МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**ВСП «ФАХОВИЙ КОЛЕДЖ ПРОМИСЛОВОЇ АВТОМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»**

**ЗАТВЕРДЖУЮ**
Заступник директора з
навчально-методичної роботи
ФКПАІТ ОНТУ
_____ Вікторія ОКСАНІЧЕНКО
_____._____.20___ року

# Методичні вказівки для самостійної роботи студентів

## ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ
(шифр за ОПП і назва навчальної дисципліни)

**галузь знань** 12 «Інформаційні технології»
(шифр і назва галузі знань)

**спеціальність** 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

**освітня програма** Комп'ютерні науки
(назва освітньої програми)

**(Шифр за ОПП** 2.4 **)**

Одеса 2022

Методичні вказівки для самостійної роботи студентів з дисципліни «Організація баз даних та знань» для підготовки фахового молодшого бакалавра за спеціальністю 122 «Комп’ютерні науки».

Укладач: викладач вищої кваліфікаційної категорії ФКПАІТ ОНТУ Ю. В. Бурмакіна

Методичні вказівки для самостійної роботи студентів затверджені на засіданні циклової комісії «Комп’ютених наук та інженерії програмного забезпечення» ФКПАІТ ОНТУ
Протокол від ___._____.20___ року, № ____

Голова циклової комісії
_____Тетяна КОСТИРЕНКО
(підпис) (власне ім’я та прізвище)

___._____.20___ року

## Пояснювальна записка

Самостійна робота студентів є однією з важливих складових освітнього процесу. Їй приділяється особлива увага так як під час її виконання формуються уміння освоювати знання без сторонньої допомоги.

У вітчизняній структурі освіти самостійній роботі студента відводиться багато часу та вона стоїть на рівні з такими видами занять як лекції, практичні і лабораторні роботи.

Методичні вказівки призначені для самостійної підготовки студентів, що вивчають бази даних. На самостійний розгляд виносяться теми, які доповнюють розглянутий на лекційних заняттях матеріал. Самостійна робота припускає опрацювання теоретичного матеріалу, відповіді на питання по кожній темі.

Студенти оформляють свою роботу у вигляді конспекту. Результати роботи можуть бути використані при виконанні практичних завдань та при написанні модульного контролю.

Форми перевірки виконання можуть бути найрізноманітніші: опитування, диктант, тест, захист практичної роботи.

Завдання з самостійної роботи видаються поступово протягом семестру в кінці відповідного лекційного чи практичного заняття.

### Функції самостійної роботи:

- сприяє засвоєнню знань, формуванню професійних вмінь і навиків, забезпечує формування професійної компетенції;
- виховує потребу в самоосвіті, розвиває пізнавальні і творчі здібності;
- спонукає до науково-дослідницької роботи.

### Види самостійної роботи:

- опрацювання навчального матеріалу;
- виконання завдань до практичних робіт;
- вирішення варіантних завдань та вправ;
- написання рефератів, докладів за різними темами.

| Тема | Кількість годин |
|---|---|
| Файлові системи | 2 |
| Компоненти середовища СУБД | 2 |
| Трирівнева архітектура системи управління базами даних | 2 |
| Головні переваги та недоліки ранніх СУБД. | 2 |
| Об'єктно-орієнтована модель даних | 2 |
| Фундаментальні властивості відношень в реляційній моделі даних | 2 |
| Обмеження цілісності для записів, таблиць та зв’язків між таблицями. | 2 |
| Дванадцять правил Кодда | 2 |
| Функціональна модель даних. | 2 |
| Модель семантичних об’єктів | 2 |
| Аномалії оновлення в базі даних | 2 |
| Історія створення та виникнення мови запитів SQL. | 2 |
| Основні поняття SQL та типи даних в SQL | 2 |
| Робота з оператором Where | 2 |
| Сортування результатів - оператор Order by | 2 |
| Організація реляційних БД. Створення БД в СУБД MS Access | 2 |
| Створення  запитів на мові SQL в СУБД MS Access | 2 |
| Внутрішні запити. | 2 |
| Використання ключових слів Any та All | 2 |
| Створення форм в СУБД MS Access | 2 |
| Створення фільтрів СУБД MS Access | 2 |
| Створення звітів в СУБД MS Access | 2 |
| Створення макросів в СУБД MS Access | 2 |
| Системний каталог | 2 |
| Налаштування параметрів доступу до даних в C++  Builder | 4 |
| Бази даних Microsoft Access в C++ Builder  з використанням ADO | 4 |
| Перенесення C++ Builder до клієнт/сервер | 2 |
| Відкриті системи | 2 |
| Сервери БД | 2 |
| **Всього годин** | **63** |

**Тема «Файлові системи»**

**(2 години)**

**Мета:** дізнатися про призначення файлових систем.

Історичним кроком став перехід до використання централізованих систем управління файлами. З погляду прикладної програми ***файл*** - це іменована область зовнішньої пам'яті, в яку можна записувати і з якої можна зчитувати дані. Правила іменування файлів, спосіб доступу до даних, що зберігаються у файлі, і структура цих даних залежать від конкретної системи управління файлами і, можливо, від типу файлу. Система управління файлами бере на себе розподіл зовнішньої пам'яті, відображення імен файлів у відповідні адреси в зовнішній пам'яті та забезпечення доступу до даних.

Перша розвинена файлова система була розроблена фірмою IBM для її серії 360. До теперішнього часу вона дуже застаріла, і ми не будемо розглядати її докладно. Зауважимо лише, що в цій системі підтримувалися як суто послідовні, так і індексно-послідовні файли, а реалізація багато в чому спиралася на можливості тільки-но з'явившихся на той час контролерів управління дисковими пристроями. Якщо врахувати до того ж, що поняття файлу в OS/360 було обрано як основне абстрактне поняття, якому відповідав будь-який зовнішній об'єкт, включно із зовнішніми пристроями, то працювати з файлами на рівні користувача було дуже незручно. Була потрібна ціла низка громіздких і перевантажених деталями конструкцій. Усе це добре знайоме програмістам середнього і старшого покоління, які пройшли через використання вітчизняних аналогів комп'ютерів IBM.

**1.1 Структури файлів**

Далі ми будемо говорити про більш сучасні організації файлових систем. Почнемо зі структур файлів. Перш за все, практично у всіх сучасних комп'ютерах основними пристроями зовнішньої пам'яті є магнітні диски з рухомими голівками, і саме вони слугують для зберігання файлів. Такі магнітні диски являють собою пакети магнітних пластин (поверхонь), між якими на одному важелі рухається пакет магнітних головок. Крок руху пакета головок є дискретним, і кожному положенню пакета головок логічно відповідає циліндр магнітного диска. На кожній поверхні циліндр "висікає" доріжку, тож кожна поверхня містить число доріжок, що дорівнює

числу циліндрів. Під час розмітки магнітного диска (спеціальної дії, що передує використанню диска) кожну доріжку розмічають на одну й ту саму кількість блоків у такий спосіб, що в кожний блок можна записати по максимуму одну й ту саму кількість байтів. Таким чином, для здійснення обміну з магнітним диском на рівні апаратури потрібно вказати номер циліндра, номер поверхні, номер блоку на відповідній доріжці і кількість байтів, яку потрібно записати або прочитати від початку цього блоку.

Однак ця можливість обмінюватися з магнітними дисками порціями, меншими за об'єм блоку, нині не використовується у файлових системах. Це пов'язано з двома обставинами. По-перше, під час виконання обміну з диском апаратура виконує три основні дії: підведення голівок до потрібного циліндра, пошук на доріжці потрібного блоку і власне обмін з цим блоком. З усіх цих дій у середньому найбільший час займає перша. Тому істотний виграш у сумарному часі обміну завдяки зчитуванню або записуванню тільки частини блоку отримати практично неможливо. По-друге, для того, щоб працювати з частинами блоків, файлова система повинна забезпечити відповідного розміру буфера оперативної пам'яті, що істотно ускладнює розподіл оперативної пам'яті.

Тому у всіх файлових системах явно або неявно виділяється деякий базовий рівень, що забезпечує роботу з файлами, які представляють набір блоків, що прямо адресуються в адресному просторі файлу. Розмір цих логічних блоків файлу збігається або кратний розміру фізичного блоку диска і зазвичай обирається рівним розміру сторінки віртуальної пам'яті, підтримуваної апаратурою комп'ютера спільно з операційною системою.

У деяких файлових системах базовий рівень доступний користувачеві, але частіше прикривається деяким вищим рівнем, стандартним для користувачів. Поширені два основні підходи. За першого підходу, властивого, наприклад, файловим системам операційних систем фірми DEC RSX і VMS, користувачі представляють файл як послідовність записів. Кожен запис - це послідовність байтів постійного або змінного розміру. Записи можна читати або записувати послідовно або позиціонувати файл на запис із зазначеним номером. Деякі файлові системи дозволяють структурувати записи на поля та оголошувати деякі поля ключами запису. У таких файлових системах можна вимагати вибірку запису з файлу за його

заданим ключем. Природно, що в цьому випадку файлова система підтримує в тому ж (або іншому, службовому) базовому файлі додаткові, невидимі користувачеві, службові структури даних. Поширені способи організації ключових файлів ґрунтуються на техніці хешування і B-дерев (ми говоритимемо про ці прийоми детальніше в наступних лекціях). Існують і багатоключові способи організації файлів.

Другий підхід, що став поширеним разом з операційною системою UNIX, полягає в тому, що будь-який файл представляється як послідовність байтів. З файлу можна прочитати вказану кількість байтів або починаючи з його початку, або попередньо здійснивши його позиціювання на байт із зазначеним номером. Аналогічно, можна записати вказане число байтів у кінець файлу, або попередньо зробивши позиціонування файлу. Зауважимо, що проте прихованим від користувача, але існуючим у всіх різновидах файлових систем ОС UNIX, є базове блокове представлення файлу.

Звичайно, для обох підходів можна забезпечити набір перетворювальних функцій, які приводять подання файлу до деякого іншого вигляду. Прикладом тому слугує підтримання стандартного файлового середовища системи програмування мовою Сі в середовищі операційних систем фірми DEC.

**1.2 Іменування файлів**

Зупинимося коротко на способах іменування файлів. Усі сучасні файлові системи підтримують багаторівневе іменування файлів за рахунок підтримання в зовнішній пам'яті додаткових файлів зі спеціальною структурою - каталогів. Кожен каталог містить імена каталогів та/або файлів, що містяться в цьому каталозі. Таким чином, повне ім'я файлу складається зі списку імен каталогів плюс ім'я файлу в каталозі, що безпосередньо містить цей файл. Різниця між способами іменування файлів у різних файлових системах полягає в тому, з чого починається цей ланцюжок імен.

У цьому відношенні є два крайні варіанти. У багатьох системах управління файлами потрібно, щоб кожен архів файлів (повне дерево довідників) цілком розташовувався на одному дисковому пакеті (або логічному диску, розділі фізичного дискового пакету, що представляється за допомогою засобів операційної системи як окремий диск). У цьому разі повне ім'я файлу починається з імені

дискового пристрою, на якому встановлено відповідний диск. Такий спосіб іменування використовується у файлових системах фірми DEC, дуже близько до цього знаходяться і файлові системи персональних комп'ютерів. Можна назвати цю організацію підтриманням ізольованих файлових систем.

Інший крайній варіант був реалізований у файлових системах операційної системи Multics. Ця система заслуговує на окрему велику розмову, в ній було реалізовано цілу низку оригінальних ідей, але ми зупинимося тільки на особливостях організації архіву файлів. У файловій системі Miltics користувачі представляли всю сукупність каталогів і файлів як єдине дерево. Повне ім'я файлу починалося з імені кореневого каталогу, і користувач не зобов'язаний був піклуватися про встановлення на дисковий пристрій будь-яких конкретних дисків. Сама система, виконуючи пошук файлу за його ім'ям, запитувала встановлення необхідних дисків. Таку файлову систему можна назвати повністю централізованою.

Звичайно, багато в чому централізовані файлові системи зручніші за ізольовані: система управління файлами бере на себе більше рутинної роботи. Але в таких системах виникають суттєві проблеми, якщо комусь потрібно перенести піддерево файлової системи на іншу обчислювальну установку. Компромісне рішення застосовано у файлових системах ОС UNIX. На базовому рівні в цих файлових системах підтримуються ізольовані архіви файлів. Один із цих архівів оголошується кореневою файловою системою. Після запуску системи можна "змонтувати" кореневу файлову систему і низку ізольованих файлових систем в одну загальну файлову систему. Технічно це здійснюється за допомогою заведення в кореневій файловій системі спеціальних порожніх каталогів. Спеціальний системний виклик кур'єр ОС UNIX дає змогу під'єднати до одного з цих порожніх каталогів кореневий каталог зазначеного архіву файлів. Після монтування загальної файлової системи іменування файлів здійснюється так само, як якщо б вона з самого початку була централізованою. Якщо врахувати, що зазвичай монтування файлової системи здійснюється під час розкручування системи, то користувачі ОС UNIX зазвичай і не замислюються про початкове походження загальної файлової системи.

### 1.3 Захист файлів

Оскільки файлові системи є загальним сховищем файлів, що належать, узагалі кажучи, різним користувачам, системи керування файлами повинні забезпечувати

авторизацію доступу до файлів. У загальному вигляді підхід полягає в тому, що стосовно кожного зареєстрованого користувача даної обчислювальної системи для кожного наявного файлу вказуються дії, які дозволені або заборонені даному користувачеві. Існували спроби реалізувати цей підхід у повному обсязі. Але це спричиняло занадто великі накладні витрати як зі зберігання надлишкової інформації, так і з використання цієї інформації для контролю правомочності доступу.

Тому в більшості сучасних систем управління файлами застосовується підхід до захисту файлів, вперше реалізований в ОС UNIX. У цій системі кожному зареєстрованому користувачеві відповідає пара цілочисельних ідентифікаторів: ідентифікатор групи, до якої належить цей користувач, і його власний ідентифікатор у групі. Відповідно, при кожному файлі зберігається повний ідентифікатор користувача, який створив цей файл, і зазначається, які дії з файлом може виконувати він сам, які дії з файлом доступні для інших користувачів тієї самої групи, і що можуть робити з файлом користувачі інших груп. Ця інформація дуже компактна, при перевірці потрібна невелика кількість дій, і цей спосіб контролю доступу задовільний у більшості випадків.

### 1.4 Режим багатокористувацького доступу

Останнє, на чому ми зупинимося у зв'язку з файлами, - це способи їх використання в багатокористувацькому середовищі. Якщо операційна система підтримує багатокористувацький режим, цілком реальна ситуація, коли два або більше користувачів одночасно намагаються працювати з одним і тим самим файлом. Якщо всі ці користувачі збираються тільки читати файл, нічого страшного не станеться. Але якщо хоча б один із них буде змінювати файл, для коректної роботи цієї групи потрібна взаємна синхронізація.

Історично у файлових системах застосовувався такий підхід. В операції відкриття файлу (першій і обов'язковій операції, з якої має починатися сеанс роботи з файлом), крім інших параметрів, вказували режим роботи (читання або зміна). Якщо до моменту виконання цієї операції від імені деякої програми А файл уже перебував у відкритому стані від імені деякої іншої програми В (правильніше казати "процесу", але ми не вдаватимемося до термінологічних тонкощів), до того ж наявний режим відкриття був несумісний із бажаним режимом (сумісні лише

режими читання), то залежно від особливостей системи програма А або повідомляла про неможливість відкриття файлу в бажаному режимі, або блокувала його доти, доки програма В не виконає операцію закриття файлу.

Зауважимо, що в ранніх версіях файлової системи ОС UNIX взагалі не було реалізовано жодних засобів синхронізації паралельного доступу до файлів. Операція відкриття файлу виконувалася завжди для будь-якого наявного файлу, якщо цей користувач мав відповідні права доступу. У разі спільної роботи синхронізацію слід було проводити поза файловою системою (і особливих засобів для цього ОС UNIX не надавала). У сучасних реалізаціях файлових систем ОС UNIX за бажанням користувача підтримується синхронізація під час відкриття файлів. Крім того, існує можливість синхронізації декількох процесів, які паралельно модифікують один і той самий файл. Для цього введено спеціальний механізм синхронізаційних захоплень діапазонів адрес відкритого файлу.

**Питання щодо самоконтролю:**

1. Що таке файлові системи?
2. Яке призначення файлових систем?
3. Які більш сучасні організації файлових систем ви знаєте?

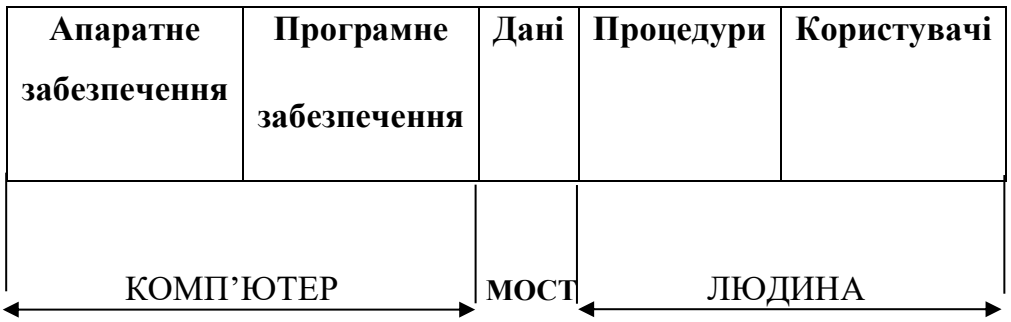
## Тема «Компоненти середовища СУБД»

## (2 години)

**Мета:** визначення головних компонентів середовища СУБД та ознайомлення з їх змістом.

В середовищі СУБД можна визначити 5 головних компонентів:

1. апаратне забезпечення;
2. програмне забезпечення;
3. дані;
4. процедури;
5. користувачі.




1. **Апаратне забезпечення**

Для роботи СУБД та додатку необхідне деяке апаратне забезпечення. Воно може змінюватися в широких межах – від одного ПК до мережі з декількох комп'ютерів. Апаратне забезпечення, яке використовується залежить від вимог даної організації та СУБД, яка використовується. Деякі СУБД призначені для роботи тільки з конкретними типами ОС чи обладнанням, інші мають змогу працювати з широким колом апаратного забезпечення та різними ОС.

2. **Програмне забезпечення**

Цей компонент охоплює програмне забезпечення самої СУБД та прикладних програм разом з ОС, охоплюючи мережеве програмне забезпечення, якщо СУБД використовується в мережі.

Звичайно, додатки створюються на мовах третього покоління. Таких як C, Cobol, Ada, Pascal чи на мовах четвертого покоління, таких як SQL, оператори якого впроваджуються до програми на мовах третього покоління. Втім, СУБД може мати свої особисті інструменти четвертого покоління, призначені для швидкої розробки додатків з використанням вбудованих непроцедурних мов запитів, генераторів звітів, форм, графічних зображень та навіть повномасштабних додатків.

3. **Дані**

Найбільш важливим компонентом середовища СУБД (з точки зору кінцевих користувачів) є дані, які грають роль мосту між комп'ютером та людиною. База даних містить як робочі дані, так і метадані, тобто **„данні про данні"**. Структура бази даних має назву **схема.**

4. **Процедури**

До процедур належать інструкції та правила, які повинні враховуватися при проектуванні та використанні БД.

Користувачам та обслуговуючому персоналу важливо надати документацію, яка містить досконалий опис процедур використання та супровід цієї системи, включаючи інструкції про правила виконання наступних дій:

- реєстрація в СУБД;
- використання деякого інструменту СУБД чи додатку;
- запуск та зупинка СУБД;
- створювання резервних копій СУБД;
- обробка збоїв апаратного та програмного забезпечення, включаючи процедури ідентифікації компоненту, який вийшов з ладу, виправлення відмовившого компоненту (наприклад, за допомогою виклику спеціаліста з ремонту апаратного забезпечення), а також відновлення бази даних після ліквідування несправностей;
- зміна структури таблиці, реорганізація бази даних, розташованої на деяких дисках, способи поліпшення продуктивності та методи архівації даних на другорядних пристроях зберігання.

## 5. Користувачі

Серед них можливо виділити 4 різних групи: адміністратори даних та баз даних, розробники БД, прикладні програмісти та кінцеві користувачі.

### o Адміністратори даних та баз даних

База даних та СУБД є корпоративними ресурсами, якими слід керувати так само, як іншими ресурсами.

**Адміністратор даних (АД)** відповідає за керування даними, враховуючи планування бази даних, розробку та супровід стандартів, бізнес-правил та ділових процедур.

