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

Конспект лекцій

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

Конспект лекцій з дисципліни «Організація баз даних та знань» для підготовки фахового молодшого бакалавра за спеціальністю 122 «Комп'ютерні науки».

Укладач: викладач вищої категорії ФКПАІТ ОНТУ Юлія БУРМАКІНА

Конспект лекцій затверджено 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

Зміст **Стор.**

Основи теорії баз даних та систем управління базами даних	4
Вступ. Основні поняття БД та СУБД. Функції СУБД	4
Переваги та недоліки СУБД	10
Ієрархічна, мережева та реляційна модель даних	15
Реляційна цілісність БД. З'язки в реляційній моделі даних	27
Теорія проектування баз даних	31
Етапи проектування бази даних. Побудова концептуальної моделі предметної області	31
Логічне проектування бази даних	38
Нормалізація відношень	42
Структурована мова запитів SQL	51
Мова SQL. Оператор Select. Запити на читання даних. Використання агрегатних функцій.	51
Використання операторів Group by та Having. Робота з багатотабличними запитами	62
Зміна змісту бази даних. Оператори Insert, Update та Delete	68
Визначення структури даних в SQL	72
Представлення та привілеї в SQL	79
СУБД в архітектурі «Клієнт - сервер»	86
Архітектура «Клієнт - сервер». Моделі архітектури «Клієнт - сервер»	86

Блок змістових модулів №1

Основи теорії баз даних та систем управління базами даних

Змістовий модуль 1.1 - Основи проектування баз даних

Тема 1.1.1 Вступ. Основні поняття БД та СУБД. Функції СУБД.

(2 години)

Мета: засвоєння головних понять, пов'язаних з базами даних та системами управління базами даних. Ознайомитися з функціями, якими повинна володіти кожна система управління базами даних, та знати їх зміст.

.**Структура заняття:**

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Поняття «бази даних» та «системи управління базами даних».
2. Призначення бази даних та системи управління базами даних.
3. Попередники баз даних.
4. Приклади баз даних.
5. Перелік функцій, які повинні бути реалізовані в кожній СУБД.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 1.1.1.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке «база даних»?
2. Яку роль виконують бази даних в теперішньому сучасному житті?
3. Навести приклади баз даних.
4. Хто або що було попередником баз даних?

5. Які переваги використання підходу з використанням баз даних?
6. Що таке «Система управління базами даних»?
7. Навести приклади систем управління базами даних.
8. Хто запропонував перелік сервісів, які повинні бути реалізовані в будь-якій повномасштабній СУБД?
9. Якими функціями володіє кожна система управління базами даних?
10. Яка функція є самою фундаментальною функцією СУБД?
11. Що означає поняття «транзакції» при роботі з базою даних? Детально розписати все про транзакції.
12. До якої функції належить використання принципу «Все або нічого»?
13. Яка функція СУБД гарантує коректне відновлення бази даних при паралельному виконанні операцій відновлення багатьма користувачами?
14. В якій функції СУБД забезпечується механізм відновлення бази даних та повернення її в несуперечливий стан?
15. З якою метою треба приховувати інформацію, розміщену в базі даних, від інших користувачів?
16. Завдяки чому здійснюється контроль доступу до даних бази даних?
17. Що таке «цілісність бази даних» та в чому вона зображується?

Поява баз даних стала самим важливим досягненням в галузі програмного забезпечення. Бази даних лежать в основі інформаційних систем та це головним чином змінило характер роботи багатьох організацій. З моменту своєї появи технологія баз даних стала захоплюючою областю діяльності, а також каталізатором багатьох значних досягнень в області програмного забезпечення.

Бази даних стали невід'ємною частиною нашого повсякденного життя.

Наприклад, доступ до бази даних потрібен при покупці продуктів в супермаркеті, коли касир з покупок зчитує штрих-код. При цьому ручний сканер, пов'язаний з додатком бази даних, використовує штрих-код для пошуку ціни обраного предмету в базі даних всіх товарів. Потім програма віднімає кількість всіх тільки що проданих товарів з товарних запасів та відобразить на касовому апараті їх коштовність. Якщо запаси опустяться нижче визначеного деякого рівня, то в такому

випадку система зможе автоматично спрямувати замовлення на поставку додаткової кількості цього товару.

Якщо при покупці використовується кредитна картка, касир повинен перевірити присутність кредитних коштів. Це можливо зробити за телефоном або автоматично за допомогою спеціального зчитуючого пристрою, який пов'язаний з комп'ютером. В будь-якому випадку при цьому використовується база даних, яка містить відомості про покупки, здійснювані за допомогою цієї кредитної картки. За номером кредитної картки спеціальний додаток звіряє ціну покупаємих в дану мить та куплених на протязі цього місяця товарів з кредитним лімітом. Після підтвердження припустимості такої покупки всі відомості про отримані товари заносяться в базу даних. Але ще до отримання підтвердження припустимості покупки додаток бази даних повинен переконатися в тому, що дана кредитна картка не знаходиться в переліку вкрадених або утрачених.

