.
15. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, максимальний вік працівників, середній оклад. Включаються відділи, де немає працівників з начальною освітою.

Завдання

В запитах з номерами 6-8 потрібно використати вкладений запит, якій розраховує деяку множину номерів відділів. Основний запит обирає записи, для яких номер відділу входить (або не входить) у розраховану множину номерів відділів. Як і в лабораторній роботі №4 слід продумати логіку при написанні вкладеного запиту. Наприклад, не є одноковими множинами множини номерів відділів, в яких немає жінок, та номерів відділів, в яких є чоловіки (тому що є відділи, в яких працюють й чоловіки, й жінки).

Лабораторна робота №4 Маніпулювання даними

Мета роботи:

3. Знайомство з мовою DML, як частиною мови Transact-SQL;
4. Отримання практичних навичок змінення даних в таблицях та усунення записів з таблиць у СКБД MS SQL Server 2000.

Перелік тем лекційного курсу

3. Команди UPDATE.
4. Команди DELETE.

Основні теоретичні відомості

Редагування даних в таблицях

Для змінення значення полів записів таблиць використовується команда Transact-SQL UPDATE, яку найчастіше використовують у наступному форматі:

UPDATE *table_name* SET *column_name* = { *expression* | DEFAULT | NULL } [**,***...n*]
[FROM < table_source >]
[WHERE < search_condition >]

Параметри команди:

table_name

Ім'я таблиці, в якій змінюють значення атрибутів.

column_name

Ім'я атрибуту (поля таблиці), значення якого змінюється.

expression

Вираз, значення якого привласнюється полю *column_name*. Тип виразу *expression* повинен бути сумісним по присвоюванню з типом даних *column_name.* В якості виразу для вирахування значення атрибуту може використовуватись запит, тобто команда SELECT (дивись розділ "Вибірка даних в Query Analyzer MS SQL Server 2000"). В даному випадку такий запит повинен повертати таблицю з одного значення (один стовпець, один рядок).

DEFAULT | NULL

Ці параметри розглядаються у наступному семестрі.

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

Одною командою можна змінити значення декільком атрибутам (полям таблиці); тобто конструкцію:

column_name = *expression* |

можна записувати кілька разів, відокремлюючи привласнення кожного атрибута комою.

FROM < table_source >

Вказує, що використовуються інші таблиці (базові таблиці, представлення, віртуальні таблиці) для визначення кортежів таблиці *table_name*, в яких будуть виконуватись зазначені зміни полів. Ця конструкція розглядалася вище, у розділі "Вибірка даних в Query Analyzer MS SQL Server 2000" для команди SELECT.

WHERE < search_condition >

Вказує умови пошуку рядків (кортежів) таблиці, в яких будуть змінюватись значення вказаних атрибутів. В умові після ключового слова WHERE можуть використовуватись атрибути той таблиці, для якої змінюються значення полів, і зв'язаних з нею таблиць, вказаних в розділі FROM < table_source >. Зв'язки між таблицями можна задавати і в розділі FROM, і в розділі WHERE (дивись команду SELECT).

Приклади використання команди UPDATE.

Для демонстрації команд редагування даних та видалення записів з бази даних, розглянемо схематичну базу даних житлового фонду міста (`City`) з трьох таблиць: `Street` (Вулиця), `House` (Будинок), `Flat` (Квартира). В таблиці `Street` буде лише два атрибута (стовпця): назва вулиці (`name_street`) та ідентифікатор вулиці в БД (`id_street`). В таблиці `House` будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці (`number`), кількість поверхів в будинку (`count_floor`), кількість квартир (`count_flat`). Первинний ключ таблиці `House` буде складовим: складається з атрибутів `id_street` та `number`. В таблиці `Flat` будуть атрибути: ідентифікатор вулиці, номер будинку на вулиці, номер квартири (`flat_number`), кількість кімнат у квартирі (`rooms`), загальна площа квартири (`all_square`).