АД консультує та дає рекомендації керівнику найвищої ланки, контролює співвідношення спільного напрямку розвитку бази даних, встановленним корпоративній меті.

**Адміністратор баз даних (АБД)** відповідає за фізичну реалізацію бази даних, враховуючи фізичне проектування та втілення проекту, за забезпечення безпеки та цілісності даних, за супровід ОС, а також за забезпечення максимальної продуктивності додатків та користувачів.

У порівнянні з АД, обов’язки АБД носять більш технічний характер, та для нього необхідне знання конкретної СКБД та системного оточування.

- **Розробники баз даних**

В проектуванні великих баз даних приймають участь два різних типи розробників: **розробники логічної бази даних** та **розробники фізичної бази даних**.

Розробник логічної бази даних займається ідентифікацією даних (тобто сутностей та її атрибутів), встановлює зв'язки між даними та обмеження, які накладаються на дані, які зберігаються.

Розробник фізичної бази даних отримує готову логічну модель даних та займається її фізичною реалізацією, у тому числі:

- перетворенням логічної моделі даних в набір таблиць та обмежень цілісності даних;
- обранням конкретних структур зберігання та методів доступу до даних, які забезпечують необхідний рівень продуктивності при роботі з базою даних;
- проектуванням різних вимагаємих засобів безпеки даних.

Розробник фізичної бази даних повинен розбиратися в функціональних можливостях кінцевої СУБД та розуміти переваги та недоліки кожного можливого варіанту втілення. Він повинен вміти обирати найбільш влаштовуючу стратегію зберігання даних за обліком усіх існуючих особливостей їх використання.

- **Прикладні програмісти**

Одразу після створення бази даних треба почати розробку додатків, які надають користувачам необхідні їм функціональні можливості. Саме цю роботу і виконують прикладні програмісти. Звичайно вони працюють на основі специфікацій, які створені системними аналітиками.

- **Кінцеві користувачі**

Користувачами є клієнти бази даних – вона проектується, створюється та підтримується для того, щоб обслуговувати їх інформаційні потреби.

Користувачів можна поділити за засобом використання ними системи:

1. **наївні користувачі** звичайно навіть і не підозрюють про присутність СУБД.

Вони звертаються до бази даних за допомогою спеціальних додатків, які дозволяють в максимальному ступені спростити виконання ними операції.

Такі користувачі ініціюють виконання операцій бази даних, виконуючі найпростіші команди чи обираючи пункти меню. Це значить, що таким користувачам не потрібно нічого розуміти про СУБД.

Наприклад, для того щоб дізнатися ціну товару, касир в супермаркеті використовує сканер для зчитування нанесеного на нього штрих-коду. В підсумку цієї простої дії спеціальна програма не тільки зчитує штрих-код, але й обирає на основі його значення ціну товару з бази даних, а також зменшує значення в іншому полі базі даних, який визначає залишок товарів на складі, після чого зображує ціну та загальну вартість на касовому апараті.

2. **користувачі з досвідом** знайомі зі структурою бази даних та можливостями СУБД. Для виконання вимагаємих операцій вони можуть використовувати таку мову запитів високого рівня, як SQL. А деякі користувачі з досвідом можуть створювати власні прикладні програми.

**Питання щодо самоконтролю:**

1. Які компоненти можна визначити в середовищі СУБД?
2. Який компонент середовища СУБД вважається найважливішим з точки зору користувачів?
3. Що таке „Метадані”?
4. Для чого призначена схема бази даних?
5. Які функції в середовищі СУБД виконує програмне забезпечення?
6. На які групи можна розподілити користувачів?
7. Чим відрізняються функції адміністратору даних від функцій адміністратору баз даних?
8. Чим займаються розробники баз даних?

## Тема «Трирівнева архітектура системи управління базами даних»

## (2 години)

**Мета:** впровадження ідентифікації трьох різних рівнів опису елементів даних, які формують архітектуру систем управління базами даних.

Найбільш фундаментальним моментом у звітах дослідницьких груп про архітектуру системи керування базами даних є ідентифікація трьох різних рівней

опису елементів даних. Ці рівні формують трьохрівневу архітектуру, яка охоплює зовнішній, внутрішній та концептуальний рівні.

Рівень, на якому сприймають дані користувачі, називається ***зовнішнім рівнем***, тоді як система керування базами даних та операційна система сприймають дані на ***внутрішньому рівні***. ***Концептуальний рівень*** зображення даних призначений для відображення зовнішнього рівня на внутрішньому та забезпечення необхідної незалежності одне від одного.

**Зовнішній рівень** – це зображення бази даних з точки зору користувачів. Цей рівень описує ту частину бази даних, яка належить до кожного користувача.

Зовнішнє зображення містить тільки ті сутності, атрибути та зв'язки «реального» світу, які цікаві користувачу, інші сутності, атрибути або зв'язки, які йому не цікаві, також можуть бути зображені в базі даних, але користувач може навіть не підозрювати про їх існування. Різні зображення можуть по-різному відобразити одні й ті ж дані. Наприклад, один користувач може продивлятися дати в форматі *день, місяць, рік*, а інший – в форматі *рік, місяць, день*.

**Концептуальний рівень** – узагальнене зображення бази даних. Цей рівень описує те, які дані зберігаються в базі даних, а також зв'язки, існуючі між ними. Фактично, це повне зображення вимог до даних з боку організації, які не залежать від будь-яких міркувань відносно способу їх зберігання.

На концептуальному рівні зображені такі компоненти:

- усі сутності, атрибути та зв'язки;
- обмеження на дані;
- семантична інформація про дані;
- інформація про заходи безпеки та підтримка цілісності.

**Внутрішній рівень** – фізичне зображення бази даних в комп'ютері. Цей рівень описує те, яка інформація зберігається в базі даних.

Внутрішній рівень описує фізичну реалізацію бази даних, призначену для досягнення оптимальної продуктивності та забезпечення економного використання дискового простору. Він містить опис структур даних та організації окремих файлів, які використовуються для зберігання даних в запам'ятовуючих пристроях.

На внутрішньому рівні зберігається така інформація:

- розподіл дискового простору для зберігання даних та індексів;

- опис подробиць зберігання записів (з вказівкою реальних розмірів зберігаємих елементів даних);
- відомості про розміщення записів;
- відомості про стиснення даних та обраних методах їх шифрування.

**Мета трирівневої архітектури** – це відокремлення користувацького зображення бази даних від її фізичного зображення.

**Питання щодо самоконтролю:**

1. Що є найбільш фундаментальним моментом у звітах дослідницьких груп про архітектуру СУБД?
2. Які рівні формують трирівневу архітектуру СУБД?
3. На якому рівні сприймають дані користувачі?
4. На якому рівні сприймають дані СУБД та операційна система?
5. Для чого призначений концептуальний рівень зображення даних?
6. Яка інформація міститься на кожному з рівнів архітектури СУБД?
7. Навіщо архітектура СУБД була поділена на три рівні?

## Тема «Об'єктно-орієнтована модель даних»

## (2 години)

**Мета:** дізнатися про об’єктно-орієнтовану модель даних та її властивості.

Створення об'єктно-орієнтованих СУБД вважається одним з найбільш перспективних напрямків в галузі розробки нових типів баз даних.

Об'єктно-орієнтовані СУБД базуються на ідеях, сформованих в об'єктно-орієнтованих мовах програмування (**наслідування**, **інкапсуляції** та **поліморфізма**). Предметна область представляється у вигляді множини класів об'єктів. Структура та поведінка об'єктів одного класу (наприклад, товарів бази даних торгового підприємства) є однаковими.

Об'єкт володіє наступними характеристиками:

***1.*** Має унікальний ідентифікатор, який однозначно визначає об'єкт.

***2.*** Належить до деякого класу, який володіє визначеними поведінкою та властивостями.

***3.*** Може обмінюватися повідомленнями з іншими об'єктами.

***4.*** Має деяку внутрішню структуру. Об'єкти, внутрішня структура яких прихована від користувачів (відомо лише, які функції може виконувати даний об'єкт), мають назву **інкапсульованими**.

Поведінка об'єкта задається за допомогою методів його класу – операцій, які можна застосовувати до об'єкту. Здібність застосовувати один й той же метод для різних класів називається **поліморфізмом**.

В об'єктно-орієнтованій моделі можливе створення нового классу об’єктів на основі вже існуючого классу. Цей процес називається **наслідуванням**. Новий клас, який має назву підклас існуючого класу (**суперкласу**), наслідує всі властивості та методи суперкласу. Крім того, для нього можуть бути визначені додаткові властивості та методи.

Об'єктно-орієнтована СУБД дозволяє зберігати об'єкти та забезпечує їх спільне використання різноманітними додатками. Для цього вона повинна володіти наступними компонентами:

***1.*** мовою баз даних, яка дозволяє декларувати класи об'єктів, а потім створювати, зберігати, вилучати та знищувати об'єкти.

***2.*** Сховищем об'єктів, до яких можуть отримати доступ різні додатки. Для ссилок на об'єкти використовуються їх ідентифікатори.

Для практичної реалізації об'єктно-орієнтованих баз даних застосовуються два підходи:

1. Використовується мова об'єктно-орієнтованого програмування (наприклад, C++), доповнений засобами, які дозволяють при необхідності зберігати об'єкти після завершення програми, за допомогою якої вони були створені.

2. Основою є реляційна система, до якої додаються об'єктно-орієнтовані компоненти.

***Недоліки об'єктно-орієнтованих баз даних:***

1) відсутнє необхідне уніфіковане теоретичне обгрунтування та стандартизована термінологія;

2) не існує формально сформульованої методології проектування баз даних;

3) відсутні засоби створення нерегламентованих запитів;

4) немає загальнихправил підтримки узгоджуванності даних.

В завершення можна відмітити, що об'єктно-орієнтовані бази даних в сучасний час дуже важкі в проектуванні та експлуатації, що обмежує їх практичне застосування. Тому, недивлячись на інтенсивні дослідження, які продовжуються, об'єктно-орієнтована модель даних доки підтримується лише декількома СУБД (POET, Jasmine, Versant, Iris).

**Питання щодо самоконтролю:**

1. На яких ідеях базуються об'єктно-орієнтовані СУБД?
2. Що означає поняття наслідування**,** інкапсуляції та поліморфізму в об'єктно-орієнтованих моделях даних?
3. Які підходи застосовуються для практичної реалізації об'єктно-орієнтованих баз даних? Описати ці підходи.
4. Які Ви можете назвати недоліки об'єктно-орієнтованих баз даних?

## Тема «Фундаментальні властивості відношень в реляційній моделі даних» (2 години)

**Мета:** дізнатися про фундаментальні властивості відношень в реляційній моделі даних.

В реляційній моделі підтримуються такі фундаментальні властивості відношень:

- відсутність кортежів-дублікатів;
- відсутність впорядкованості кортежів;
- відсутність впорядкованості атрибутів;
- атомарність значень атрибутів.

Розглянемо детальніше кожне з цих властивостей.

### 1. Відсутність кортежів-дублікатів

Та властивість, що відношення не містять кортежів-дублікатів, виходить з визначення відношення як безлічі кортежів. У класичній теорії множин за визначенням кожна множина складається з різних елементів.

З цієї властивості витікає наявність у кожного відношення так званого первинного ключа - набору атрибутів, значення яких однозначно визначають кортеж відношення. Для кожного відношення принаймні повний набір його атрибутів має

цю властивість. Проте при формальному визначенні первинного ключа потрібно забезпечення його «мінімальності», тобто в набір атрибутів первинного ключа не повинні входити такі атрибути, які можна відкинути без збитку для основної властивості, - однозначно визначати кортеж. Поняття первинного ключа є виключно важливим у зв'язку з поняттям цілісності баз даних.

**2. Відсутність впорядкованості кортежів**

Властивість відсутності впорядкованості кортежів відношення також є наслідком визначення відношення-екземпляра як множини кортежів. Відсутність вимоги до підтримки порядку на безлічі кортежів відношення дає додаткову гнучкість СУБД при зберіганні баз даних в зовнішній пам'яті і при виконанні запитів до бази даних. Це не суперечить тому, що при формулюванні запиту до БД, наприклад, на мові SQL можна зажадати сортування результуючої таблиці відповідно до значень деяких стовпців. Такий результат, взагалі кажучи, не відношення, а деякий впорядкований список кортежів.

**3. Відсутність впорядкованості атрибутів**

Атрибути відношень не впорядковані, оскільки за визначенням схема відношення є безліч пар {ім'я атрибуту, ім'я домена}. Для посилання на значення атрибуту в кортежі відношення завжди використовується ім'я атрибуту. Ця властивість теоретично дозволяє, наприклад, модифікувати схеми існуючих стосунків не лише шляхом додавання нових атрибутів, але і шляхом видалення існуючих атрибутів. Проте у більшості існуючих систем така можливість не допускається, і хоча впорядкованість набору атрибутів відношення явно не потрібно, часто як неявний порядок атрибутів використовується їх порядок в лінійній формі визначення схеми відношення.

**4. Атомарність значень атрибутів**

Значення усіх атрибутів є атомарними. Це витікає з визначення домена як потенційної множини значень простого типу даних, тобто серед значень домена не може міститися множина значень (відношення). Прийнято говорити, що в реляційних базах даних допускаються тільки нормалізовані відношення або відношення, представлені в першій нормальній формі. Потенційним прикладом ненормалізованого відношення є наступне:

| ***Номер відділення*** | ***Відділення*** | | |
|---|---|---|---|
| | ***Номер співробітника*** | ***ПІБ співробітника*** | ***ЗП співробітника*** |
| 310 | 2934 | Євдокімов М. К. | 3000 |
| | 2935 | Афоніна О. М. | 3200 |
| | 2936 | Петров А. А. | 4000 |
| 313 | 2937 | Константинов П. Р. | 4500 |
| 315 | 2938 | Мішина Г. Є. | 3900 |

Можна сказати, що тут ми маємо бінарне відношення, значеннями атрибуту ВІДДІЛЕННЯ якого є відношення. Помітимо, що початкове відношення СПІВРОБІТНИКИ є нормалізованим варіантом відношення ВІДДІЛЕННЯ:

| ***Номер співробітника*** | ***ПІБ співробітника*** | ***ЗП співробітника*** | ***Номер відділення*** |
|---|---|---|---|
| 2934 | Євдокімов М. К. | 3000 | 310 |
| 2935 | Афоніна О. М. | 3200 | 310 |
| 2936 | Петров А. А. | 4000 | 310 |
| 2937 | Константинов П. Р. | 4500 | 313 |
| 2938 | Мішина Г. Є. | 3900 | 315 |

Нормалізовані відношення складають основу класичного реляційного підходу до організації баз даних. Вони мають деякі обмеження (не будь-яку інформацію зручно представляти у вигляді плоских таблиць), але істотно спрощують маніпулювання даними. Розглянемо, наприклад, два ідентичних оператори занесення кортежу:

Зарахувати співробітника Кузнєцова А. В. (номер співробітника - 3000, ЗП співробітника - 3700) у відділення номер 320 і

Зарахувати співробітника Кузнєцова А. В. (номер співробітника - 3000, ЗП співробітника - 3700) у відділення номер 310.

Якщо інформація про співробітників представлена у вигляді відношення СПІВРОБІТНИКИ, обидва оператори виконуватимуться однаково (вставити кортеж у відношення СПІВРОБІТНИКИ). Якщо ж працювати з ненормалізованим відношенням ВІДДІЛЕННЯ, то перший оператор виразиться в занесення кортежу, а

другий - на додавання інформації про Кузнєцова в множинне значення атрибуту ВІДДІЛЕННЯ кортежу з первинним ключем 310.

**Питання щодо самоконтролю:**

1. Які фундаментальні властивості відношень підтримуються в реляційній моделі?
2. Що означає властивість відношення «Відсутність кортежів-дублікатів»?
3. Що означає властивість відношення «Відсутність впорядкованості кортежів»?
4. Що означає властивість відношення «Відсутність впорядкованості атрибутів»?
5. Що означає властивість відношення «Атомарність значень атрибутів»?

## Тема «Дванадцать правил Е. Ф. Кодду»

## (2 години)

**Мета**: ознайомлення зі змістом правил французського вченого та математика Е. Ф. Кодду, яким повина відповідати кожна реляційна СУБД.

Французський вчений та математик Е. Ф. Кодд запропанував 12 правил, яким повинна відповідати кожна реляційна система управління базами даних. Ними стали:

**1. Правило інформації**

Вся інформація в базі даних повина бути зображена виключно на логічному рівні та тільки одним способом – у вигляді значень, які є в наявності в таблицях. Фактично це неформальне визначення реляційної бази даних.

**2. Правило гарантійного доступу**

Логічний доступ до всіх та кожного елементу даних (атомарному значенню) в реляційній базі даних повинен забезпечуватися шляхом використання комбінації імен таблиць, первинного ключа та імені стовпця.

Правило 2 вказує на роль первинного ключу при пошуку інформації в базі даних. Ім’я таблиці дозволяє знайти потрібну таблицю, ім’я стовбця дозволяє знайти потрібний стовбець, а первинний ключ дозволяє знайти строку, яка містить необхідний елемент даних.

**3. Правило підтримки недійсних значень**

В теперішній базі даних повина бути реалізована підтримка недійсних значень, які відрізняються від рядків символів нулевої довжини, рядки символів пробілу та від нуля або іншого числа та використовуються для зображення відсутніх даних незалежно від типу цих даних.

Правило 3 вимагає, щоб відсутні дані можна було б зобразити за допомогою недійсних значень (NULL).

**4. Правило дінамічного каталогу, заснованого на реляційній моделі**

Опис бази даних на логічному рівні повинен бути зображений в такому ж саме вигляді, що й головні дані, щоб користувачі, володіючі відповідними правами, мали змогу працювати з ним за допомогою тієї ж реляційної мови, яку вони використовують для роботи з головними даними.

Правило 4 говорить, що реляційна база даних повина сама себе описувати. Іншими словами, база даних повина вміщувати набір системних таблиць, які описують структуру самої бази даних.

**5. Правило вичерпної підмови даних**

Реляційна система може підтримувати різні мови та режими взаємодії з користувачем (наприклад, режим питань та відповідів). Але повина існувати хоча б одна мова, оператори якої можна було б зобразити у вигляді строк символів у відповідність з деяким чітко зазначеним синтаксисом та який в повній мірі підтримує наступні елементи:

- визначення даних;
- визначення уявлень;
- обробка даних (інтерактивна та програмна);
- умови цілісності;
- ідентифікація прав доступу;
- межі транзакцій (початок, завершення, відміна).

Правило 5 потребує, щоб СУБД використовувала мову реляційної бази даних. Така мова повина підтримувати всі головні функції СУБД – створення бази даних, читання та введення даних, реалізація захисту бази даних та ін.

**6. Правило оновлення уявлень**

Всі уявлення, які теоретично можна оновити, повинні бути доступні для оновлення.

Правило 6 відноситься до уявлень, які є віртуальними таблицями, дозволяючими показувати різним користувачам різні фрагменти структури бази даних.

**7. Правило додавання, оновлення та знищення**

Можливість працювати з відношеннями як з одним операндом повина існувати не тільки при читанні даних, але й при їх додаванні, оновленні та знищенні даних.

Правило 7 акцентує увагу на тому, що бази даних за своєю природою орієнтовані на множину. Воно потребує, щоб операції додавання, оновлення та знищення можна було б виконати над множиною рядків.

**8. Правило незалежності фізичних даних**

Прикладні програми та утіліти для роботи з даними повині на логічному рівні залишатися недоторканими при будь-яких змінах засобів зберігання даних або методів доступу до них.

**9. Правило незалежності логічних даних**

Прикладні програми та утіліти для роботи з даними повині на логічному рівні залишатися недоторканими при внесенні в базові таблиці будь-яких змін, які теоретично дозволяють зберігти недоторканими наявні в цих таблицях дані.

Правила 8 та 9 визначають відокремлення користувача та прикладної програми від низькорівневої реалізації бази даних.

**10. Правило незалежності умов цілісності**

Повина існувати можливість визначення умов цілісності, специфічніх для кожної конкретної реляційної бази даних, на підмові реляційної бази даних та зберігання їх в каталозі, а не в прикладній програмі.

Правило 10 визначає, що мова бази даних повина підтримувати обмежувальні умови, які накладаються на вхідні дані та дії, які можуть бути виконані над даними.

**11. Правило незалежності поширювання**

Реляційна СУБД не повинна залежити від потреб конкретного користувача.

Правило 11 говорить про те, що мова баз даних повина забезпечувати можливість роботи з розподіленими даними, які розміщені на інших комп’ютерних системах.

**12. Правило єдиної мови**

Якщо в реляційній системі є низькорівнева мова (за один раз опрацьовує один

запис), то повинна бути відсутня можливість використання її для того, щоб обійти правила та умови цілісності, виявленні на реляційной мові високого рівня (за один раз опрацьовує декілька записів).

Правило 12 попереджує використання інших можливостей для роботи з базою даних окрім мови бази даних, так як це може викликати порушення її цілістності.

**Питання щодо самоконтролю:**

1. Хто запропанував 12 правил, яким повина відповідати кожна реляційна СУБД?
2. Які правила були запропановані для реляційної СУБД?
3. Яке правило реляційної СУБД вказує на роль первинного ключу при пошуку інформації в базі даних?
4. Які правила реляційної СУБД визначають відокремлення користувача та прикладної програми від низькорівневої реалізації бази даних?
5. Що визначає правило незалежності умов цілісності?

## Тема «Функціональна модель даних»

## (2 години)

**Мета:** знати призначення функціональної моделі даних.

Ця модель була запропонована Шипменом в 1981 р.

Модель грунтується на положенні про можливість представлення зв'язків між даними, що зберігаються в базі даних, у вигляді математичних функцій. Тому у функціональній моделі даних використовуються два основні поняття: сутність і функція.

Сутність може бути об'єктом реального світу (абстрактна сутність) або бути текстовим рядком або числом (проста сутність). Застосування математичних функцій до конкретної сутності при заданих значеннях аргументів дає однозначний результат.

Діаграми функціональної моделі даних багато в чому аналогічні ER-діаграмам, але зв'язки між ними представлені у вигляді функцій.

**Тема «Модель семантичних об’єктів»**

**(2 години)**

**Мета:** знати призначення моделі семантичних об’єктів.

Модель вперше запропонована Кренке в 1988 р.

База даних є сукупністю семантичних об'єктів. Кожен об'єкт відображає деякий елемент реального світу і характеризується набором атрибутів. Зв'язки між об'єктами представляються атрибутами цих об'єктів.

Розглянемо діаграми семантичних об'є ктів **Магазин**, **Продавец**, **Товар** (рис. 1):


Рис. 1. - Діаграми семантичних об’єктів

Поряд з одним з атрибутів кожного семантичного об'єкту приводиться покажчик ID, що означає, що даний атрибут використовується як ідентифікатор об'єкту. Для позначення унікальності значень ідентифікуючого атрибуту покажчик підкреслюється (для семантичних об'єктів вимога унікальності ідентифікатора не є обов'язковою).

Для кожного атрибуту вказана його **кардинальність** (мінімальна і максимальна кількість входжень цього атрибуту в об'єкт). Наприклад, якщо для атрибуту **Цена** об'єкту **Товар** приводиться кардинальність **1, 1**, це означає, що товар обов'язково повинен мати ціну, і лише одну. Атрибут **Производитель** об'єкту **Товар** має кардинальність **1, N**. Це вказує на те, що один і той же товар може виготовлятися одним або декількома виробниками. Атрибут **Склад** об'єкту **Магазин** має

кардинальність **0, N**. Отже, склад при магазині може бути відсутнім, або їх є декілька. Атрибути, які приймають більш за одне значення, називаються багатозначними.

У об'єкті **Магазин** є груповий атрибут **Адрес**. Атрибути, що входять в його склад, об'єднані дужкою.

У об'єкті **Магазин** існує також атрибут об'єктного типа **Продавец** з кардинальністю **1, N** (див. рис. 1). Це вказує на те, що даний об'єкт пов'язаний з одним або декількома об'єктами **Продавец** (у магазині можуть працювати один або декілька продавців). Для забезпечення зв'язку між даними об'єктами в об'єкт **Продавец** обов'язково повинен входити атрибут **Магазин**, що характеризує цей об'єкт.

Діаграми семантичних об'єктів допускають створення агрегованих об'єктів, підкласів об'єктів.

## Тема «Аномалії оновлення в базі даних»

## (2 години)

**Мета:** вивчення недоліків надмірності даних в базі даних та проблем, які виникають внаслідок надмірних даних.

Головною метою проектування реляційної бази даних є групування атрибутів в відношення таким чином, щоб зменшити надмірність даних та таким чином скоротити об’єм пам’яті, необхідний для фізичного зберігання відношень, які зображуються у вигляді таблиць.

Проблеми, які пов’язані з надмірністю даних, можна простежити, порівнявши відношення «Співробітники» та «Відділення» з відношенням «Співробітники в відділеннях».

**Відношення «Співробітники»**

| Номер співробіт-ника | ПІБ | Адреса співробітника | Посада | Дата народження | ЗП | Номер відділ. |
|---|---|---|---|---|---|---|
| 21 | Філатов Андрій Петрович | Одеса, Вільямса 7, 45 | Менеджер | 01.05.1970 | 2950 | 5 |
| 37 | Нікітіна Ганна Миколаївна | Черкаси, Корольова 67, 3 | Секретар | 16.09.1980 | 1700 | 3 |
| 14 | Федоров Микола Сергійович | Черкаси, б-р Шевченко 19, 5 | Директор | 28.10.1969 | 5500 | 3 |
| 9 | Краснова Олена Валеріївна | Київ, Пушкінська 21, 44 | Бухгалтер | 31.12.1964 | 4200 | 7 |
| 5 | Петренко Оксана Вікторівна | Одеса, Левітана 4, 90 | Менеджер | 03.04.1972 | 2950 | 3 |
| 41 | Васильєва Ганна Семенівна | Одеса, Грецька 47, 15 | Бухгалтер | 18.02.1968 | 4200 | 5 |

**Відношення «Відділення»**

| Номер відділення | Адреса відділення | Телефон відділення |
|---|---|---|
| 5 | Одеса, Ген. Бочарова 19 | 551440 |
| 7 | Київ, Іванова 8 | 285190 |
| 2 | Черкаси, Шевченко 14 | 438615 |
| 4 | Запоріжжя, Гоголя 82 | 648369 |
| 3 | Одеса, Ген. Петрова 1 | 659569 |
| 1 | Луганськ, Київська 67 | 938544 |

**Відношення «Співробітники в відділеннях»**

| Номер співр | ПІБ | Адреса співробітника | Посада | ДН | ЗП | Номер відділ | Адреса відділення | Телефон відділення |
|---|---|---|---|---|---|---|---|---|
| 21 | Філатов Андрій Петрович | Одеса, Вільямса 7, 45 | Менеджер | 01.05. 1970 | 2950 | 5 | Одеса, Ген. Бочарова 19 | 551440 |
| 37 | Нікітіна Ганна Миколаївна | Черкаси, Корольова 67, 3 | Секретар | 16.09. 1980 | 2700 | 3 | Одеса, Ген. Петрова 1 | 659569 |
| 14 | Федоров Микола Сергійович | Черкаси, б-р Шевченко 19, 5 | Директор | 28.10. 1969 | 5500 | 3 | Одеса, Ген. Петрова 1 | 659569 |
| 9 | Краснова Олена Валеріївна | Київ, Пушкінська 21, 44 | Бухгалтер | 31.12. 1964 | 4200 | 7 | Київ, Іванова 8 | 285190 |
| 5 | Петренко Оксана Вікторівна | Одеса, Левітана 4, 90 | Менеджер | 03.04. 1972 | 2950 | 3 | Одеса, Ген. Петрова 1 | 659569 |
| 41 | Васильєва Ганна Семенівна | Одеса, Грецька 47, 15 | Бухгалтер | 18.02. 1968 | 4200 | 5 | Одеса, Ген. Бочарова 19 | 551440 |

Відношення «Співробітники в відділеннях» є альтернативною формою зображення відношень «Співробітники» та «Відділення».

В відношенні «Співробітники в відділеннях» створюються надмірні дані, так як відомості про відділення компанії повторюються в записах про кожного співробітника цього відділення.

В протилежність цьому, в відношенні «Відділення» відомості про відділення компанії створюються лише в одному записі, а в відношенні «Співробітники» повторюється лише номер відділення компанії, який уявляє собою місце роботи кожного співробітника.

При роботі з відношеннями, які мають надмірні дані, можуть виникати проблеми, які називаються **аномаліями оновлення** та діляться на:

1. аномалії вставки;
2. аномалії знищення;
3. аномалії модифікації.

Розглянемо детальніше про кожну з аномалій.

### 1. Аномалії вставки

Існує 2 головних різновиди аномалій вставки, які можна зобразити за допомогою відношення «Співробітники в відділеннях».

**1.** При додаванні відомостей про нових співробітників у відношення «Співробітники в відділеннях» необхідно вказати і відомості про відділення компанії, в якому ці співробітники працюють.

Наприклад, при додаванні відомостей про нового співробітника відділення компанії №7 необхідно ввести відомості про це відділення компанії №7, які повинні співпадати з відомостями про це ж відділення компанії в інших записах відношення «Співробітники в відділеннях».

Відношення «Співробітники» та «Відділення» не постраждають від такого потенційного неспівпадання даних, так як для кожного співробітника в відношенні «Співробітники» необхідно буде ввести тільки номер існуючого відділення компанії. Окрім цього, відомості про відділення компанії з номером №7 записуються в базу даних один раз у вигляді одного рядка відношення «Відділення».

**2.** Для додавання відомостей про нове відділення компанії, яке ще не має своїх власних співробітників, необхідно буде присвоїти визначення Null всім атрибутам переліку персоналу відношення «Співробітники в відділеннях».

Так як атрибут «Номер співробітника» є первинним ключем відношення «Співробітники в відділеннях», то спроба ввести визначення Null в атрибут «Номер співробітника» викличе порушення цілісності сутностей та буде відхилена. Тобто, в відношення «Співробітники в відділеннях» неможливо ввести запис про нове відділення компанії, який містить визначення Null в атрибуті «Номер співробітника».

Структура відношень «Співробітники» та «Відділення» допомагає уникнути виникнення цієї проблеми, так як відомості про відділення компанії заносяться в відношення «Відділення» незалежно від введення відомостей про співробітників. Відомості про співробітників, які будуть працювати в новому відділенні компанії, можуть бути додані в відношення «Співробітники» трохи згодом.

## 2. Аномалії знищення

При знищенні з відношення «Співробітники в відділеннях» запису з інформацією про останнього співробітника деякого відділення компанії, відомості про це відділення компанії будуть повністю знищенні з бази даних.

Наприклад, після знищення з відношення «Співробітники в відділеннях» запису «Краснова Олена Валеріївна» з особовим номером «9» з бази даних неявним чином будуть знищенні всі відомості про відділення компанії з номером 7.

Структура відношень «Співробітники» та «Відділення» допомагає попередити виникнення цієї проблеми, так як відомості про відділення компанії характеризуються окремо від записів з відомостями про співробітників. Пов'язує ці два відношення тільки атрибут «Номер відділення».

При знищенні з відношення «Співробітники» запису з номером співробітника «9» відомості про відділення компанії №7 в відношенні «Відділення» залишаться недоторканими.

### 3. Аномалії модифікації

При спробі змінити значення одного з атрибутів для деякого відділення компанії в відношенні «Співробітники в відділеннях», необхідно оновити відповідні значення в записах для всіх співробітників цього відділення. Якщо не всі записи з даними в відношенні «Співробітники в відділеннях» будуть оновленні, то в такому разі база даних буде мати суперечливі відомості.

Тобто, в нашому прикладі для відділення компанії з номером 3 в записах, які ставляться до різних співробітників, помилково можуть бути вказані різні значення номеру телефону цього відділення.

Всі наведені приклади показують те, що краще зберігати інформацію в окремих відношеннях «Співробітники» та «Відділення», а не в одному відношенні «Співробітники в відділеннях».

**Питання щодо самоконтролю:**

1. Що таке «Надмірність даних»?
2. Що таке «Аномалії оновлення»?
3. Які види аномалій оновлення Ви знаєте?
4. При яких обставинах виникають різні види аномалій оновлення? Довести свою відповідь на своїх прикладах.

**Тема «Історія створення та виникнення мови запитів SQL»**

**(2 години)**

**Мета:** дізнатися про історію розвитку структурованої мови запитів SQL.

**SQL** (**S**tructured **Q**uery **L**anguage) – це структурована мова запитів до реляційних баз даних.

На початку 1970-х років в одній із дослідницьких лабораторій компанії IBM розробили експериментальну реляційну СУБД IBM System R, для якої потім створили спеціальну мову SEQUEL, яка дає змогу відносно просто керувати даними в цій СУБД.

Абревіатура SEQUEL розшифровувалася як Structured English QUEry Language - "структурована англійська мова запитів". Пізніше з юридичних міркувань мову SEQUEL було перейменовано на SQL.

Метою розробки було створення простої непроцедурної мови, якою міг скористатися будь-який користувач, навіть той, хто не має навичок програмування. Власне розробкою мови запитів займалися Дональд Чемберлін і Рейс Бойс.

**Стандарти мови SQL**

Оскільки до початку 1980-х років існувало кілька варіантів СУБД від різних виробників, причому кожен із них мав власну реалізацію мови запитів, було ухвалено рішення розробити стандарт мови, що гарантуватиме переносимість ПЗ з однієї СУБД на іншу (за умови, що вони підтримуватимуть цей стандарт).

В 1983 році Міжнародна організація зі стандартизації (ISO) та Американський національний інститут стандартів (ANSI) приступили до розробки стандарту мови SQL. Стандарт SQL було вперше опубліковано у 1986 р. (SQL-86), він забезпечував мінімальну функціональність.

Оновлювався:

1989 - (SQL-89) механізм підтримки цілісності посилань,

1992 - (SQL-2) розширена функціональність,

1999 - (SQL-3) додано підтримку регулярних виразів, рекурсивних запитів, підтримку тригерів, і деякі об'єктно-орієнтовані можливості.

2003 - (SQL:2003) - введено розширення для роботи з XML-даними, генератори послідовностей і засновані на них типи даних,

2006 – (SQL:2006) - функціональність роботи з XML-даними значно розширено. З'явилася можливість спільно використовувати в запитах SQL і Xquery,

2008 – (SQL:2008) - поліпшено можливості віконних функцій, усунуто деякі невизначеності стандарту SQL:2003.

**Переваги мови SQL**

*Незалежність від конкретної СУБД* Незважаючи на наявність діалектів і відмінностей у синтаксисі, здебільшого тексти SQL-запитів можна досить легко перенести з однієї СУБД в іншу. Існують системи, розробники яких від самого початку орієнтувалися на застосування щонайменше кількох СУБД.

*Наявність стандартів* Наявність стандартів і набору тестів для виявлення сумісності та відповідності конкретної реалізації SQL загальноприйнятому стандарту тільки сприяє "стабілізації" мови.

*Декларативність* За допомогою SQL програміст описує тільки те, які дані потрібно витягти або модифікувати. Те, яким чином це зробити, вирішує СУБД безпосередньо під час обробки SQL-запиту. Однак не варто думати, що це повністю універсальний принцип - програміст описує набір даних для вибірки або модифікації, однак йому водночас корисно уявляти, як СУБД буде розбирати текст його запиту.

**Недоліки мови SQL**

*Невідповідність реляційній моделі даних* Творці реляційної моделі даних Едгар Кодд, Крістофер Дейт та їхні прихильники вказують на те, що SQL не є істинно реляційною мовою.

*Складність* Хоча SQL і замислювався як засіб роботи кінцевого користувача, зрештою він став настільки складним, що перетворився на інструмент програміста.

*Відступи від стандартів* Незважаючи на наявність міжнародного стандарту, багато компаній, що займаються розробкою СУБД (наприклад, Oracle, Sybase, Microsoft, MySQL AB), вносять зміни в мову SQL, застосовувану в розроблюваній СУБД. Таким чином, з'являються специфічні для кожної конкретної СУБД діалекти мови SQL.

*Складність роботи з ієрархічними структурами* Раніше діалекти SQL більшості СУБД не пропонували способу маніпуляції деревоподібними структурами.

**Тема «Робота з оператором Where»**

**(2 години)**

**Мета:** ознайомитися з призначенням та синтаксисом оператора мови SQL Where.

## Обрання рядків з використанням оператора WHERE

В підсумку виконання оператора Select обираються всі рядки таблиці. Але дуже часто треба тим чи іншим чином обмежити набір рядків, які розташовуються в підсумковій таблиці запиту. Це досягається за допомогою оператора **Where**, який має такий формат: **Where критерій пошуку**

Він складається з ключового слова Where, за яким йде перелік умов пошуку, які визначають ті рядки, які повинні бути обрані при виконані запита. Припустимо до 40 виразів, пов'язаних логічними операторами And або Or.

Існує п'ять головних типів умов пошуку:

1. **порівняння** – порівнюються результати обчислення одного виразу з результатами обчислення іншого виразу;
2. **діапазон** (between) – перевіряється, потрапляє або ні результат обчислення виразу в заданий діапазон значень;
3. **належність до множини** (In) – перевіряється, належить або ні результат обчислення виразу до заданої множини значень;
4. **відповідність шаблону** (Like) – перевіряється, відповідає або ні деяке рядкове значення заданому шаблону;
5. **значення Null** – перевіряється, містить або ні даний стовпець визначення Null (невідоме або пусте значення). Використовується як для числових, символьних так й датових полів.

В SQL можна використовувати такі оператори порівняння:

== - рівність; < - менше; <= - менше або рівно;

> - більше; >= - більше або рівно;

<> - нерівність (стандарт ISO);

**!=** - нерівність (використовується в деяких діалектах).

Найбільш складні предикати можуть бути побудовані за допомогою логічних операторів **And**, **Or** або **Not**, а також за допомогою дужок, які використовуються для визначення порядку обчислення виразу.

Обчислення виразу в умовах пошуку виконується за такими правилами:

- вираз обчислюється з лівого боку на правий;
- першими обчислюються підвирази в дужках;
- оператори Not виконуються до виконання операторів And та Or;
- оператори And виконуються до виконання операторів Or.

**Приклад. Порівняння умов пошуку**

Перелічити весь персонал з розміром заробітної плати більш ніж 13500 грн.

```
Select [Номер співробітника],Прізвище,Ім’я,По-батькові,Посада, ЗП
From Співробітники
Where ЗП>13500;
```

**Приклад. Складні умови пошуку**

Перелічити адреси всіх відділень компанії в Одесі або Києві.

```
Select [Номер відділення],Місто,Вулиця,Район,[Поштовий індекс]
From Відділення
Where Місто=‘Одеса’ or Місто=‘Київ’;
```

**Приклад. Використання діапазону Between в умовах пошуку**

Перелічити весь персонал з заробітною платою від 14000 до 14500 грн.

```
Select [Номер співробітника], Прізвище, Ім’я, По-батькові,Посада, ЗП
From Співробітники
Where ЗП Between 14000 and 14500;
```

Цей запит можна записати ще таким чином:

```
Select [Номер співробітника],Прізвище,Ім’я,По-батькові,Посада, ЗП
From Співробітники
Where ЗП>=14000 and ЗП<=14500;
```

**Приклад. Умови пошуку з перевіркою входження в множину (In / Not In)**

Скласти перелік всіх бухгалтерів та менеджерів компанії.

```
Select [Номер співробітника],Прізвище,Ім’я,По-батькові,Посада
From Співробітники
Where Посада In (‘Бухгалтер’, ‘Менеджер’);
```

Існує ще заперечна версія цієї перевірки (Not In), яка використвується для відбору будь-яких значень, окрім тих, які вказані в переліку. Запит можна записати таким чином:

```
Select [Номер співробітника],Прізвище,Ім'я,По-батькові,Посада
From Співробітники
Where Посада='Бухгалтер' Or Посада='Менеджер';
```

Але використання ключового слова **In** уявляє собою найбільш ефективний спосіб запису умов пошуку, особливо якщо набір припустимих значень є дуже великим.

**Приклад. Умови пошуку з вказівкою шаблонів (Like / Not Like)**

Знайти всіх співробітників, які проживають в місті Одеса.

При виконанні цього запита треба організувати пошук рядка «Одеса», який може розташовуватися в будь-якому місці значень стовпця «Адреса» таблиці «Співробітники». В SQL існують два спеціальних символи шаблону, які використовуються при перевірці символьних значень:

1. *** (%)** - символ проценту являє будь-яку послідовність від нуля або більшої кількості символів;
2. **_** - символ підкреслювання являє собою будь-який один символ.