Розглянемо декілька прикладів змінення даних в БД `City` командою UPDATE.

1. Змінити назву вулиці Пушкінська на Лермонтівська:

```
UPDATE Street SET name_street='Лермонтівська'
WHERE name_street='Пушкінська'
```

2. Привласнити значення «три» для кількості кімнат у квартирі номер 4 в будинку №1 на вулиці Дерібасівській:

```
UPDATE Flat SET rooms=3 WHERE flat_number=4 and number='1'
and  id_street=
            (SELECT id_street FROM  Street
             WHERE name_street='Дерібасовська')
```

Видалення записів в таблиці

Для видалення записів з таблиці використовується команда Transact-SQL DELETE, яку найчастіше використовують у наступному форматі:

DELETE [FROM] *table_name*
[FROM < table_source >]
[WHERE < search_condition >]

Параметри команди:

table_name

Ім'я таблиці, в якій видаляються записи. Ключове слово FROM перед іменем таблиці можна пропускати. Тобто команди:

```
DELETE Flat
```

та

```
DELETE FROM Flat
```

є ідентичними. До речі, вказані команди повністю видалять всі записи з вказаної таблиці (`Flat`). Щоб видалити не всі записи таблиці, а лише деякі, які відповідають заданій умові, потрібно використовувати параметр WHERE < search_condition >. Цей параметр, а також параметр FROM < table_source > є аналогічними цим же параметрам розглянутої команди UPDATE, і більш докладно розглядаються в команді SELECT.

Якщо командою DELETE видаляють записи з таблиці, первинний ключ якої використовується як зовнішній ключ іншої (підлеглої) таблиці, то SQL Server спочатку перевіряє, чи є в підлеглій таблиці рядки, зовнішній ключ яких дорівнює первинному ключу в тих рядках основної таблиці, що видаляються. Якщо такі рядки є, то може бути два варіанта подій:

- якщо для підлеглої таблиці в описі зовнішнього ключа вказане обмеження цілісності CASCADE, то разом з рядками основної таблиці будуть видалені всі зв'язані з ними зовнішнім ключем рядки підлеглої таблиці;
- якщо для підлеглої таблиці в описі зовнішнього ключа вказане обмеження цілісності NO ACTION (або не вказане нічого), то SQL Server заборонить видаляти рядки основної таблиці, які зв'язані з рядками підлеглої таблиці.

Наприклад, для бази даних `City`, що розглядається у прикладах вище, для будинку №1 по вулиці Дерібасівській в базу даних записані 3 квартири. Нехай каскадного видалення записів не передбачено. Тому наступна команда (видалення вказаного будинку):

```
DELETE house WHERE number='1' AND id_street=
(SELECT id_street FROM street WHERE
 name_street='Дерібасовська')
```

не буде виконана, і вона викличе помилку:

```
DELETE statement conflicted with TABLE REFERENCE constraint
'FK__flat__7F60ED59'. The conflict occurred in database 'city',
table 'flat'.
```

База даних `Personal` не має каскадного видалення даних, тому при спробі, наприклад, усунути з таблиці `Post` посаду, до якої в базі даних приписані працівники (в таблиці `Worker`), SQL Server заборонить видаляти рядки таблиці `Post` і виведе повідомлення про помилку.

Звіт по лабораторній роботі повинен містити:

- Відповіді на контрольні питання.
- Текст завдання, SQL-код виконаного завдання, копію екрану (Screenshot) з результатом виконання завдання.

Перелік запитань до лабораторної роботи №4

3. Яка команда Transact-SQL використовується для видалення записів з таблиці? Наведіть синтаксис команди с поясненням її параметрів.
4. Яка команда Transact-SQL використовується для змінення записів в таблиці? Наведіть синтаксис команди с поясненням її параметрів.