Всі інші символи в шаблоні уявляють лише себе. Наприклад:

- **Адреса Like «К*»** – цей шаблон означає, що першим символом значення обов'язково повинен бути символ **К**, а всі інші символи не уявляють зацікавлення та не перевіряються;
- **Адреса Like «К_ _ _»** – цей шаблон означає, що значення повинне мати довжину, яка дорівнює чотирьом символам, до того ж першим символом обов'язково повинен бути символ **К**;
- **Адреса Like «*а»** – цей шаблон визначає будь-яку послідовність символів довжиною не менш одного символу, до того ж останнім символом обов'язково повинен бути символ **а**;
- **Адреса Like «*Київ*»** – цей шаблон означає, що нас цікавить будь-яка послідовність символів, яка містить підрядок **Київ**;

- **Адреса Not Like «К*»** – цей шаблон вказує на те, що треба знайти будь-які рядки, які не починаються з символу **К**.

```
Select [Номер співробітника],Прізвище,Ім'я,По-батькові,Посада
From Співробітники
Where Адреса Like '*Одеса*';
```

**Приклад . Використання значення Null в умовах пошуку**

Скласти перелік всіх відвідувань здаваємого в оренду объекта нерухомості з номером 36, за якими не было зроблено коментарів.

```
Select [Номер огляду], [Дата огляду]
From Огляд
Where [Номер об'єкта]= 36 and Коментар is null;
```

**Питання щодо самоконтролю**:

1. Для чого призначений оператор Where?
2. Які існують типи умов пошуку?
3. Які оператори порівняння можна використовувати в SQL?
4. Завдяки якому ключовому слову перевіряється, відповідає або ні деяке рядкове значення заданому шаблону?
5. Якими способами можна задати діапазон значень?

## Тема «Сортування результатів - оператор ORDER BY»

## (2 години)

**Мета:** ознайомитися з призначенням та синтаксисом оператора SQL Order by.

Рядки в підсумковій таблиці sql-запиту не підпорядковані яким-небудь чином. Але їх можна необхідним чином відсортувати. Для цього використовується оператор **ORDER BY.**

Оператор ORDER BY вміщує список відокремлених комами ідентифікаторів стовпців, за якими необхідно підпорядкувати підсумкову таблицю sql-запиту. ORDER BY дозволяє підпорядкувати обрані записи в порядку збільшення (Asc) або зменшення (Desc) значень будь-якого стовпця або комбінації стовпців, незалежно від того, присутні ці стовпці в таблиці підсумків або ні.

Оператор ORDER BY завжди повинен бути останнім елементом в операторі Select.

При роботі з оператором ORDER BY можуть використовуватися такі слова:

1. **Top** - повертає відповідну кількість записів, які знаходяться на початку або в кінці діапазону, який описаний за допомогою оператору ORDER BY.

   **Формат: Top N,** де N - ціле значення.

2. **Percent** - повертає відповідний відсоток записів, які знаходяться на початку або в кінці діапазону, який описаний за допомогою оператору ORDER BY.

   **Формат: Top N Percent,** де N - ціле значення.

**Приклад. Сортування за значенням одного стовпця**

Скласти звіт про заробітну плату всіх співробітників компанії, розташувавши рядки в порядку зменшення суми заробітної плати.

Інформація про співробітників компанії може бути представлена відношенням «Співробітники», яке містить стовпці з такими атрибутами:

**Відношення «Співробітники»**

| Код | ПІБ | Адреса | Телефон | Посада | Стать | ДН | ЗП | Номер відділ. |
|---|---|---|---|---|---|---|---|---|
| 21 | Філатов Андрій Петрович | Одеса, Вільямса 7, 45 | 496433 | Менеджер | Ч | 01.05.1970 | 12950 | 5 |
| 37 | Нікітіна Ганна Миколаївна | Черкаси, Корольова 67, 3 | | Секретар | Ж | 16.09.1980 | 12700 | 3 |
| 14 | Федоров Микола Сергійович | Черкаси, б-р Шевченко 19, 5 | 480091 | Директор | Ч | 28.10.1969 | 15500 | 3 |
| 9 | Краснова Олена Валеріївна | Київ, Пушкінська 21, 44 | 735565 | Бухгалтер | Ж | 31.12.1964 | 14200 | 7 |
| 5 | Петренко Оксана Вікторівна | Одеса, Левітана 4, 90 | 489657 | Менеджер | Ж | 03.04.1972 | 12950 | 3 |
| 41 | Васильєва Ганна Семенівна | Одеса, Грецька 47, 15 | 228900 | Бухгалтер | Ж | 18.02.1968 | 14200 | 5 |

```
Select Код, ПІБ, З_П
From Співробітники
Order by ЗП desc;
```

**Приклад. Сортування за деякими стовпцями**

Вивести скорочений перелік об'єктів нерухомості, які здаються в оренду, розташувавши за їх типом. Інформація про об'єкти нерухомості може бути представлена відношенням «Об'єкт нерухомості», яке містить стовпці з такими атрибутами:

**Відношення «Об'єкт нерухомості»**

| Номер об'єкту | Місто | Вулиця | Район | Тип | Кімнати | Орендна плата | Номер власника |
|---|---|---|---|---|---|---|---|
| 1 | Одеса | Ген. Бочарова | Суворовський | Квартира | 2 | 250 у.о. | 45 |
| 2 | Київ | Пушкіна | Центральний | Квартира | 1 | 350 у.о. | 16 |
| 3 | Миколаїв | Київська | Центральний | Будинок | 3 | 300 у.о. | 72 |
| 4 | Одеса | І. Рабіна | Малиновський | Квартира | 1 | 280 у.о. | 11 |

```
Select [Номер об'єкту], Тип, Кімнати, [Орендна плата]
From [Об'єкт нерухомості]
Order by Тип;
```

Для того, щоб підпорядкувати об'єкти нерухомості ще й в порядку зменшення орендної плати, необхідно додатково вказати молодший ключ сортування.

```
Select Номер_об'єкту, Тип, Кімнати, Орендна_плата
From [Об'єкт нерухомості]
Order by Тип, Орендна_плата desc;
```

Після написання такого sql-коду підсумок буде підпорядкований спочатку за типом об'єкту нерухомості, який здається в оренду, а в межах одного типу об'єкту – в порядку зменшення значення орендної плати.

**Питання щодо самоконтролю**:

1. Для чого призначений оператор Order by?
2. За якими видами розрізняють сортування підсумків? Що треба для цього вказати в операторі Order by?
3. Чому оператор Order by повинен бути останім оператором в sql-запиті?
4. Для чого використовуються ключові слова Top та Percent при роботі з оператором Order by?

**Тема «Організація реляційних баз даних. Створення бази даних в СУБД Access»**

**(2 години)**

**Мета:** ознайомитися з призначенням реляційної СУБД MS Access та етапами проектування в ній бази даних. Навчитися створювати нову базу даних в реляційній СУБД MS Access декількома способами. Навчитися створювати таблиці для бази даних в СУБД Access та здійснювати поєднання інформації з декількох таблиць завдяки створенню зв'язків між ними.

## Організація реляційних баз даних

Система управління базою даних MS Access стала найбільш продаваємою в світі системою управління базою даних, тому що в ній оптимальним чином поєднуються міцність та легкість в керуванні.

Вона достатньо міцна та насичена для того, щоб користувачі мали змогу створити з її допомогою закінчені додатки з незначним використанням програмування чи зовсім не програмуючи.

У той же час система керування базою даних Access містить сучасну мову програмування Visual Basic for Application, яка може бути використана для створення найбільш сучасних додатків.

Крім цього, пакет системи керування базою даних Access достатньо легкий в використанні, так що за короткий час новачок має змогу навчитися керувати своїми власними даними за допомогою системи керування базою даних Access.

В Microsoft Access перш ніж створювати таблиці, форми та інші об'єкти необхідно задати структуру бази даних. Добра структура бази даних є основою для створення адекватної вимогам ефективної бази даних.

Виділяють такі етапи проектування бази даних:

1. визначення мети створення бази даних;
2. визначення таблиць, які повинна містити в собі база даних;
3. визначення необхідних в таблиці полів;
4. завдання індивідуального значення кожному полю;
5. визначення зв'язків між таблицями;
6. відновлення структури бази даних;

7. додавання даних та створення інших об’єктів бази даних.

**1. Визначення мети створення бази даних**

На першому етапі проектування бази даних необхідно визначити мету створення бази даних, головні функції та інформацію, яку вона повинна містити. Тобто необхідно визначити головні теми таблиць бази даних та інформацію, яку будуть містити поля таблиць.

База даних повинна відповідати вимогам тих, хто буде безпосередньо з нею працювати. Для цього необхідно визначити теми, які повинна покривати база даних, звіти, які вона повинна видавати, проаналізувати форми, які в теперішню мить використовуються для запису даних.

**2. Визначення таблиць, які повинна містити база даних**

Одним з найбільш важким етапом в процесі проектування бази даних є розробка таблиць, так як результати, які повинна видавати база даних (звіти, вихідні форми та інші) не завжди дають загальну уяву про структуру таблиці.

При проектуванні таблиць зовсім необов’язково використовувати Microsoft Access. Спочатку найкраще розробити структуру на папері. При проектуванні таблиць рекомендується керуватися наступними головними принципами:

- інформація в таблиці не повинна дублюватися. Не повинно бути повторень та й між таблицями.

Коли відповідна інформація зберігається тільки в одній таблиці, то змінювати її доведеться тільки в одному місці. Це робить роботу більш ефективною, а також виключає можливість неспівпадіння інформації в різних таблицях. Наприклад, в одній таблиці повинні міститися адреси та телефони клієнтів.

- кожна таблиця повинна містити інформацію тільки на одну тему.

Відомості на кожну тему опрацьовуються набагато легше, ніж якщо б містилися в незалежних одна від одної таблицях. Наприклад, адреси та закази клієнтів зберігаються в різних таблицях, з тим щоб при знищенні заказу інформація про клієнта залишалась в базі даних.

**3. Визначення необхідних в таблиці полів**

Кожна таблиця містить інформацію на окрему тему, а кожне поле в таблиці містить окремі відомості по темі таблиці. Наприклад, в таблиці з даними про клієнта можуть міститися поля з назвою компанії, адресою, країною та номером телефону.

При розробці полів для кожної таблиці необхідно пам’ятати:

1. кожне поле повинне бути пов’язане з темою таблиці;
2. в таблиці повинна бути присутня вся необхідна інформація;
3. інформацію слід розбивати на найменші логічні одиниці (наприклад, поля «Ім’я» та «Прізвище», а не загальне поле ПІБ).
4. **Завдання індівідуальних значень кожному полю**

З тим щоб Microsoft Access мав змогу зв’язати дані з різних таблиць, наприклад, дані про клієнта та його закази, кожна таблиця повинна містити поле чи набір полів, які будуть задавати індивідуальні значення кожному запису в таблиці. Таке поле чи набір полів називається **головним ключем.**

5. **Визначення зв’язків між таблицями**

Після розподілу даних по таблицях та визначення ключових полів необхідно обрати схему для зв’язку даних в різних таблицях. Для цього потрібно визначити зв’язки між таблицями.

**6. Відновлення структури бази даних**

Після проектування таблиць, полів та зв’язків необхідно ще раз проглянути структуру бази даних та виявити можливі недоліки. Бажано це зробити на цьому етапі, доки таблиці не заповнені даними.

Для перевірки необхідно створити декілька таблиць, визначити зв’язки між ними та занести декілька записів до кожної таблиці, потім продивитися, відповідає база даних поставленим вимогам.

Рекомендується також створювати чернеткові вихідні форми та звіти та перевіряти, видають вони потрібну інформацію чи ні. Крім того необхідно виключити з таблиць всі можливі повторювання даних.

**7. Додавання даних та створення інших об’єктів бази даних.**

Якщо структури таблиць відповідають поставленим вимогам, то можна заносити дані. Потім можна створювати запити, форми, звіти, макроси та модулі.

*Запити* використовуються для відбору даних з бази даних. Відбір даних можливо здійснювати на таких мовах як: SQL та QBE.

*Форми* використовуються для введення та модифікації даних в базі даних.

Звичайно користувачі працюють з формами, а не з таблицями. Форми дають можливість більш художньо відобразити дані в базі даних.

*Звіти* використовуються для видачі даних на принтер.

*Макроси* використовуються для автоматизації виконання окремих операцій над базою даних.

*Модулі* – це створення додатків з використанням мови VBA (**V**isual **B**asic for **A**pplication).

## Створення бази даних в СУБД MS Access

Для того щоб запустити MS Access необхідно виконати наступні дії:

1. на «Линейке задач» Windows (Task Bar) натиснути «Пуск» (Start) та перемістити покажчик миші на пункт «Программы» (Programs);
2. в меню «Программы» (Programs) натиснути на опції «Microsoft Access».

Після запуску MS Access відображає діалогове вікно, яке дозволяє створити нову базу даних або відкрити вже існуючу.

Базу даних можна створювати двома способами:

1. створити пусту базу даних, а потім створювати таблиці та інші об’єкти, які будуть необхідні;
2. використовувати один з можливих шаблонів бази даних, які постачаються разом з MS Access, та використавши «Мастер базы данных», зробити його відповідним пред’явленим вимогам.

При створенні пустої бази даних або використанні «Мастера базы даннях», MS Access відображає діалогове вікно «Файл новой базы даннях», для того, щоб дати можливість визначити ім’я нової бази даних та папку, де буде знаходитися нова база даних.

Для того щоб дати ім’я новій базі даних, необхідно ввести ім’я в поле «Имя файла» («File Name»). Для імен файлів потрібно використовувати такі ж самі правила, що й при призначенні імен іншим файлам в Windows: довжина імені не повинна перевищувати 256 символів, та в ім’я можна вміщувати пробіли.

MS Access автоматично присвоїть файлу розширення .mdb (**M**icrosoft **D**ata**B**ase). MS Access пропонує для нових баз даних, які створюються, такі імена, як db1.mdb або db2.mdb, але бажано використовувати більш обгрунтовані імена для того, щоб легше було розрізняти файли.

## Створення таблиць та зв'язків між таблицями в СУБД Access

Після того як база даних буде створена, необхідно приступити до створення таблиць. Відкрите вікно бази даних має в якості означеної за умовчуванням закладку «Таблиці», яка з'явилась зверху всіх інших закладок, розташованих у верхній частині вікна бази даних.

Якщо натиснути по кнопці «Создать» («New») MS Access відобразить діалогове вікно «Новая таблица» («New Table»), яке пропонує обрання одного з наступних методів створення таблиці:

1. **режим таблицы** («Datasheet View») створює нову таблицю та відображає її в режимі таблиці таким чином, що в неї одразу можна вводити дані;
2. **конструктор** («Design View») відображає нову таблицю в режимі конструктору, що дає можливість задавати поля за допомогою спеціальних методів;
3. **мастер таблиц** («Table Wizard») дозволяє створювати таблицю за допомогою майстру;
4. **импорт таблиц** («Import Table»)
5. **свіязь с таблицами** («Link Table»)

} дозволяє створювати спеціальні типи таблиць, які використовуються для роботи з даними з інших додатків.

## Визначення полів

Якщо обрати «Режим конструктору» в діалогову вікні «Новая таблица», MS Access відобразить вікно «Таблица» в режимі конструктору, де для кожного поля в окремості треба ввести всю необхідну інформацію, тобто заповнити стовбці **«Имя поля»** («Field Name»), **«Тип данных»** («Data Type») та **«Описание»** («Description»).

1. **«Имя поля»** – необхідно використовувати ті ж самі правила присвоєння імен для полів, що для інших об'єктів MS Access. Імена можуть вміщувати в себе до 64 символів.
2. **«Тип данных»** - MS Access, як й інші системи керування базами даних потребує визначення типу даних, які будуть міститися в кожному полі.

Є можливість обрати наступні типи даних:

- **«Текстовый»** («Text») – розмір – до 255 символів, включаючи букви, цифри та спеціальні символи.

- **«Поле Memo»** («Memo») – розмір – до 65000 символів. На відміну від текстових полів, поле з даними цього типу має змінну довжину та користувач не завдає її максимальний розмір.
- **«Числовой»** («Number») – цей тип містить числові дані, які використовуються в обчисленнях. Тип чисел, які містить поле, та точність обчислень залежить від розміру, який користувач завдасть числовому полю. Деякі числові поля містять тільки цілі числа, інші можуть містити числа з багатьма десятковими знаками.
- **«Дата/Время»** («Date/Time») – містить дати та час. В залежності від формату, який користувач присвоює полю, можливо вводити в нього календарні дані чи значення часу в тій чи іншій формі.
- **«Денежный»** («Currency») – містить числа, які використовуються в грошових обчислюваннях, в обчислюваннях с точністю до чотирьох цифр праворуч від десяткової коми.
- **«Счётчик»** («AutoNumber») – містить послідовні числа, які Access вводить автоматично. Access розміщує число 1 в цьому полі в перший запис, який користувач вводить в таблицю, число 2 – в другий запис та т.п. Неможливо змінити числа, які Access вводить в це поле.
- **«Логический»** («Yes/No») – використовується для зберігання тільки двох значень, визначаємих як ІСТИНА/НЕПРАВДА, ТАК/НІ чи ВІМКНУТИ/ВИМКНУТИ, в залежності від формату, який користувач задає даному полю в панелі «Свойства поля» («Properties»).

За умовчуванням поля ТАК/НІ відображаються в таблицях у вигляді вимикачей. Коли вимикач є активним, тобто позначений, то значення дорівнює ІСТИНА (чи ТАК, чи ВІМКНУТИ), а коли неактивний, значення дорівнює НЕПРАВДА (чи НІ, чи ВИМКНУТИ).

- **«Поле обекта OLE»** («OLE Object») – цей тип даних вміщує дані з інших додатків, які підтримують технологію OLE (**O**bject **L**inking And **E**mbedding) – зв’язок та впровадження об’єктів.

Це поле може бути використане для приєднання зображень, звукових файлів та даних іншого типу, які доступні з іншого будь-якого windows-додатку, який підтримує OLE.

- **«Гіперссылка»** («HyperLink») – це поле містить адресу об’єкту, документу або web-сторінки, який можна вивести на екран звичайним натиском на відповідному полі. Можна також вміщувати ці поля в форми та звіти.

Наприклад, якщо існує таблиця, в якій містяться дані про компанії, які є клієнтами деякої фірми, можна розмістити в неї поле гіперзноски з адресою сторінки кожної компанії. Дані такої сторінки може відобразити будь-який користувач звичайним натисканням по цьому полю в таблиці або в формах введення даних, які містять це поле.

- **«Мастер подстановок»** («Lookup Wizard») – дозволяє створювати поле, за допомогою якого користувач має змогу обрати значення з переліку.

MS Access автоматично здійснює програмну перевірку внесення даних на тип даних. Наприклад, в числові поля можна заносити тільки числа, а в поля з датою – тільки дійсні календарні дати. Якщо користувач спробує в поле ввести дані не того типу, вони не будуть прийняті та користувач отримає від MS Access повідомлення про помилку.

3. **«Описание»** – розділ опису є необов’язковим для заповнення, але інформація, яка буде внесена в цей розділ відображається в рядку стану при внесенні даних для окремого поля, полегшуючи процес внесення.

**Визначення первинного ключу**

MS Access має змогу видобувати дані швидше, якщо таблиця має одне поле в якості первинного ключу. Це ключове поле завжди повинне бути присутнім при роботі з базою даних.

**Ключове поле** – це поле або декілька полів, які ідентифікують кожний запис в таблиці.

Найбільш поширеним первинним ключем є послідовно пронумероване поле, таке, як, наприклад, «Номер співробітника», якому надано унікальне значення для кожного запису в таблиці. Це поле використовується в якості первинного ключу.

Краще за все при створюванні первинного ключу використовувати поле, яке містить тип даних «Счётчик».

Для того, щоб створити первинний ключ, треба спочатку описати це поле в режимі конструктору, як будь-яке інше поле. Потім виділити це поле, після чого або обрати опцію «Ключевое поле» («Primary Key») з меню «Правка», або натиснути на панелі інструментів «Ключевое поле».

MS Access виводить на екран зображення ключу в лівій частині поля первинного ключу.

MS Access може створювати поле первинного ключу, коли користувач не створив його самостійно. Коли зачиняється вікно «Таблиця», MS Access виводить на екран діалогове вікно та запрошує підтвердження на визначення поля первинного ключу. Необхідно натиснути «Да», якщо треба, щоб MS Access створив це поле.

Якщо одне з полів таблиці має тип даних «Счётчик», то MS Access призначить його в якості первинного ключу. Та навпаки, MS Access додасть в таблицю нове поле з типом даних «Счётчик» та присвоїть йому ім’я «Код».

**Визначення властивостей полів**

СКБД MS Access надає засоби для вказівки додаткових обмежень для полів таблиці, які вводяться в секції «Свойства поля» вікна «Таблиця» в режимі конструктору.

З кожним полем таблиці пов’язаний набір властивостей, які можуть бути використані для налагоджування засобу зберігання даних в полі, методів роботи з полем та способу його відображення на екрані. Наприклад, розміщуючи необхідне значення в властивості «Розмер поля» («Field Size») можна завдавати максимальну кількість символів, яку припускається вводити в конкретне поле типу «Текстовый» («Text»).

Тип даних поля визначає набір властивостей, які будуть припустимі при опису цього поля. Доступ до властивостей полів здійснюється обранням необхідного поля в верхній секції вікна «Таблица» в режимі конструктору, після чого з’явиться можливість обрання необхідних властивостей в нижній секції цього вікну.

Встановлення значень властивостей полів в вікні «Таблица» в режимі конструктору гарантує, що поля будуть мати коректні значення властивостей при використанні їх описів у процесі створення форм або звітів.

1. **Властивість «Розмер поля»** («Field Size») – є доступним тільки полям з типом

даних «Текстовый», «Числовой» або «Счётчик» та призначений для завдання максимальної довжини даних, які можуть зберігатися в цьому полі.

2. **Властивість «Формат поля»** («Format») – призначена для визначення способу відображення та виведення на друк полів, які містять числа, дату, відмітки часу та текстові дані.

СУБД MS Access надає широкий діапазон форматів для зображення даних різних типів. Наприклад, поля, які містять дату/час, можуть відображати дати в різних форматах – короткому («Short Date»), середньому («Medium Date») або довгому («Long Date»). Дата «21 листопада 2013 року» може бути зображена таким чином «21.11.13» (короткий формат), «21-лист-13» (середній формат) або «21 листопада 2013» (довгий формат).

3. **Властивість «Число десятичных знаков»** («Decimal Places») – відноситься тільки до полів з числовим або грошовим типом даних та призначена для вказівки кількості дрібних десяткових знаків, які повинні виводитися при відображенні чисел.

4. **Властивість «Маска ввода»** («Input Mask») – використовується при внесені даних в таблицю для організації контролю формату значень, які вводяться. Маска визначає тип символу, який є припустимим в кожній позиції вхідного поля. Крім цього, маска введення може спрощувати внесення даних за рахунок автоматичного розміщення спеціальних форматуючих символів в необхідні позиції та генерації повідомлень про помилки при спробі внесення некоректної інформації.

СУБД MS Access надає широкий діапазон символів маски, які дозволяють ефективно керувати процесом внесення даних.

**Символи масок внесення**

| Символ | Значення |
|---|---|
| **0** | Необхідна цифра. |
| **9** | Можна ввести цифру або пробіл, але не обов'язково. |
| **#** | Може бути занесена цифра, пробіл, «+» або «-», але не обов'язково. |
| **L** | Повинна бути введена буква. |
| **?** | Може бути введена буква, але не обов'язково. |
| **A** | Повинна бути введена буква або цифра. |
| **A** | Може бути введена буква або цифра, але не обов'язково. |
| **&** | Може бути введений будь-який символ або пробіл. |
| **C** | Може бути введений будь-який символ або пробіл, але не обов'язково. |
| < | Перетворює всі символи праворуч до нижнього регістру. |
| > | Перетворює всі символи праворуч до верхнього регістру. |
| **!** | Вказує на те, що маску потрібно заповнювати з правої на ліву сторону, а не навпаки. Цей символ можна використовувати, якщо символи ліворуч є необов'язковими. ! можна використовувати в будь-якій позиції маски. |
| \ | Наступний символ слід уявляти буквально, а не у вигляді коду. |

5. **Властивість «Подпись»** («Caption») – використовується для внесення найбільш повного опису імені полю або для розміщення іншої корисної інформації, які відображаються в заголовках різних уявлень.

Наприклад, якщо властивості «Подпись» поля «Н готелю» дати значення «Номер готелю», то саме це значення буде видано в заголовок відповідного стовпця сітки даних замість імені полю.

6. **Властивість «Значение по умолчанию»** («Default Value») – з метою прискорення процесу внесення даних та зменшення кількості можливих помилок, для будь-якого полю можна вказати значення, які будуть автоматично розміщуватися в відповідне поле внесення при створенні нового запису.

Ця властивість не може бути застосована для даних типу «Счётчик» та «Поле объекта OLE».

7. **Властивість «Условие на значение»** («Validation Rule») та **«Сообщение об**

**ошибке»** («Validation Text»).

Властивість «Условие на значение» призначена для завдання вимог, яким повинне відповідати внесене в поле значення. При внесенні даних, для яких встановлені правила перевірки не виконуються, користувачу відображається повідомлення, яке містить текст, розташований в властивості «Сообщение об ошибке».

Правила перевірки можуть використовуватися для визначення діапазону припустимих значень числових полів та датових полів. Завдання схожих обмежень суттєво зменшує кількість помилок, які можуть винукнути при внесенні даних в таблицю.

8. **Властивість «Обязательное поле»** («Required»)

Обов'язковими називають такі поля, які повинні містити конкретні значення в будь-якому з записів таблиці. Якщо властивості «Обязательное поле» деякого поля дати значення «Так» («Yes»), це буде значити, що це поле є обов'язковим. В підсумку, при внесенні нових записів в таке поле обов'язково необхідно буде внести значення, яке відрізняється від Null. Таким чином, встановлення того чи іншого значення властивості «Обязательное поле» є еквівалентним дозволу або забороні розміщення в це поле визначення Null.

Так як поля первинних ключів грають особливу роль, висувається жорстка вимога: ключові атрибути завжди повинні бути обов'язковими полями та не повинні містити пустих значень. При створенні таблиці для кожного поля за умовчуванням приймається, що воно не є обов'язковим.

9. **Властивість «Пустые строки»** («Allow Zero Length») – використовується для вказівки того, припускається розміщувати в текстове поле символьний рядок нулевої довжини («  »).

Ця властивість приймається тільки для полів типу «Текстовый», «Поле Memo» та «Гіперссылка».

Якщо потрібно, щоб СКБД MS Access при занесенні в деяке поле пустого значення розташовувала в таблицю замість значення Null строку нулевої довжини, то обом властивостям «Пустые строки» та «Обязательное поле» необхідно встановити значення «Так» («Yes»).

Але властивість «Пустые строки» працює незалежно від властивості «Обязательное поле». Останнє призначене тільки для вказівки того, припускається чи ні розміщення в даному полі значення Null. Якщо властивості «Пустые строки» надано значення «Так», то строки нулевої довжини будуть вважатися припустимим значенням для даного поля незалежно від стану його властивості «Обязательное поле».

**10. Властивість «Індексированое поле»** («Indexed») – може використовуватися для створення індексів для одного поля. Наявність індексу прискорює виконання запитів до проіндексованого поля, а також операції сортування та групування.

**Встановлення зв'язків між таблицями**

Для того щоб встановити зв'язок між таблицями необхідно виконати команду «Схема данных» з меню «Сервіс»:

1. З'явиться вікно «Схема данных». Якщо зв'язок встановлюється вперше, то воно буде містити діалогове вікно «Добавление таблицы». Якщо вікно «Добавление таблицы» відсутнє, його можна відкрити, обравши команду «Добавить таблицу» з меню «Связи» або обравши піктограму «Добавить таблицу».
2. Обрати таблицю, яка буде використовуватися для встановлення зв'язків, потім виконати натискання по кнопці «Добавить», для додавання таблиці в вікно «Схема данных».
3. Повторити дії, які були описані в пункті №2 для кожної таблиці, яка приймає участь в встановленні зв'язку.
4. Для створення зв'язків між таблицями перемістити поле (або поля), які необхідно зв'язати на відповідне поле іншої таблиці. В більшості зв'язків ключове поле першої таблиці зв'язується з аналогічним полем другої таблиці. Після переміщення поля з'явиться діалогове вікно «Связи».
5. В діалоговому вікні зображені назви таблиць, між якими встановлюються зв'язки та імена полів для зв'язку. Полям, завдяки яким створюються зв'язки між таблицями, необов'язково мати однакові імена, але вони повинні бути одного типу. Виключення складають поля лічильників, які можна пов'язувати з числовими полями.

6. Для автоматичної підтримки цілісності бази даних треба встановити прапорець «Обеспечение целостности даннях». Крім цього прапорця в вікні зображені й інші:
    - *Каскадное обновление связанных полей.* При ввімкненні даного режиму зміни, зроблені в пов'язаному полі першої таблиці, автоматично заносяться в поля пов'язанної таблиці, яка містить ті ж самі дані.
    - *Каскадное удаление связанных полей.* При ввімкненні даного режиму знищення записів в першій таблиці призводить до знищення відповідних записів пов'язанної.
7. Виконати натискання по кнопці «Создать». Потім закрити вікно «Связи». При запиті про збереження зв'язку виконати натискання по кнопці «Да».

**Питання щодо самоперевірки:**

1. Для чого при проектуванні бази даних потрібно задавати структуру?
2. На які етапи поділяється проектування бази даних?
3. Який етап проектування бази даних є самим першим та що на ньому визначається?
4. Якими принципами рекомендується керуватися при проектуванні таблиць бази даних?
5. За що відповідає етап визначення зв'язків між таблицями?
6. Що треба зробити після проектування таблиць, полів та зв'язків?
7. Які об'єкти, окрім таблиць, можна створювати при роботі з базою даних?
8. Які дії треба виконати для того щоб запустити MS Access?
9. Як створити нову базу даних в MS Access?
10. Якими способами можна створити базу даних MS Access?
11. Як присвоїти новій базі даних ім'я?
12. Яке розширення мають бази даних, створені в MS Access?
13. Для чого призначена закладка «Таблиці» в вікні бази даних?
14. Якими методами можна створити таблицю в MS Access?
15. Для чого потрібні стовбці «Ім'я поля», «Тип даних» та «Опис» при визначенні полів? Який з них є необов'язковим для заповнення?
16. Якими типами даних можна користуватися при роботі з полями таблиць бази даних MS Access?

17. Чим «Поле Memo» відрізняється від текстових полів?
18. Який тип даних дозволяє створювати поле, за допомогою якого користувач має змогу обрати значення з переліку.
19. Що таке «Ключове поле»? Яким чином можна створити ключове поле в MS Access? Який тип даних найчастіше використовується для ключу, призначеного в якості ключового?
20. Для чого потрібна секція „Властивості поля" вікна „Таблиця" в режимі конструктору?
21. Як встановити зв'язок між таблицями в СУБД Access?

## Тема «Створення запитів в СУБД MS Access»

## (2 години)

**Мета:** навчитися створювати запити в СУБД MS Access.

Запит дозволяє вилучати дані з бази даних. Виконання запитів нагадує завдання питання таблиці бази даних.

Запит до існуючої в базі даних інформації треба сконструктувати таким чином, щоб вказати СКБД, які саме дані нас цікавлять. Частіше за все використовується тип запитів, який називають **запитами на відбір**. Вони дозволяють продивлятися, аналізувати або вносити зміни в дані, збережені в одній або декількох таблицях.

При виконанні запиту на відбір СКБД MS Access розміщує обрані дані в **динамічний набір даних** (Dynaset), який уявляє собою динамічно створюване уявлення (вигляд), яке містить дані, які вилучені з однієї або декількох таблиць.

Дані обираються та сортуються у відповідності з вказаними в запиті вимогами. Іншими словами, динамічний набір даних уявляє собою оновлений набір записів, визначений таблицею або запитом, який можна роздивлятися як окремий об'єкт.

Крім запитів на відбір, в середовищі СУБД MS Access може бути створено й багато інших корисних типів запитів.

**Типи запитів, які підтримуються в СУБД MS Access**

| № | Тип запиту | Опис |
|---|---|---|
| **1** | Запити на відбір | Містять формування запиту до бази даних, яке визначається як набір критеріїв для відбору даних з однієї або більш таблиць. |
| **2** | Запити з узагальненням | Передбачає виконання обчислювань з використанням даних з деякої групи записів. |
| **3** | Запити з параметром | Виконання цих запитів супроводжується виведенням одного або більше раніш визначених діалогових вікон, призначених для внесення користувачем конкретних значень параметрів запиту. |
| **4** | Запити на відбір дублікатів | Виконують пошук дублюючих записів в межах однієї таблиці. |
| **5** | Запити на відбір записів, які не мають відповідності | Працюють зі зв'язаними таблицями та виконують пошук записів однієї таблиці, які не мають відповідностей в іншій таблиці. |
| **6** | Перехресні запити | З їх допомогою великий об'єм даних може бути підрахований та зображений у форматі невеликої електронної таблиці. |
| **7** | Запити з автопідставлянням | При виконанні запиту виконується автоматичне підставляння зазначених значень до записів, які знов створюються. |
| **8** | Активні запити (запити на знищення, додавання, оновлення та створення таблиці) | Дозволяють за одну операцію внести зміни в багату кількість записів. зміни передбачають знищення, додавання або оновлення записів в таблиці, а також створення нової таблиці. |
| **9** | Специфічні запити | Цей тип запитів використовується для модифікації запитів, про які було розказано вище та для визначення властивостей форм та звітів. |

На початку процедури створення нового запиту треба в вікні бази даних обрати закладку «Запросы» та натиснути кнопку «Создать». Після цього СУБД MS Access відображає діалогове вікно «Новый запрос» («New Query»). Зображений в цьому вікні перелік припустимих варіантів подальших дій дозволяє почати створення нового запиту з нуля та виконати всі необхідні дії самостійно (варіант «Конструктор»), або скористатися для створення запиту допомогою одного з майстрів СУБД MS Access, назви яких складають частину списку, що залишилася.

Майстри представляють собою один з варіантів допоміжних програм бази даних. Вони завдають користувачам ряд питань про створюваний запит, після чого генерують його текст на отриманих відповідях.

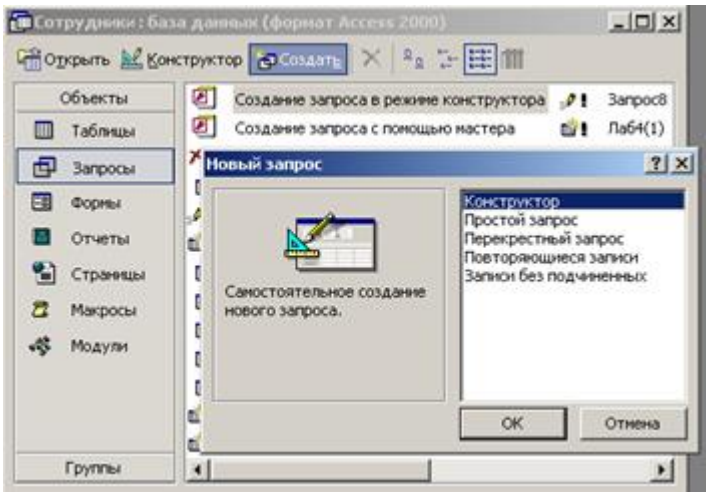



**Рисунок 1** – Вікно «Новый запрос»

Після обрання режиму конструктору з’явиться діалогове вікно «Запрос на выборку», яке складається з двох панелей.

Панель, розташована зверху містить список усіх полів таблиці. Нижня панель вікна запиту містить область, яка має назву **бланк запиту**.

Одночасно з діалоговим вікном «Запрос на выборку» активується вікно «Добавление таблицы» з закладками «Таблицы», «Запросы» та «Таблицы и запросы», яке необхідно закрити за допомогою вікна «Закрыть». Далі треба в меню «Вид» обрати пункт «Режим SQL» або натиснути по кнопці «SQL» на панелі інструментів. В разі виконання цих дій MS Access відобразить вікно «Запрос1: запрос на выборку», де можна прописувати sql-код запиту.

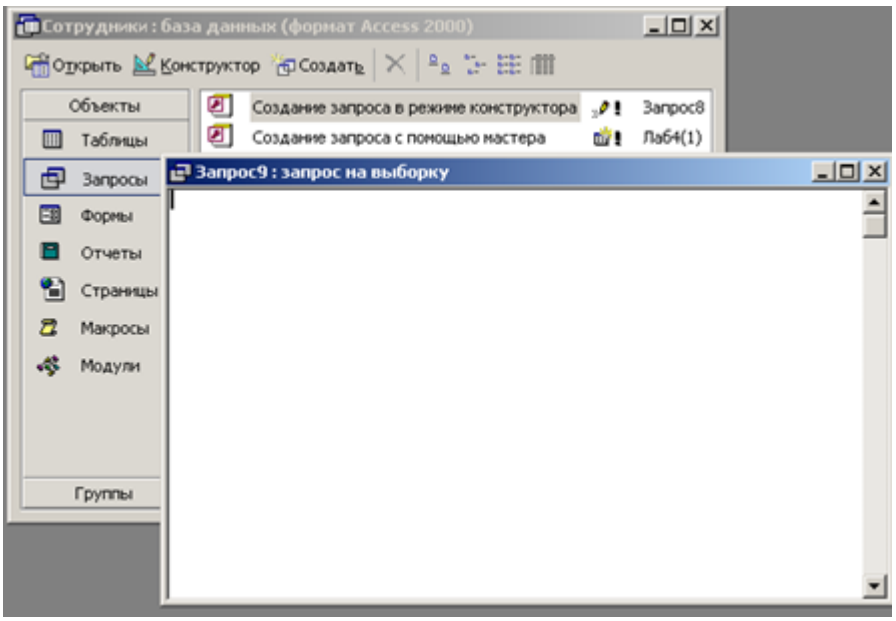



**Рисунок 2** - Вікно «Запрос1: запрос на выборку»

Після завершення заповнення бланку запиту для отримання динамічного набору даних необхідно запустити запит на виконання. Зробити це можна, обравши опцію «Запуск» з меню «Запрос» або натиснувши на панелі інструментів по кнопці «Запуск» з зображенням знаку оклику.

Запит можна запустити на виконання, перейшовши в «Режим конструктора». Треба обрати з меню «Вид» опцію «Режим таблицы» або натиснути на панелі інструментів «Режим таблицы». Після короткої паузи, впродовж якої MS Access виконає запит, на екран буде виведена таблиця з полями та записами, які вказані в запиті, та відсортованими в належному порядку.

**Питання щодо самоперевірки:**

1. Для чого потрібен запит в базі даних?
2. Які типи запитів підтримуються в СУБД Access?
3. Який тип запиту використовується частіше за всіх типів запитів?
4. Виконання якого типу запиту супроводжується виведенням одного або більше раніш визначених діалогових вікон, призначених для внесення користувачем конкретних значень параметрів запиту?
5. Якими способами можна створити запит в СУБД Access?
6. Що таке «Бланк запроса»?
7. Для чого призначене вікно «Добавление таблицы» та одночасно з яким діалоговим вікном воно активується?
8. Як запустити запит на виконання?

**Тема «Внутрішні запити»**

**(2 години)**

**Мета:** познайомитися з типами внутрішніх запитів та навчитися їх створювати

**Внутрішний запит** – це оператор SELECT, впроваджений в тіло іншого оператора SELECT.

Зовнішній (другий) оператор SELECT використовує результат виконання внутрішнього (першого) оператора для визначення змісту кінцевого результату всієї операції.

Внутрішні запити можуть бути розміщені в операторах WHERE та HAVING зовнішнього оператора SELECT – в цьому випадку вони отримують назву підзапитів або вкладених запитів. Окрім цього внутрішні оператори SELECT можуть використовуватися в операторах INSERT, UPDATE, DELETE.

Існує три типи підзапитів:

1. **скалярний підзапит** – повертає значення, яке обирається з перетину одного стовпця з одним рядком, тобто одне значення. Скалярний підзапит може використовуватися всюди, де треба визначити лише одне значення.
2. **рядковий підзапит** - повертає значення декількох стовпців таблиці, але у вигляді одного рядка. Рядковий підзапит може використовуватися скрізь, де використовується конструктор рядкових значень – взагалі це предикати.
3. **табличний підзапит** - повертає значення одного або більшої кількості стовпців таблиці, які розміщені в більш ніж одному рядку. Табличний підзапит може використовуватися усюди, де припускається вказувати таблицю – наприклад, як операнд предиката IN.