Приклади використання команд UPDATE і DELETE до БД Personal:

1. Працівника з прізвищем Ільїн, який працює в відділу Маркетингу на посаді економіста, перевести на посаду менеджера.
2. Всіх працівників, які працюють як основні працівники, і прийняті на роботу до 2008 року, перевести на повну ставку (тобто 1).
3. Звільнити (усунути з таблиці Worker) усіх сумісників відділу виробництва.
4. Звільнити всіх працівників, яким на кінець року буде не менше 61 року.

Перші два пункти завдання виконуються командою UPDATE, останні два – командою DELETE.

1. Працівника з прізвищем Ільїн, який працює в відділу Маркетингу на посаді економіста, перевести на посаду менеджера.

```
UPDATE Worker SET Id_post=
(SELECT Id_post FROM Post WHERE Name_post='менеджер')
WHERE Id_man IN
(SELECT Id_man FROM Persona WHERE Surname='Ильин')
```

```
AND Id_post=
(SELECT Id_post FROM Post WHERE Name_post='экономист')
AND Id_otdel=
(SELECT Id_otdel FROM Otdel WHERE Name_otdel='Маркетинга')
```

Як видно з наведеного SQL-коду, в команді не використовувалась необов'язкова опція команди FROM (в якій перелічуються додаткові таблиці для пошуку рядків, поля в яких змінюються). Замість цього використані 3 вкладені запити, які локалізують необхідний рядок таблиці `Worker`.

Результат виконання команди UPDATE видно з порівняння рисунків 8 і 9.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	.75
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50
11	9	3	7	2008-11-23 00:00:00.000	1	1.00
12	10	1	4	2008-09-11 00:00:00.000	1	1.00

Рисунок 8 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50
11	9	3	8	2008-11-23 00:00:00.000	1	1.00

Рисунок 9 – Зміна даних в таблиці `Worker` після команди UPDATE

2. Всіх працівників, які працюють як основні працівники, і прийняті на роботу до 2008 року, перевести на повну ставку.

```
UPDATE Worker SET stavka=1
WHERE stavka<1 AND Main_work=1 AND YEAR(date_work)<2008
```

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	1	1	1	2000-01-10 00:00:00.000	1	1.00
2	2	2	5	2000-01-20 00:00:00.000	1	1.00
3	3	4	10	2009-12-15 00:00:00.000	0	.50
4	4	2	6	2010-05-11 00:00:00.000	1	1.00
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	.75
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50
9	8	4	9	2006-08-18 00:00:00.000	1	1.00
10	9	2	6	2010-12-25 00:00:00.000	0	.50

Рисунок 10 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	1	1	1	2000-01-10 00:00:00.000	1	1.00
2	2	2	5	2000-01-20 00:00:00.000	1	1.00
3	3	4	10	2009-12-15 00:00:00.000	0	.50
4	4	2	6	2010-05-11 00:00:00.000	1	1.00
5	5	1	2	2000-01-12 00:00:00.000	1	1.00
6	6	3	8	2002-02-11 00:00:00.000	1	1.00
7	7	4	10	2003-11-12 00:00:00.000	1	1.00
8	8	1	3	2005-09-12 00:00:00.000	0	.50

Рисунок 11 – Зміна даних в таблиці `Worker` після команди UPDATE

3. Звільнити (усунути з таблиці Worker) усіх сумісників відділу виробництва.

```
DELETE Worker WHERE Main_work=0 AND Id_otdel=
(SELECT id_otdel FROM Otdel WHERE Name_otdel='Производства')
```