**Приклад. Використання підзапиту з перевіркою на рівність**

Скласти перелік персоналу, який працює в відділенні компанії, яке розташоване на вулиці Катеринінська №15.

Наприклад, інформація про відділення компанії може бути описана відношенням «Відділення», яке містить стовпці з атрибутами «Номер відділення», «Місто», «Вулиця», «Район», «Поштовий індекс», «Телефон».

Аналогічно, інформація про співробітників компанії може бути зображена відношенням «Співробітники», яке містить стовпці з атрибутами «Номер

співробітника», «ПІБ», «Адреса», «Телефон», «Посада», «Стать», «Дата народження», «ЗП», «Номер відділення».

**Відношення «Відділення»**

| Номер відділення | Місто | Вулиця | Район | Поштовий індекс | Телефон |
|---|---|---|---|---|---|
| 5 | Одеса | Ген. Бочарова 19 | Суворівський | 65135 | 551440 |
| 7 | Київ | Іванова 8 | Центральний | 87505 | 285190 |
| 2 | Черкаси | Шевченко 14 | Центральний | 43701 | 438615 |
| 4 | Запоріжжя | Гоголя 82 | Центральний | 50883 | 648369 |
| 3 | Одеса | Катеринінська 15 | Приморський | 65120 | 659569 |
| 1 | Луганськ | Київська 67 | Північний | 49913 | 938544 |

| Номер спів. | ПІБ | Адреса | Тел | Поса-да | Стать | ДН | ЗП | Номер відділ. |
|---|---|---|---|---|---|---|---|---|
| 21 | Філатов Андрій Петрович | Одеса, Вільямса 7, 45 | 496433 | Менед-жер | Ч | 01.05. 1970 | 2950 | 5 |
| 37 | Нікітіна Ганна Миколаївна | Черкаси, Корольова 67, 3 | | Секре-тар | Ж | 16.09. 1980 | 2700 | 3 |
| 14 | Федоров Микола Сергійович | Черкаси, б-р Шевченко 19, 5 | 480091 | Дирек-тор | Ч | 28.10. 1969 | 5500 | 3 |
| 9 | Краснова Олена Валеріївна | Київ, Пушкінська 21, 44 | 735565 | Бухгал-тер | Ж | 31.12. 1964 | 4200 | 7 |
| 5 | Петренко Оксана Вікторівна | Одеса, Левітана 4, 90 | 489657 | Менед-жер | Ж | 03.04. 1972 | 2950 | 3 |
| 41 | Васильєва Ганна Семенівна | Одеса, Грецька 47, 15 | 228900 | Бухгал-тер | Ж | 18.02. 1968 | 4200 | 5 |

**Відношення «Співробітники»**

```
Select [Номер співробітника], ПІБ, Посада
From Співробітники
Where [Номер відділення] =
        (Select  [Номер відділення]
         From Відділення
         Where Вулиця = «Катеринінська 15»);
```

Внутрішній оператор SELECT

```
(Select  Номер_відділення
 From Відділення
 Where Вулиця = «Катеринінська 15»);
```

призначений для визначення номеру відділення компанії, розташованого за адресою «Катеринінська 15» (існує лише одне відділення компанії, тому даний приклад є прикладом скалярного підзапиту).

Після отримання номеру необхідного відділення компанії виконується зовнішній підзапит, призначений для відбору відомостей про співробітників цього відділення компанії. Тобто, внутрішній оператор SELECT повертає таблицю з одним значенням «3». Воно являє собою номер того відділення компанії, яке розташоване в будинку №15 на вулиці Катеринінська.

В результаті зовнішній оператор SELECT має такий вигляд:

```
Select [Номер співробітника], ПІБ, Посада
From Співробітники
Where [Номер відділення] = 3;
```

| Код | ПІБ | Посада |
|---|---|---|
| 37 | Нікітіна Ганна Миколаївна | Секретар |
| 14 | Федоров Микола Сергійович | Директор |
| 5 | Петренко Оксана Вікторівна | Менеджер |

Підзапит уявляє собою інструмент створення тимчасової таблиці, зміст якої вилучається та обробляється зовнішнім оператором. Підзапит може вказуватися після операторів порівняння (тобто операторів =, <, >, <=, >=, < >) в операторах Where та Having. Текст підзапиту повинен бути розміщений у дужках.

**Приклад. Використання підзапитів з функціями**

Скласти перелік всіх співробітників компанії, які мають заробітну плату більш ніж середня заробітна плата, з вказівкою того, наскільки їх заробітна плата перевищує середню заробітну плату в цілому по компанії.

```
Select [Номер співробітника], ПІБ, Посада, ЗП –
        (Select Avg(ЗП)
         From Співробітники) As ЗП1
From Співробітники
Where ЗП >
        (Select Avg(ЗП)
         From Співробітники);
```

Потрібно відзначити, що неможливо напряму використовувати «Where ЗП > Avg(ЗП)», так як використовувати функції в операторі Where заборонено. Для досягнення необхідного результату потрібно створити підзапит, який підрахує середнє значення заробітної плати, а потім використовувати його в зовнішньому

операторі Select, призначеному для відбору відомостей про тих співробітників компанії, в яких заробітна плата перевищує середнє значення.

Тобто, підзапит повертає значення середньої заробітної плати компанії, яке дорівнює деякому значенню, наприклад 3750 грн. Результат виконання цього скалярного підзапиту використовується в зовнішньому операторі Select для обчислення відхилу заробітної плати від середнього рівня та для відбору відомостей про співробітників. Тому зовнішній оператор Select буде мати такий вигляд:

```
Select [Номер співробітника], ПІБ, Посада, ЗП – 3750 As ЗП1
From Співробітники
Where ЗП > 3750;
```

| Номер співробітника | ПІБ | Посада | ЗП1 |
|---|---|---|---|
| 14 | Федоров Микола Сергійович | Директор | 1750 |
| 9 | Краснова Олена Валеріївна | Бухгалтер | 450 |
| 41 | Васильєва Ганна Семенівна | Бухгалтер | 450 |

**Питання щодо самоперевірки:**

1. Що таке «Внутрішній запит» або «Підзапит»?
2. В яких операторах розташовують внутрішні запити?
3. Які типи внутрішніх запитів існують?
4. Що повертає в підсумку скалярний підзапит?
5. Який тип підзапиту повертає значення декількох стовпців таблиці, але у вигляді одного рядка?

## Тема «Ключові слова Any та All»

## (2 години)

**Мета:** познайомитися з ключовими словами Any та All та навчитися створювати запити з використанням цих слів.

Ключові слова **Any** та **All** можуть використовуватися з підзапитами, які повертають один стовпець.

Якщо підзапиту буде передувати Any, то умова порівняння буде вважатися виконаною лише в тому випадку, якщо воно виконується хоча б для одного зі значень в підсумковому стовпці підзапиту.

Якщо підзапиту буде передувати All, то умова порівняння буде вважатися виконаною лише в тому випадку, якщо воно виконується для кожного значення в підсумковому стовпці підзапиту.

Якщо в результаті виконання підзапиту буде отримане пусте значення, то для All умова порівняння буде вважатися виконаною, а для Any – не виконаною.

Додатково можна використовувати ключове слово **Some** замість Any.

**Приклад. Використання ключових слів Any або Some**

Знайти всіх співробітників компанії, заробітна плата яких більш ніж заробітна плата хоча б одного співробітника відділення компанії з номером 3.

Для виконання цього запиту скористуємося таблицею «Співробітники».

**Відношення «Співробітники»**

| Номер спів. | ПІБ | Адреса | Телефон | Посада | Стать | ДН | ЗП | Номер відділ. |
|---|---|---|---|---|---|---|---|---|
| 21 | Філатов Андрій Петрович | Одеса, Вільямса 7, 45 | 496433 | Менеджер | Ч | 01.05. 1970 | 2950 | 5 |
| 37 | Нікітіна Ганна Миколаївна | Черкаси, Корольова 67, 3 | | Секретар | Ж | 16.09. 1980 | 2700 | 3 |
| 14 | Федоров Микола Сергійович | Черкаси, б-р Шевченко 19, 5 | 480091 | Директор | Ч | 28.10. 1969 | 5500 | 3 |
| 9 | Краснова Олена Валеріївна | Київ, Пушкінська 21, 44 | 735565 | Бухгалтер | Ж | 31.12. 1964 | 4200 | 7 |
| 5 | Петренко Оксана Вікторівна | Одеса, Левітана 4, 90 | 489657 | Менеджер | Ж | 03.04. 1972 | 2950 | 3 |
| 41 | Васильєва Ганна Семенівна | Одеса, Грецька 47, 15 | 228900 | Бухгалтер | Ж | 18.02. 1968 | 4200 | 5 |

```
Select [Номер співробітника], ПІБ, Посада, ЗП
From Співробітники
Where ЗП > Any
        (Select  ЗП
         From Співробітники
         Where [Номер відділення]=3);
```

Хоча цей запит може бути записаний з використанням підзапиту, який визначає мінімальну заробітну плату співробітників компанії відділення з номером 3, після

чого зовнішній підзапит зможе обрати відомості про всіх співробітників компанії, заробітна плата яких перевищує це значення, можливий ще й інший підхід, який використовує ключові слова Any або Some. В цьому випадку внутрішній підзапит створює набір числових значень, а зовнішній запит обирає відомості про тих співробітників компанії, заробітна плата яких більш ніж заробітна плата будь-якого з значень в цьому наборі.

**Приклад. Використання ключового слова All**

Знайти всіх співробітників компанії, заробітна плата яких більш ніж заробітна плата всіх співробітників відділення компанії з номером 3.

Для виконання цього запиту також скористуємося таблицею «Співробітники».

```
Select [Номер співробітника], ПІБ, Посада, ЗП
From Співробітники
Where ЗП > All
        (Select  ЗП
         From Співробітники
         Where [Номер відділення]= 3);
```

Цей запит є схожим на попередній. В цьому випадку так саме можна було б скористатися підзапитом, який визначив би максимальну заробітну плату співробітників компанії відділення з номером 3, після чого за допомогою зовнішнього запиту обрати відомості про всіх співробітників компанії, заробітна плата яких перевищує це значення.

**Питання для самоперевірки:**

1. Для чого призначені ключові слова Any та All?
2. Якщо підзапиту буде передувати ключове слово Any, то умова порівняння буде вважатися виконаною або не виконаною? Обгрунтувати свою відповідь.
3. Якщо підзапиту буде передувати ключове слово All, то умова порівняння буде вважатися виконаною або не виконаною? Обгрунтувати свою відповідь.
4. Для чого використовується ключове слово Some?

## Тема «Створення форм в СУБД Access»

## (2 години)

**Мета:** навчитися будувати форми декількома способами.

Дані можна продивлятися безпосередньо в таблицях СУБД Access. Але це не завжди зручно, так як іноді неможливо вивести на екран всі поля одного запису одночасно.

Форми дозволяють змінити розташування даних на екрані з метою полегшення перегляду інформації на екрані. Можна вивести форми на друк – форми в першу чергу призначені для зображення даних на екрані.

Найбільш легким способом роботи є використання **автоформи**, яку можна створити натиском однієї кнопки. А **майстри форм** дозволяють створювати найбільш складні форми.

Для того, щоб створити автоформу, необхідно обрати таблицю або запит, на основі яких буде будуватися форма та скористатися відкриваючим переліком інструменту для створення нових об'єктів, обравши звідти інструмент «Автоформа». При першому зберіганні форми Access запитає її ім'я.

**Приклад створення простої автоформи:**

1. у випадку потреби необхідно відчинити вікно бази даних або натиснути на закладці «Таблиці». Обрати в переліку таблиць потрібну таблицю;
2. натиснути по кнопці «Автоформа» або скористатися переліком інструменту «Новий об'єкт» для обрання потрібної опції звідти, якщо це потрібно. Після невеликої паузи, необхідної Access для створення форми, на екрані з'явиться автоформа для цієї таблиці.
3. продивившись, треба закрити форму. Коли Access запитає підтвердження на запис форми, необхідно натиснути «Да» та зберегти її під іменем, яке встановлене за умовчуванням або надати нове ім'я.

Створення та робота з формами здійснюється за аналогом з іншими об'єктами Access. Першим треба в вікні бази даних натиснути на закладці «Формы» для того, щоб відобразити перелік форм, а потім використовувати кнопки вікна бази даних:

1. щоб створити нову форму, необхідно натиснути по кнопці «Создать» во вкладці «Формы» вікна бази даних;
2. для того, щоб скористатися вже існуючою формою, треба виділити її та натиснути по кнопці «Открыть»;
3. для зміни конструкції форми, необхідно виділити її та натиснути по кнопці «Конструктор».

При створенні нової форми Access відображає на екрані діалогове вікно «Новая форма», де спочатку необхідно за допомогою відкриваючого списку обрати таблицю або запит, які є основою для форми. Потім треба натиснути по закладці «Новая форма» («Blank Forms»), щоб створити пусту користувацьку форму або обрати опцію «Мастер форм» для того, щоб Access автоматично створив форму.

Як й при роботі з іншими об'єктами, створювати форму можна, обравши таблицю або запит в вікні бази даних та натиснувши по кнопці «Создать» (або обравши опцію «Форма» з меню «Вставка»). MS Access відображає на екрані діалогове вікно «Новая форма» з цією таблицею або запитом, які вже вміщені в відкриваючий список цього вікна.

Діалогове вікно «Новая форма» надає такі опції:

- **«Конструктор»** – створення форми з нуля;
- **«Мастер форм»** – створення форми за допомогою майстера;
- **«Автоформа в столбец»** – створення форми з полями, які розташовані один над одним;
- **«Автоформа ленточная»** – створення автоформи з полями записів, розташованими по вікну в вигляді таблиці. Ця форма дозволяє одночасно продивлятися декілька записів та має свою власну лінійку прокрутки, за допомогою якої можна прокручувати таблицю;
- **«Автоформа табличная»** – створення автоформи, яка схожа на таблицю;
- **«Диаграмма»** – створення графіку за допомогою «Майстру діаграм»;
- **«Сводная таблица»** – створення зведеної таблиці.

**Приклад створення форми за допомогою «Мастера форм»:**

1. Щоб створити нову форму, необхідно обрати спосіб створення форми «Створення форми за допомогою Мастера форм» во вкладці «Формы» вікна бази даних, натиснути подвійним натиском миші.
2. Відобразиться діалогове вікно «Создание форм», де спочатку необхідно за допомогою відкриваючого списку обрати таблицю або запит, які є основою для форми, а потім – обрати поля, які будуть присутні на формі. Натиснути «Далее».

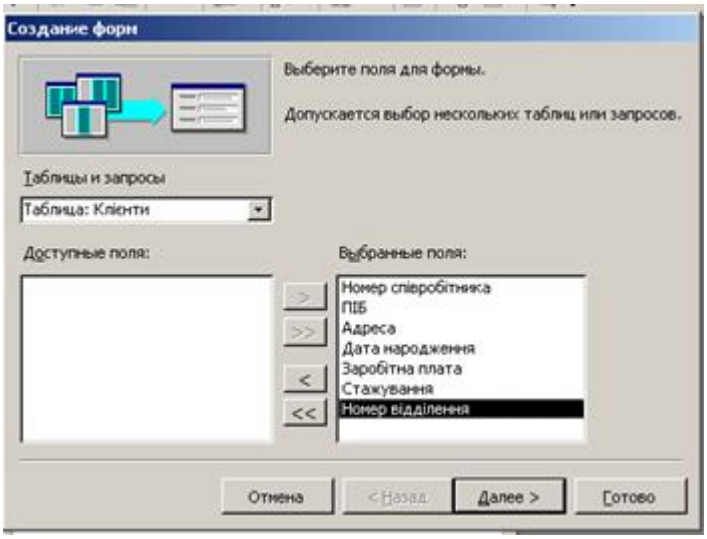



**Рисунок 1** – Вікно «Создание форм»

**3.** Обрати зовнішній вигляд форми:

- в один стовпець;
- стрічковий;
- табличний;
- вирівняний;
- зведена таблиця;
- зведена діаграма.

Натиснути «Далее».

**4.** Обрати потрібний стиль форми, тобто на якому фоні буде відображатися форма та натиснути кнопку «Далее».

**5.** Задати ім’я нової форми та натиснути кнопку «Готово», після чого на екрані з’явиться створена форма.

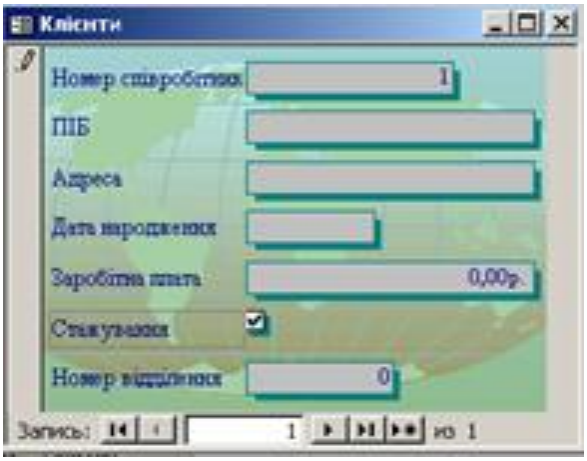



**Рисунок 2** – Форма «Клієнти»

## Створення форм за допомогою режиму конструктора

Створення форм за допомогою конструктора дає змогу користувачу керувати всіма елементами форми. Тому найбільш важливі форми, а також форми, які містять багато елементів повинні створюватися в режимі конструктора.

Для того щоб відчинити вже існуючу форму в режимі конструтору треба обрати її та натиснути на кнопку з написом «Конструктор».

Для створення нової форми в режимі конструктора треба натиснути кнопку «Создать», потім обрати спосіб створення форми «Конструктор», а в нижньому рядку обрати таблицю, на даних якої буде створена форма та натиснути кнопку «Ok».

Після цього на екрані з'явиться вікно, в якому знаходиться область сірого кольору. Якщо відчинити форму в режимі конструктору, то в структурі форми можна виділити три розділи:

- ***Роздел заголовка*** форми (містить назву форми, назву підзавдання або пункту меню, може містити логотип фірми, дату та ін.)
- ***Область данных***. Елементи керування, які містяться в області даних найчастіше є уявленням одного запису таблиці (або запиту). В цьому випадку кажуть про тип форми «в стовпець». Якщо треба відобразити одночасно декілька записів, то в такому випадку використовують стрічкову форму. Все, що міститься в області даних, є елементами керування. Фоновий малюнок, який знаходиться під елементами керування, показує розмір робочого поля форми.
- ***Роздел примечания*** форми (може містити підсумкові відомості, пояснення до області даних, спеціальні кнопки навігації, ссилки на інші форми та ін.)

Розмір форми можна змінювати. Для цього треба піднести курсор миші до правої або нижньої межі, або до правого нижнього кута (як зображене на рисунку) та після того, як курсор змінить свій вигляд натиснути ліву кнопку миші та, не відпускаючи її, розтягнути форму до бажаних розмірів. Змінити розмір форми можна в будь-який час.

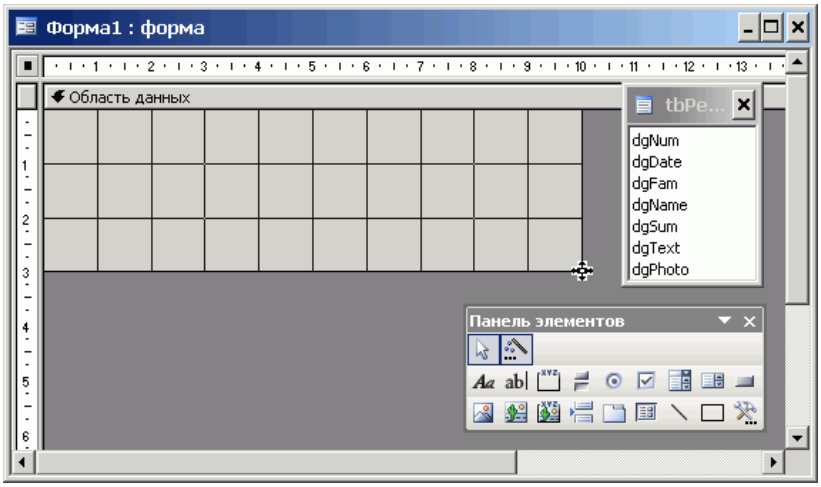



**Рисунок 3** – Форма в режимі конструктора

Після визначення розмірів форми, треба розмістити на ній візуальні елементи. Елементи форми можуть бути пов'язаними та непов'язаними. Пов'язані – прив'язані до поля початкової таблиці або запиту. Непов'язані (або вільні) – відображають результат обчислень або є даними (текстами, малюнками та ін.), який залишається незмінним незалежно від того, який запис в цей момент переглядають в формі.