Як видно з порівняння рисунків 12 і 13, два записи з таблиці `Worker` видалились.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	3	4	10	2009-12-15 00:00:00.000	0	.50
2	7	4	10	2003-11-12 00:00:00.000	1	1.00
3	8	4	9	2006-08-18 00:00:00.000	1	1.00
4	16	4	12	2009-12-15 00:00:00.000	1	1.00
5	19	4	9	2006-08-18 00:00:00.000	1	1.00
6	23	4	4	2006-09-11 00:00:00.000	1	1.00
7	25	4	10	2010-05-11 00:00:00.000	1	1.00
8	26	4	13	2010-12-25 00:00:00.000	0	.50
9	29	4	9	2006-08-18 00:00:00.000	1	1.00
10	30	4	10	2003-11-12 00:00:00.000	1	1.00
11	31	4	14	2008-05-10 00:00:00.000	1	1.00
12	32	4	10	2010-03-12 00:00:00.000	1	1.00
13	33	4	13	2009-12-15 00:00:00.000	1	1.00
14	34	4	10	2006-09-11 00:00:00.000	1	1.00
15	40	4	9	2006-08-18 00:00:00.000	1	1.00

Рисунок 12 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	7	4	10	2003-11-12 00:00:00.000	1	1.00
2	8	4	9	2006-08-18 00:00:00.000	1	1.00
3	16	4	12	2009-12-15 00:00:00.000	1	1.00
4	19	4	9	2006-08-18 00:00:00.000	1	1.00
5	23	4	4	2006-09-11 00:00:00.000	1	1.00
6	25	4	10	2010-05-11 00:00:00.000	1	1.00
7	29	4	9	2006-08-18 00:00:00.000	1	1.00
8	30	4	10	2003-11-12 00:00:00.000	1	1.00
9	31	4	14	2008-05-10 00:00:00.000	1	1.00
10	32	4	10	2010-03-12 00:00:00.000	1	1.00
11	33	4	13	2009-12-15 00:00:00.000	1	1.00
12	34	4	10	2006-09-11 00:00:00.000	1	1.00
13	40	4	9	2006-08-18 00:00:00.000	1	1.00

Рисунок 13 – Видалення записів з таблиці
(звільнення працівників з номерами 3 і 26)

4. Звільнити всіх працівників, яким на кінець року буде не менше 61 року.

```
DELETE Worker WHERE Id_man IN
(SELECT Id_man FROM Persona
WHERE YEAR(GETDATE())-YEAR(birthdate)>=61)
```

Як видно з порівняння рисунків 14 і 15, два записи з таблиці `Worker` видалились.

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	7	4	10	2003-11-12 00:00:00.000	1	1.00
2	8	4	9	2006-08-18 00:00:00.000	1	1.00
3	16	4	12	2009-12-15 00:00:00.000	1	1.00
4	19	4	9	2006-08-18 00:00:00.000	1	1.00
5	23	4	4	2006-09-11 00:00:00.000	1	1.00
6	25	4	10	2010-05-11 00:00:00.000	1	1.00
7	29	4	9	2006-08-18 00:00:00.000	1	1.00
8	30	4	10	2003-11-12 00:00:00.000	1	1.00
9	31	4	14	2008-05-10 00:00:00.000	1	1.00
10	32	4	10	2010-03-12 00:00:00.000	1	1.00
11	33	4	13	2009-12-15 00:00:00.000	1	1.00
12	34	4	10	2006-09-11 00:00:00.000	1	1.00
13	40	4	9	2006-08-18 00:00:00.000	1	1.00
14	39	3	8	2002-02-11 00:00:00.000	1	1.00
15	37	3	7	2003-11-12 00:00:00.000	1	1.00
16	38	3	7	2008-11-23 00:00:00.000	1	1.00
17	35	3	8	2004-01-20 00:00:00.000	1	1.00
18	36	3	8	2010-05-11 00:00:00.000	1	1.00