Поруч з сірою областю або в області панелей інструментів можна побачити «Панель элементов» з кнопками, які дозволяють перенести на форму різні візуальні компоненти. Для того, щоб елемент став частиною форми його треба перенести на форму. Для цього необхідно натиснути один раз по кнопці «Панели элементов», яка відповідає необхідному елементу (для того, щоб обрати його), а потім натиснути один раз на формі. В цьому випадку створюється вказаний компонент стандартного розміру. Задати необхідні ширину та висоту можна за допомогою маленьких чорних квадратів, які з'являються навколо активного компонента. Можна одразу ж створити компонент з необхідними розмірами. Для цього після обрання компонента треба не натискати на формі, а розтягнути на ній прямокутник з бажаними розмірами компонента.

Панель елементів форми має такі елементи керування:

 - елемент **«Надпись»** служить для додавання написів, заголовків, інструкцій.

 - елемент **«Поле»** використовується для створення полів введення та розрахункових полів.

 - кнопка вимкнення програм-майстерів.

 - елемент **«Группа»**.

 - **выключатели** особливо зручні при використанні в групах. В такій групі легко побачити, який з вимикачів натиснутий. Замість напису на вимикачі можна розташувати малюнок.

 - **переключатели** звичайно використовуються в групі для відображення набору параметрів, з яких треба обрати один. З цими елементами можна зв'язати команди, наприклад, які виконують фільтрацію.

 - елемент **«Флажок»**. Використовується для введення значень в логічні поля. Якщо прапорець відмічений, то значення логічного поля = Істина (1, Так). В іншому випадку – Неправда (0, Ні). Прапорці також можна розташовувати в групі. Вони діють аналогічно перемикачам, але в відмінність від них, припускають множинне обрання.

 **- «Поле со списком»**

 **- «Список»**

Використовуються для організації введення даних, обираючи значення зі списку значень або таблиці значень. Ці елементи дуже корисні при введенні даних з пов'язаних таблиць. Поля зі списком займають менше місця на формі, а список їх значень виводиться на екран лише за вказівкою користувача. Списки займають більше місця, але в них завжди відображається декілька можливих для введення значень.

 - елемент **«Прямоугольник»** служить для виділення підпису поля разом з полем або для виділення декількох елементів форми можна скористатися таким засобом оформлення, як прямокутник. Після обрання піктограми навколо обраних об'єктів малюється рамка. Для того, щоб прямокутник не перетинав елементи, які розташовані під ним, з меню треба обрати команду «Формат» - «На задній план». Після цього можна змінити розташування, колір, розмір прямокутника.

 - **«Командна кнопка»** використовується для виконання операцій навігації, по керуванню даними та ін. (наприклад, перейти на новий запис, відчинити або зачинити форму, знищити запис).

 - елемент **«Свободная рамка объекта»** не пов'язаний з жодним полем таблиць БД. Об'єкт, який знаходиться в ній, виконує роль ілюстрації та призначений для оформлення форми.

- елемент «Присоединённая рамка объекта». З такою рамкою пов’язане одне з полів таблиці. В ній відображається зміст цього поля. Цей зміст може змінюватися при переході від одного запису до іншого.

Наприклад, для того щоб малюнок помістився до рамки поля, треба виділити поле OLE, натиснути праву кнопку миші та в контекстному меню обрати пункт «Свойства». В вікні властивостей обрати пункт «Установка размеров» та зі списку обрати значення «По размеру рамки». Зачинити форму та підтвердити зберігання змін.

- елемент «Вкладка» дозволяє розмістити багато інформації на обмеженій площині. На вкладках розташовують інші елементи керування.

Розпочинаємо створювати форму завжди з завдання заголовка або назви форми. Для цього натискаємо лівою кнопкою миші по кнопці, а потім на формі розтягуємо великий прямокутник у верхньому краю. Одразу ж після створення, елемент «Надпись» знаходиться в режимі редагування тексту, тобто треба тут же ввести текст. Завершується введення тексту натисканням клавіши Enter. Одразу ж після створення елемента його властивості мають значення, які задаються за умовчуванням та текст не схожий на заголовок форми, так як розмір шрифту, який використовується, дуже малий. Для того, щоб змінити розмір шрифту треба відредагувати властивості елемента «Надпись». В контекстному меню присутні засоби форматування, але лише їх обмежений набір.

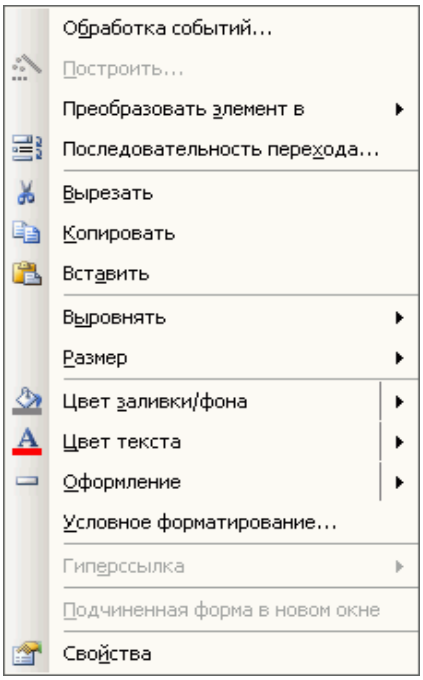



**Рисунок 4** – Властивості форми

При виділенні будь-якого візуального елемента в Access в правому верхньому куті з’являється великий чорний квадрат. Використовуючи його можна перерозташовувати елемент по формі, а також за допомогою натискання правою кнопкою миші вивести контекстне меню для роботи з елементом. Для того щоб вивести вікно властивостей обраного в дану мить візуального елемента треба або обрати пункт «Свойства» в контекстному меню об’єкта, або натиснути кнопку  .

Вікно властивостей має декілька сторінок:

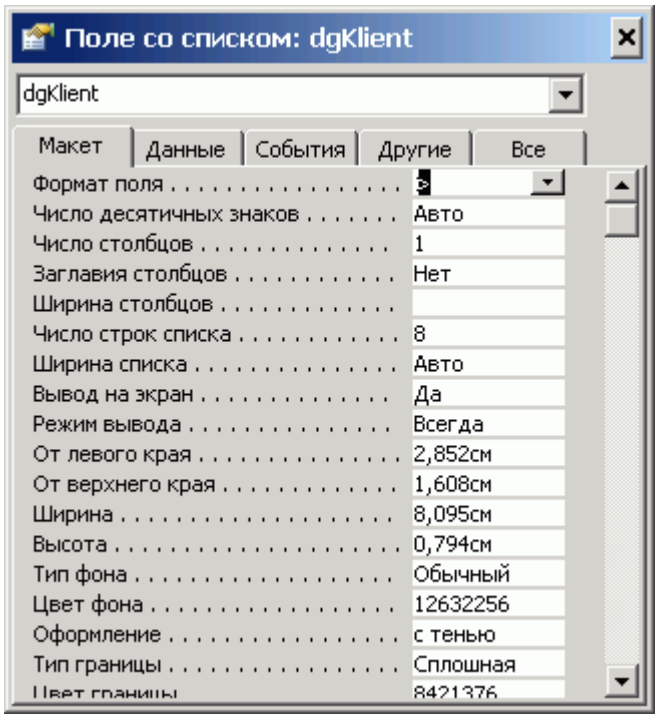



**Рисунок 5** – «Поле со списком» форми

1. **Макет.** На даній сторінці зібрані властивості, які відповідають за зовнішній вигляд елемента, такі як: відстань від лівого та верхнього краю контейнера, найменування, розмір, колір та дані шрифту, колір межі, колір фону та ін.
2. **Данные.** Якщо елемент може бути пов’язаний з полем бази даних, то на цій сторінці можна заповнити властивості, які відповідають за зв’язок з даними, а також умови введення.
3. **События.** На цій сторінці зображені різні події, на які може реагувати елемент. Деякі події обробляються самим візуальним елементом, але, як правило, розробник сам пише код, який визначає реакцію на ту або іншу подію. Код, який визначає реакцію на подію, називається обробником події. Найпоширеніші події, які виникають в Access: Вхід, Вихід, Внесені зміни,

Отримання фокуса, Втрата фокуса, Натискання кнопки, Подвійне натискання кнопки та т. д.

4. **Другие.** На цій сторінці розташовані властивості, які не можна віднести до трьох попередніх категорій. Вони звичайно використовуються в програмуванні.
5. **Все.** На цій сторінці відображаються всі властивості, які є лише в елемента.

**Питання щодо самоперевірки:**

1. Що таке «Форма»?
2. Чому краще переглядати дані в режимі форми?
3. Для чого призначені форми?
4. Що таке «Автоформа» та як її можна створити?
5. Які опції надає діалогове вікно «Нова форма»?
6. Як створити форму за допомогою «Майстру форм»?
7. Для чого призначений спосіб створення форми «Конструктор»?

## Тема «Створення фільтрів в СУБД MS Access»

## (2 години)

**Мета:** навчитися створювати фільтри.

Як відомо, запити є незалежними об’єктами, які розташовуються в вікні бази даних разом з таблицями та іншими об’єктами.

Іноді краще вбудовувати властивості запиту в існуючий об’єкт замість того, щоб створювати запит як новий об’єкт, окремий від інших за допомогою фільтру можна вбудовувати властивості запиту в таблицю або форму без створення окремого об’єкту-запиту.

**Фільтр** – засіб, який призначений для зміни конструкції таблиці або форми, частиною якої він є.

Так як у фільтрів немає окремих імен, таблиця або форма можуть мати тільки один приєднаний до себе фільтр. Якщо створити новий фільтр, він замістить вже існуючий. Але можна зберегти фільтр для постійного використання, записавши його як запит. Також можна створити фільтр на основі існуючого запиту.

Після створення фільтру для таблиці або форми з’явиться кнопка «Применение фільтра». Натискаючи послідовно по цій кнопці, можна або застосувати фільтр, щоб відобразити на екрані тільки деякі рядки, або відмінити фільтрацію для відображення всіх записів.

**Зауваження**: якщо до здійснення пошуку даних був використаний фільтр, наступний пошук інформації Access буде здійснювати тільки серед відфільтрованих записів. якщо Access не може знайти запис, який введений, то причина міститься в попередній фільтрації даних.

Існує чотири типи фільтрів:

1. **«Фильтр по выделенному фрагменту»** - визначає, які записи будуть відображені на екрані шляхом виділення даних в таблиці в «Режиме таблицы»;
2. поле **«Фільтр для»** - дозволяє вносити умови відбору безпосередньо в контексному меню;
3. **«Обычный фильтр»** - визначає, які записи будуть відображені на екрані в режимі форми, яка є аналогом таблиці в «Режиме таблицы»;
4. **«Розширенный фильтр»** - вказує, які записи будуть відображені на екрані та визначає порядок, в якому вони з’являються на екрані за допомогою вікна «Розширенный фильтр»/ «Сортировка», яке є таким самим як й вікно запиту.

**Зауваження**: незалежно від типу створеного фільтру, його можна відобразити на екран двома способами: обрати з меню «Фильтр» опції «Фильтр для формы» або «Розширенный фильтр»/ «Сортировка». Умови відбору фільтру відобразяться в вікні «Фильтр для формы» або в вікні «Розширенный фильтр».

1. **<u>Фільтр по выделенному фрагменту</u>**

Для створення цього фільтру треба вивести на екран вікно таблиці або форми, на основі яких необхідно створити фільтр, та виділити значення, яке треба знайти в будь-якому з записів, де б не знаходилось це значення.

Потім необхідно обрати з меню «Записи» опцію «Фильтр» та з подменю «Фильтр» обрати «Фильтр по выделенному фрагменту» або натиснути по кнопці «Фильтр по выделенному» на стандартній панелі інструментів.

Короткий шлях: можна просто натиснути правою кнопкою миші по полю, щоб виділити весь його зміст та відобразити на екрані контекстне меню для цього поля, потім обрати в цьому контекстному меню опцію «Фильтр по выделенному».

**2. Використання поля «Фильтр для»**

Для того, щоб застосувати поле «Фильтр для», треба натиснути по стовпцю поля, для якого необхідно ввести умову відбору, та ввести умову відбору в його контекстне меню.

**3. «Звичайний фільтр»**

Для того, щоб створити звичайний фільтр, треба відобразити на екрані вікно таблиці або вікно форми, для яких необхідно створити фільтр. Потім обрати опцію «Фильтр» з меню «Записи», після чого з подменю – опцію «Изменить фильтр» або натиснути по кнопці «Изменить фильтр» на панелі інструментів.

MS Access відобразить вікно створення фільтру, яке містить один запис з таблиці, розташований також, як й в режимі таблиці.

Потім треба ввести значення, які необхідно знайти, в відповідні поля даного запису або для кожного поля треба скористатися відкриваючим списком для відбору значення зі списку всіх унікальних значень, які були введені в поле. MS Access автоматично додасть обмеження до введеного значення, так само як це відбувається при створенні запитів.

Якщо значення заносяться в декілька полів, то між ними буде встановлене логічне відношення типу «И». Для того, щоб встановити для умов відбору відношення «Или», необхідно натиснути по закладці «Или» в нижній частині цього вікна для того, щоб відобразити на екрані новий зразок запису, та заповнити для нього умови відбору. Після заповнення форми для повернення до таблиці треба з меню «Записи» обрати опцію «Применить фильтр», або натиснути по кнопці «Применение фильтра». При цьому на екрані з’являться лише записи, які відповідають встановленим умовам. Можна натиснути в вікні «Фильтр» правою кнопкою миші для відображення контекстного меню, з якого необхідно обрати опцію «Применить фильтр».

**4. «Розширенный фільтр»**

Фільтр може бути створений способами, якими користувалися для створення запитів. Для створення поширеного фільтру треба відобразити на екрані таблицю

(або форму), до якої необхідно застосувати фільтр, обрати з меню «Записи» та з подменю «Фильтр» – опцію «Розширенный фильтр» для відображення на екрані вікна «Фиільтр».

Вікно «Фильтр» практично таке ж саме як й вікно запиту. Воно автоматично містить список полів таблиці або запиту.

На відміну від запиту, який фільтрує записи та поля, фільтр відображає на екрані всі поля таблиці та фільтрує лише записи. Він немає в бланку побудови рядка «Вывод на экран», так як виводить на екран всі поля. Користувач переміщує поля з таблиці в бланк побудови, для того щоб внести для них умови відбору під ними, але це жодним чином не вплине на кількість полів, які виводяться на екран.

Після створення фільтру, його треба застосувати, обравши з меню «Фильтр» команду «Применить фильтр» або натиснувши по кнопці «Применение фильтра». Можна також натиснути по області вікна «Фильтр» правої кнопкою миші для виведення на екран контекстного меню та з цього меню обрати опцію «Применить фильтр». Якщо закрити діалогове вікно «Фильтр», фільтр автоматично не буде застосований.

**Збереження фільтру як запиту**

Так як таблиця може мати тільки один фільтр, який втрачається при створенні нового фільтру, в MS Access є можливість зберегти фільтр для постійного використання у вигляді запиту. Для того щоб його зберегти, треба натиснути по області вікна «Фильтр» правою кнопкою миші для відображення контекстного меню. Обрати опцію «Сохранение в виде запроса» (або просто натиснути по кнопці «Сохранение в виде запроса»), яка при роботі з вікном «Фильтр» з’являється на панелі інструментів замість кнопки «Сохранение».

MS Access відображає діалогову панель «Сохранение в виде запроса», де треба внести ім’я для запиту. Новий запит з’явиться на екрані в вікні бази даних та може бути використаний так саме, як будь-який інший запит.

**Використання запиту в якості фільтру**

Можна скористатися в якості фільтру вже існуючим запитом. Запит повинен бути на відбір, який базується виключно на таблиці або запиті, до яких необхідно застосовувати запит в якості фільтру.

Натиснувши по області вікна «Фильтр» правою кнопкою миші, треба обрати «Загрузить с запроса» (або натиснути по кнопці «Загрузить с запроса», яка з'являється при роботі в вікні «Фильтр», на панелі інструментів замість кнопки «Открыть»). MS Access виводить на екран діалогове вікно «Фильтр, который применяется» («Applicable Filter»), де зображується список запитів, які можна використати до цього об'єкту в якості фільтрів. Після цього треба обрати запит з цього списку та натиснути «Ок». Вікно «Фильтр» автоматично заповнюється такими ж визначеннями, що й вікно запиту.

**Питання для самоперевірки:** вивчити конспект, знати відповіді на такі питання:

1. Що таке «Фільтр»?
2. Для чого призначені фільтри?
3. Чим відрізняється фільтр від запиту?
4. Для яких об'єктів СУБД MS Access створюються фільтри?
5. Що з'явиться на панелі інструментів після створення фільтру?
6. Які типи фільтрів Ви знаєте?
7. Для чого фільтр зберігають у вигляді запиту та як це зробити?
8. Як створити фільтр?

## Тема «Створення звітів в СУБД Access»

## (2 години)

**Мета:** навчитися створювати звіти в СУБД Access

**Звітом** називають будь-який набір даних, призначений для друку.

Так як звіт призначений спеціально для виведення на друк, MS Access у випадку необхідності виведе звіт на екран в вікні попереднього перегляду. Користувач немає можливості редагувати дані, які відображаються в звіті.

Після натискання по кнопці «Создать» MS Access відображає на екрані діалогове вікно «Новый отчёт». Як й при роботі з формами, треба використовувати відкриваючий список цього вікна для обрання таблиці або запиту, для яких створюється звіт.

Потім необхідно обрати одну з опцій:

- **«Конструктор»** – створення користувацького звіту з нуля;
- **«Мастер отчётов»** – використання «Майстру звітів» для створення звіту;
- **«Автоотчёт в столбец»** – створення автозвіту (обраний за умовчуванням автозвіт містить списки імен полів та змісту цих полів, які розташовані один під одним. Таке розташування не зовсім є зручним для звітів), в якому поля розташовані один під одним;
- **«Автоотчёт ленточный»** – створення автозвіту, в якому поля розташовані один за одним. Такий звіт корисний в тому випадку, якщо б він базувався на запиті, який містить лише декілька полів, та вони вміщуються на екрані та на сторінці при друку;
- **«Мастер диаграмм»** – відображає на екрані «Майстер діаграм», який використовується для створення графіків;
- **«Почтовые наклейки»** – виводить на екран вікно «Создание почтовых наклеек», за допомогою якого можна створювати наклейки або інші етикетки.

**Питання для самоконтролю:**

1. Що таке «Звіт»?
2. Для чого призначені звіти?
3. Яким способом можна створити новий звіт?
4. Як називається звіт, в якому поля розташовані одне під одним?
5. Для чого призначений спосіб створення звітів «Поштові наклейки»?

## Тема «Створення макросів в СУБД Access»

## (2 години)

**Мета:** ознайомитися з поняттям та призначенням макросів. Навчитися створювати макроси.

В Access макроси – це визначенна послідовність операцій. Вони зберігаються в вікні, яке нагадує таблицю, в такому порядку, в якому їх треба виконувати. Коли запускається макрос Access виконує ці дії. Макрос може істотньо полегшити використання Access. Однією з корисних особливостей макросів є можливість їх з’єднання з кнопками, які розміщуються на формі. Ці кнопки можуть виконувати операції, які часто повторюються (наприклад, відчиняти діалогові вікна для пошуку,

змінювати порядок сортування даних або друкувати звіт). Використовуючи макроси, можна створювати додатки з користувацьким меню та діалоговими вікнами.

Для створення макроса необхідно відчинити вкладку «Макросы» в вікні БД та виконати натискання по кнопці «Создать». Цю ж дію можна виконати з використанням пунктів меню «Вставка» -> «Макрос». В діалоговому віні, яке відкриється, відображена інформація такого виду: вікно макросів складається з двох частин: верхньої та нижньої. В верхній частині знаходиться перелік макрокоманд, які треба виконати та необов'язкові пояснення до цих команд. В нижній частині вікна знаходяться аргументи макрокоманди.

При розробці макросу необхідно задати дії (ввести макрокоманди), які він повинен буде виконати (наприклад, відчинити форму, надрукувати звіт, виконати запит або експортувати зміст таблиці в файл електронної таблиці). В Access такі дії можна визначити двома способами:

1. обрати з переліку в стовпці «Макрокоманда» (або ввести їх власноруч),
2. перемістити об'єкти з вікна БД в стовпець «Макрокоманда» вікна макросів.