Рисунок 14 – Попередні значення в таблиці `Worker`

	Id_man	Id_otdel	Id_post	Date_work	Main_work	Stavka
1	8	4	9	2006-08-18 00:00:00.000	1	1.00
2	16	4	12	2009-12-15 00:00:00.000	1	1.00
3	19	4	9	2006-08-18 00:00:00.000	1	1.00
4	23	4	4	2006-09-11 00:00:00.000	1	1.00
5	25	4	10	2010-05-11 00:00:00.000	1	1.00
6	29	4	9	2006-08-18 00:00:00.000	1	1.00
7	30	4	10	2003-11-12 00:00:00.000	1	1.00
8	31	4	14	2008-05-10 00:00:00.000	1	1.00
9	32	4	10	2010-03-12 00:00:00.000	1	1.00
10	33	4	13	2009-12-15 00:00:00.000	1	1.00
11	34	4	10	2006-09-11 00:00:00.000	1	1.00
12	40	4	9	2006-08-18 00:00:00.000	1	1.00
13	39	3	8	2002-02-11 00:00:00.000	1	1.00
14	35	3	8	2004-01-20 00:00:00.000	1	1.00
15	36	3	8	2010-05-11 00:00:00.000	1	1.00

Рисунок 15 – Видалення записів з таблиці
(звільнення працівників з номерами 7 і 38)

Виконання лабораторної роботи

Оскільки всі студенти працюють з одною базою даних, то після виконання одним студентом команди на змінення або усунення записів в таблиці, інші студенти не зможуть виконати ті самі дії. Тому викладач вирішує, у якому порядку який студент виконує команди (наприклад, з першої до останньої підряд; з другої до останньої, а потім – першу; і т.д.). Після

виконання кожним студентом одної команди (UPDATE або DELETE) викладач видаляє всі записи з таблиці `Worker` і потім з файлу вводить знову всі попередні записи в цю таблицю.

Запити на оновлення:

1. Для працівників молодше 40 років, що працюють по основній роботі не на повну ставку і мають оклад менше 3000, встановити повну ставку.
2. Для працівників з вищою освітою, що працюють за сумісництвом на ставку менше 0.5 і мають оклад менше 2500, встановити ставку 0.5.
3. Для неодружених працівників з дітьми, що працюють по основній роботі на ставку менше 0.7 і мають оклад менше 2800, додати 0.15 частину до ставки.
4. Для працівників відділів «Комп'ютерний» та «Бухгалтерія», що мають вищу освіту та працюють по основній роботі на ставку менше 0.8, додати 0.1 частину до ставки.
5. Для працівників молодше 35 років, що мають стаж більше 10 років, працюють не на повну ставку і мають оклад менше 3000, встановити повну ставку.
6. Для працівників молодше 35 років, що працюють на посадах «Системний адміністратор» або «Програміст» по основній роботі на ставку менше 0.75, додати 0.2 частини до ставки.

Запити на усунення:

1. Звільнити (видалити з таблиці Worker) сумісників, які старші за 65 років та працюють на посадах «Директор» або «Головний бухгалтер».
2. Звільнити (видалити з таблиці Worker) працівників зі стажем роботи менше 5 років, які старші за 60 років та працюють на посадах з окладом більше 4000.
3. Звільнити (видалити з таблиці Worker) сумісників без вищої освіти, що працюють на посадах з окладом більше 3500, якщо вони працюють не в відділі виробництва.
4. Звільнити (видалити з таблиці Worker) основних працівників з рівнем освіти «Середня» з посад, які вони займають як сумісники.
5. Звільнити (видалити з таблиці Worker) працівників зі стажем роботи менше 4 років, які старші за 65 років та працюють на посадах з кількістю ставок (`count_post`) у відділі не більше 2.
6. Звільнити (видалити з таблиці Worker) сумісників з рівнем освіти «Середня», що працюють на посадах «Бухгалтер» або «Системний адміністратор».

Література

1. Дейт К. Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.: ил.
2. Дет Л.Дж. Введение в системы баз данных – 6-е изд. – К.: Диалектика, 1998.
3. Мейер С.М. Проектирование баз данных – М.: Мир, 1987.
4. Мамаев Е.В. Micrjsoft® SQL Sever 2000. — СПб.: БХВ-Петербург, 2004. — 1280 с.: ил.
5. Дэвидсон Л. Проектирование баз даннях на SQL Server 2000.: Пер. с англ. – М..: БИНОМ. Лаборатория знаний, 2003. – 680 с., ил.
6. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1440 с.: ил.