Розглянемо більш детально кожен з цих способів.

**1-й спосіб:**

В вікні макросів виконати натискання мишкою на першій порожній комірці в стовпці «Макрокоманда». Потім виконати натискання по кнопці відчинення переліку. При цьому відкриється перелік припустимих макрокоманд. Обрати з переліку команду, яку повинен виконати макрос або набрати цю команду власноруч.

Виконати натискання в нижній частині вікна або натиснути клавішу [F6] та вказати аргументи дії.

**2-й спосіб:**

Для створення макроса необхідно виконати такі дії:

Перемістити вікно макросів та змінити його розміри таким чином, щоб одночасно можна було б бачити вікно макросів та бази даних. Обрати вкладку об'єкту, який буде відчинятися макросом. Виконати натискання на потрібному об'єкті та перемістити його в порожній рядок стовпця «Макрокоманда» вікна макросів. Після цього в стовпці «Макрокоманда» з'явиться відповідна команда.

В розділі «Аргументы» макрокоманди з'являться аргументи. При необхідності їх можна змінювати.

**Макрокоманди, їх призначення та аргументи:**

1. возобновить - відновлює розмір максимізованого або мінімізованого вікна. Аргументів немає.
2. Вивести в формате - виводить дані в файл іншого формату. Аргументи - Тип об'єкта; Ім'я об'єкта; Формат виведення; Ім'я файлу; Автозапуск; Файл шаблона.
3. Вивод на екран - визначає, повинна або ні СКБД Access оновлювати екран під час виконання макроса. Аргументи - Включити виведення; Текст рядка стану.
4. Обрати об'єкт - обрати вказаний об'єкт. Аргументи - Тип об'єкта; Ім'я об'єкта; В вікні бази даних.
5. Виконати команду - виконує команду меню. Аргументи – Команда.
6. Вихід - Ініціює вихід з Access. Аргументи – Параметри.
7. Додати меню - додає меню в рядок користувацького меню. Аргументи - Назва меню; Ім'я макроса; Текст рядка стану.
8. Задати значення - встановлює значення поля, елементу керування або властивості. Аргументи – Елемент; Вираз.
9. Зачинити - зачиняє вказаний об'єкт або активне вікно (при відсутності аргумента). Аргументи - Тип об'єкта; Ім'я об'єкта; Зберігання.
10. Запуск запиту SQL - виконує запит, використовуючи оператор SQL. Аргументи - Інструкція SQL; Використовувати транзакцію.
11. Запуск макроса - виконує інший макрос. Аргументи - Ім'я макроса; Кількість повторів; Умова повтору.
12. Копіювати об'єкт - копіює об'єкт бази даних в іншу базу даних або в ту ж базу, але під іншим ім'ям. Аргументи - База даних; Нове ім'я; Тип об'єкта; Ім'я об'єкта.
13. До елемента керування - переміщує курсор в поле або в елемент керування активної форми, таблиці даних або динамічного набору запиту. Аргумент - Елемент керування.
14. На запис - переміщує курсор на вказаний запис. Аргументи - Тип об'єкта; Ім'я об'єкта; Запис; Зміщення.
15. Знайти запис - знаходить перший запис, який задовільняє умовам, визначеним дією «Знайти запис» або значеннями в діалоговому вікні «Знайти».

**16.** Зупинити макрос - зупиняє виконання поточного макроса. Аргументів немає.

**17.** Відчинити запит - відчиняє обраний запит у вказанному режимі. Аргументи - Ім’я запита; Режим; Режим даних.

**18.** Відчинити форму - відчиняє форму у вказанному режимі (форми, конструктора, перегляду або таблиці). Аргументи - Ім’я форми; Режим; Ім’я фільтра; Умова відбору; Режим даних; Режим вікна.

**19.** Пісочні годинники - під час виконання макроса змінює форму вказателя миші на форму пісочних годинників. Аргументи – Ввімкнути.

**20.** Розвернути - максимізує активне вікно. Аргументів немає.

**21.** Звернути - мінімізує активне вікно. Аргументів немає.

**22.** Сигнал - видає звуковий сигнал. Аргументів немає.

**23.** Наступний запис - знаходить наступний запис, який задовільняє умовам, визначенним подією «Знайти запис» або значеннями в діалоговому вікні «Знайти». Аргументів немає.

**24.** Повідомлення - відображає вікно з повідомленням або попередженням. Аргументи – Повідомлення; Сигнал; Тип; Заголовок.

В стовпці «Примечание» можна вести будь-які коментарі, які допоможуть простежити яку дію виконує макрос.

**Збереження макросів**

Для збереження макроса треба виконати наступні дії: обрати команду «Файл» -> «Сохранить». Або виконати натискання по піктограмі «Сохранить» на панелі інструментів. Якщо макрос зберігається вперше, Access запитає для нього ім’я. Ввести ім’я створенного макроса, виконати натискання по кнопці «ОК» та закрити вікно макросів, натиснувши комбінацію клавиш [**Ctrl**]+[**F4**].

**Виконання макроса**

Після завершення розробки макроса можна перевірити його роботу, запустивши макрос на виконання. Самим простим засобом є наступний: відкрити вкладку «Макросы» в вікні БД, виділити макрос та натиснути по кнопці «Запуск» або виконати подвійне натискання на макросі в вікні БД.

Інший спосіб запуску наступний: Обрати команду «Сервис»-> «Макрос»-> «Запуск макроса». В діалоговому вікні «Запуск макроса», яке з’явиться, обрати або

ввести ім’я макроса. Якщо макрос відчинений в режимі конструктора, його можна виконати, натиснувши по кнопці «Запуск» на панелі інструментів.

Крім того, макроси можна виконати за допомогою кнопок, які додаються до форми. Кнопки макросів можна створювати методом перетискання.

1. Відчинити потрібну форму в режимі конструктора, перемістити форму та встановити її розміри таким чином, щоб можна було б бачити й вікно БД.
2. Відчинити вкладку «Макросы» в вікні БД для відображення макросів.
3. Перетащити потрібний макрос в те місце форми, де необхідно розташувати кнопку. Створенна кнопка з’явиться в вікні форми.

Макрос можна настроїти таким чином, щоб він виконувався автоматично при запуску БД. Для цього достатньо при зберіганні створенного макроса присвоїти йому ім’я Autoexec. Макроси Autoexec звичайно використовуються для відчинення форм, з якими частіш усього працюють користувачі, або для розміщення на екрані декількох найвикористовуємих форм та/або звітів. Утримуючи клавішу [Shift] під час відчинення БД, можна відмінити запуск макроса Autoexec.

**Редагування макросів**

Структура таблиці в вікні макросів нагадує структуру звичайної таблиці БД. Команди редагування текста, які використовуються для знищення, переносу та копіювання змісту комірок, можуть застосовуватися в рамках таблиці макроса.

Редагування макроса здійснюється в режиме конструктора. Аргументи та краткий опис макрокоманди відображаються в вікні макросів лише при її маркіровці.

Наприклад, для вставки додаткової макрокоманди в існуючий макрос треба виконати наступні дії:

1. обрати вкладку «Макросы» в вікні БД;
2. обрати макрос для редагування, виконавши на ньому натискання мишкою;
3. відчинити макрос в режимі конструктора, обравши кнопку «Конструктор»;
4. обрати макрокоманду, перед якою треба виконати вставку нової, виконати натискання мишкою в одному з полів цієї макрокоманди або маркіровав весь рядок цілком;
5. обрати пункт меню «Вставка», підпункт «Строки». Перед маркірованим рядком буде доданий порожній рядок;
6. поместити до цього рядка нову макрокоманду;

7. зберігти макрос («Файл» -> «Сохранить»).

**Копіювання макросів**

Макроси можна копіювати з однієї БД в іншу або в одну й ту ж саме БД під різними іменами. Це економить час при створенні макросів, які виконують схожі задачі.

Існуючий макрос можна скопіювати, виконавши наступні дії:

1. в вікні БД обрати потрібний макрос;
2. обрати команду «Правка» -> «Копировать»;
3. для копіювання макроса в іншу БД зачинити поточну та відчинити ту, до якої буде копіюватися макрос. Обрати в вікні БД вкладку «Макросы».
4. обрати команду «Правка» -> «Вставить».

В діалоговому вікні, яке з'явиться, ввести ім'я макроса. Якщо макрос копіюється в ту ж БД, в якій знаходиться існуючий макрос, то макросу, якій копіюється, треба дати інше ім'я.

**Питання щодо самоперевірки:**

1. Що таке «макрос»?
2. Якими способами можна створить макрос?
3. Які існують макрокоманди та яке в них призначення?
4. Як зберегти макрос?
5. Як можна запустити макрос на виконання?
6. Як можна відредагувати макроси?

**Тема «Системний каталог»**

**(2 години)**

**Мета:** дізнатися призначення системного каталогу.

Системним каталогом називається сукупність спеціальних таблиць БД. Їх створює, супроводжує та володіє ними сама СУБД. Ці системні таблиці містять інформацію, яка описує структуру БД. Таблиці системного каталогу створюються автоматично при створенні бази даних. Звичайно вони об'єднуються під спеціальним системним ідентифікатором користувача з таким іменем, як SYSTEM, MASTER або DBA.

При обробці операторів SQL СУБД постійно звертається до даних системного каталогу. Наприклад, для того щоб опрацювати двотабличний оператор SELECT, СУБД повинна:

1. перевірити, існують або ні дві вказані таблиці;
2. впевнитися, що користувач має дозвіл на доступ до них;
3. перевірити, існують або ні стовпці, на які є посилання в даному запиті;
4. встановити, до яких таблиць відносяться неповні імена стовпців;
5. визначити тип даних кожного стовпця.

Якщо б системні таблиці служили лише для задовільнення внутрішніх потреб СУБД, то для користувачів бази даних вони не представляли б практично жодного зацікавлення. Однак системні таблиці, як правило, доступні також й для користувачів – запити до системних каталогів дозволені майже у всіх СУБД. За допомогою запитів до системних каталогів можна отримати інформацію про структуру БД, навіть якщо ніколи раніше з нею не працювали.

Користувачі можуть лише зчитувати інформацію з системного каталогу. СУБД забороняє користувачам модифікувати системні таблиці безпосередньо, так як це може порушувати цілісність бази даних. СУБД сама додає, знищує та оновлює рядки системних таблиць під час модифікації структури бази даних. Зміни в системних таблицях відбуваються також в якості побічного результату виконання таких операторів DDL, як CREATE, ALTER, DROP, GRANT та REVOKE.

**Зміст системного каталогу**

Кожна таблиця системного каталогу містить інформацію про окремий структурний елемент БД. В склад майже всіх комерційних реляційних СУБД входять з невеликою різницею, системні таблиці, кожна з яких описує один з наступних п’яти елементів:

1. Таблиці. В каталозі описується кожна таблиця бази даних: вказується її ім’я, власник, кількість стовпців, які в ній містяться, їх розмір та т. д.
2. Стовпці. В каталозі описується кожен стовпець бази даних: приводиться ім’я стовпця, ім’я таблиці, до якої він належить, тип даних стовпця, його розмір, дозволені або ні значення null та т. д.
3. Користувачі. В каталозі описується кожен зареєстрований користувач бази даних: вказується ім’я користувача, його пароль в зашифрованому вигляді та інші дані.

4. Уявлення. В каталозі описується кожне уявлення, яке є в базі даних: вказуються його ім’я, ім’я власника, запит, який є визначенням уявлення, та т. д.
5. Привілеї. В каталозі описується набір привілеїв, наданих в базі даних: приводяться імена тих, хто надав привілеї, та тих, кому вони надані, вказуються самі привілеї, об’єкт, на які вони розповсюджуються, та т. д.

Ось приклади декількох запитів, які використовуються для вилучення інформації про структуру бази даних з уявлень системного каталогу, визначених в стандарті SQL2:

1. Вивести імена всіх таблиць та уявлень користувача, який працює в теперішній момент з базою даних.

```
SELECT TABLE_NAME
FROM TABLES
```

2. Вивести ім’я, позицію та тип даних для кожного стовпця у всіх уявленнях.

```
SELECT TABLE_NAME, C.COLUMN_NAME, ORDINAL_POSITION,
DATAJTYPE
FROM COLUMNS
WHERE (COLUMNS.TABLE_NAME IN (SELECT TABLE_NAME FROM
VIEWS))
```

3. Визначити, скільки стовпців є в таблиці STUDENTS.

```
SELECT COUNT(*)
FROM COLUMNS
WHERE (TABLE_NAME = «STUDENTS»)
```

## Тема «Відкриті системи»

## (2 години)

**Мета:** дізнатися про концепцію відкритих систем.

Реальне поширення архітектури "клієнт-сервер" стало можливим завдяки розвитку та широкому впровадженню в практику концепції відкритих систем.

Основним змістом підходу відкритих систем є спрощення комплексування обчислювальних систем за рахунок міжнародної та національної стандартизації апаратних і програмних інтерфейсів. Головною спонукальною причиною розвитку

концепції відкритих систем стали повсюдний перехід до використання локальних комп'ютерних мереж і ті проблеми комплексування апаратно-програмних засобів, що їх спричинив цей перехід. У зв'язку з бурхливим розвитком технологій глобальних комунікацій відкриті системи набувають ще більшого значення і масштабності.

Ключовою фразою відкритих систем, спрямованою в бік користувачів, є незалежність від конкретного постачальника. Орієнтуючись на продукцію компаній, що дотримуються стандартів відкритих систем, споживач, який купує будь-який продукт такої компанії, не потрапляє до неї в рабство. Він може продовжити нарощування потужності своєї системи шляхом придбання продуктів будь-якої іншої компанії, що дотримується стандартів. Причому це стосується як апаратних, так і програмних засобів і не є необґрунтованою декларацією. Реальну можливість незалежності від постачальника перевірено у вітчизняних умовах.

Практичною опорою системних і прикладних програмних засобів відкритих систем є стандартизована операційна система. Нині такою системою є UNIX. Фірмам-постачальникам різних варіантів ОС UNIX у результаті тривалої роботи вдалося дійти згоди про основні стандарти цієї операційної системи. Зараз усі поширені версії UNIX в основному сумісні в частині інтерфейсів, що надаються прикладним (а в більшості випадків і системним) програмістам. Як здається, незважаючи на появу системи Windows NT, що претендує на стандарт, саме UNIX залишиться основою відкритих систем у найближчі роки.

Технології та стандарти відкритих систем забезпечують реальну і перевірену практикою можливість виробництва системних і прикладних програмних засобів із властивостями мобільності (portability) та інтероперабельності (interoperability). Властивість мобільності означає порівняльну простоту перенесення програмної системи в широкому спектрі апаратно-програмних засобів, що відповідають стандартам. Інтероперабельність означає спрощення комплексування нових програмних систем на основі використання готових компонентів зі стандартними інтерфейсами.

Використання підходу відкритих систем вигідне і виробникам, і користувачам. Насамперед відкриті системи забезпечують природне розв'язання проблеми поколінь апаратних і програмних засобів.

Виробники таких засобів не змушені розв'язувати всі проблеми заново; вони можуть принаймні тимчасово продовжувати комплексувати системи, використовуючи наявні компоненти.

Зауважимо, що при цьому виникає новий рівень конкуренції. Усі виробники зобов'язані забезпечити деяке стандартне середовище, але змушені домагатися його якомога кращої реалізації. Звісно, через якийсь час наявні стандарти почнуть відігравати роль стримування прогресу, і тоді їх доведеться переглядати.

Перевагою для користувачів є те, що вони можуть поступово замінювати компоненти системи на більш досконалі, не втрачаючи працездатності системи. Зокрема, у цьому криється розв'язання проблеми поступового нарощування обчислювальних, інформаційних та інших потужностей комп'ютерної системи.

## Тема «Сервери баз даних»

## (2 години)

**Мета:** дізнатися про призначення серверів баз даних.

Термін "сервер баз даних" зазвичай використовують для позначення всієї СУБД, що ґрунтується на архітектурі "клієнт-сервер", включно із серверною та клієнтською частинами. Такі системи призначені для зберігання і забезпечення доступу до баз даних. Хоча зазвичай одна база даних цілком зберігається в одному вузлі мережі і підтримується одним сервером, сервери баз даних являють собою просте і дешеве наближення до розподілених баз даних, оскільки загальна база даних доступна для всіх користувачів локальної мережі.

### 1. Принципи взаємодії між клієнтськими та серверними частинами

Доступ до бази даних від прикладної програми або користувача здійснюється шляхом звернення до клієнтської частини системи. Як основний інтерфейс між клієнтською і серверною частинами виступає мова баз даних SQL.

Ця мова по суті справи являє собою поточний стандарт інтерфейсу СУБД у відкритих системах. Збірна назва SQL-сервер відноситься до всіх серверів баз даних, заснованих на SQL. Дотримуючись застережень під час програмування, деякі з яких були розглянуті на попередніх лекціях, можна створювати прикладні інформаційні системи, мобільні в класі SQL-серверів.

Сервери баз даних, інтерфейс яких ґрунтується виключно на мові SQL, мають свої переваги та свої недоліки. Очевидна перевага - стандартність інтерфейсу. У межі, хоча поки що це не зовсім так, клієнтські частини будь-якої SQL-орієнтованої СУБД могли б працювати з будь-яким SQL-сервером незалежно від того, хто його зробив.

Недолік теж досить очевидний. За такого високого рівня інтерфейсу між клієнтською і серверною частинами системи на стороні клієнта працює занадто мало програм СУБД. Це нормально, якщо на стороні клієнта використовується малопотужна робоча станція. Але якщо клієнтський комп'ютер має достатню потужність, то часто виникає бажання покласти на нього більше функцій керування базами даних, розвантаживши сервер, який є вузьким місцем усієї системи.

Одним із перспективних напрямків СУБД є гнучке конфігурування системи, за якого розподіл функцій між клієнтською та користувацькою частинами СУБД визначається під час встановлення системи.

**2. Переваги протоколів віддаленого виклику процедур**

Згадувані вище протоколи віддаленого виклику процедур є особливо важливими в системах управління базами даних, заснованих на архітектурі "клієнт-сервер".

По-перше, використання механізму віддалених процедур дає змогу справді перерозподіляти функції між клієнтською та серверною частинами системи, оскільки в тексті програми віддалений виклик процедури нічим не відрізняється від віддаленого виклику, а отже, теоретично будь-який компонент системи може бути розміщений і на стороні сервера, і на стороні клієнта.

По-друге, механізм віддаленого виклику приховує відмінності між взаємодіючими комп'ютерами. Фізично неоднорідна локальна мережа комп'ютерів приводиться до логічно однорідної мережі взаємодіючих програмних компонентів. У результаті користувачі не зобов'язані серйозно піклуватися про разову закупівлю сумісних серверів і робочих станцій.

**3. Типовий розподіл функцій між клієнтами і серверами**

У типовому на сьогоднішній день випадку на стороні клієнта СУБД працює тільки таке програмне забезпечення, яке не має безпосереднього доступу до баз даних, а звертається для цього до сервера з використанням мови SQL.

У деяких випадках хотілося б включити до складу клієнтської частини системи деякі функції для роботи з "локальним кешем" бази даних, тобто з тією її частиною, яка інтенсивно використовується клієнтською прикладною програмою. У сучасній технології це можна зробити тільки шляхом формального створення на стороні клієнта локальної копії сервера бази даних і розгляду всієї системи як набору взаємодіючих серверів.

З іншого боку, іноді хотілося б перенести більшу частину прикладної системи на бік сервера, якщо різниця в потужності клієнтських робочих станцій і сервера надто велика. Загалом при використанні RPC це зробити неважко. Але потрібно, щоб базове програмне забезпечення сервера дійсно дозволяло це. Зокрема, при використанні ОС UNIX проблеми практично не виникають.

**4. Вимоги до апаратних можливостей і базового програмного забезпечення клієнтів і серверів**

Із попередніх міркувань видно, що вимоги до апаратури та програмного забезпечення клієнтських і серверних комп'ютерів різняться залежно від виду використання системи.

Якщо поділ між клієнтом і сервером досить жорсткий (як у більшості сучасних СУБД), то користувачам, які працюють на робочих станціях або персональних комп'ютерах, абсолютно байдуже, яка апаратура та операційна система працюють на сервері, аби тільки він справлявся з потоком запитів, що виникають.

Але якщо можуть виникнути потреби перерозподілу функцій між клієнтом і сервером, то вже зовсім не все одно, які операційні системи