Коли при плануванні відпустки Ви звертаєтесь в туристичне агентство, співробітник цього агентства за Вашим запитом продивляється базу даних з відомостями про всі наявні путівки та розклад польотів. При бронюванні якої-небудь путівки система бази даних повинна виконати всі потрібні для цього дії. В даному випадку потрібно бути впевненим, що два різних співробітника агентства не бронюють одну й ту ж саме путівку або для даного рейсу не забронювані міста за межами припустимої кількості. Наприклад, якщо в літаку деякого рейсу залишилось тільки одне вільне місце та два співробітники туристичного агентства спробують його забронювати, то система повинна коректно опрацювати цю ситуацію та дозволити забронювати останнє місце тільки одному співробітнику, відправивши іншому співробітнику повідомлення про відсутність вільних місць.

В будь-якому навчальному закладі може існувати база даних з інформацією про студентів, відвідуємих ними курсах, отримуємих стипендіях, вже пройдених та вивчаємих в теперішній час дисциплінах, а також про підсумки здачі різних екзаменів. Крім цього, може також підтримуватися база даних з інформацією про прийом студентів в наступному році, а також база даних з особистими даними про співробітників та відомості про їх заробітну плату для бухгалтерії.

Попередником бази даних була файлова система, тобто набір додатків, які виконували окремі необхідні користувачу операції, наприклад, такі як створення звітів. Кожна програма визначала та керувала своїми власними даними.

Хоча файлова система була значним досягненням у порівнянні з ручною картотекою, її використання все ж таки було пов'язано з великими проблемами, які взагалі були пов'язані з надмірністю даних та залежністю програм від даних.

Для підвищення ефективності роботи необхідно було використовувати новий підхід, а саме базу даних (**D**ata**b**ase) та систему управління базою даних (**D**ata**b**ase **M**anagement **S**ystem - **DBMS**).

База даних – це використовуємий набір логічно пов'язаних даних (та опис цих даних), призначенних для задовільнення інформаційних потреб організації.

База даних – це єдине, велике сховище даних, яке одноразово визначається, а потім використовується одночасно багатьма користувачами з різних підрозділів.

Замість розрізнених файлів з надмірними даними, тут всі дані зібрані разом з мінімальною часткою надмірністю. База даних вже не належить деякому одному відділу, а є загальним корпоративним ресурсом. Вона зберігає не тільки робочі дані цієї організації, але й їх опис. За цією причиною бази даних ще називають **набіром інтегрованих записів з самоописом.**

У сукупності опис даних називають **системним каталогом** або **словником даних**, а самі елементи опису називають **метаданими**, тобто „даними про дані". Саме наявність самоопису даних в базі даних забезпечує в ній незалежність між програмами та даними.

В підході з використанням бази даних структура даних відокремлена від додатків та зберігається в базі даних. Додавання нових структур даних або зміна існуючих ніяким чином не впливає на додатки, за умови що вони не залежать безпосередньо від змінних компонентів. Наприклад, додавання нового поля в запис або створення нового файлу ніяким чином не вплине на роботу існуючих додатків. Але знищення поля з використовуємого додатком файлу вплине на цей додаток, а тому його також необхідно буде відповідним чином модифікувати.

Система управління базою даних (СУБД) – це програмне забезпечення, за допомогою якого користувачі мають змогу визначати, створювати та підтримувати базу даних, а також здійснювати до неї контролюємий доступ.

Е. Ф. Кодд (французький математик) запропанував перелік сервісів, які повинні бути реалізовані в будь-якій повномасштабній СУБД:

1. **зберігання, вилучення та відновлення даних**

СУБД повинна надавати користувачам можливість зберігати, вилучати та відновлювати дані в базі даних. Це є найбільш фундаментальною функцією СУБД. Засіб реалізації цієї функції повинен приховувати від кінцевого користувача внутрішні деталі фізичної реалізації системи (наприклад, файлову організацію чи структури зберігання, які використовуються).

2. **підтримка транзакцій**

СУБД повинна мати механізм, який гарантує виконання або всіх операцій відновлення даної транзакції, або жодної з них.

Транзакція уявляє собою набір дій, які виконуються окремими користувачами або прикладною програмою з метою доступу чи зміни змісту бази даних.

Прикладами простих транзакцій може бути додавання в базу даних відомостей про співробітника. Відновлення відомостей про заробітну плату деякого співробітника чи знищення відомостей про деякий об'єкт нерухомості.

Прикладом найбільш важкої транзакції може бути знищення відомостей про співробітника з бази даних та передача відповідальності за всі керовані ним об'єкти нерухомості іншому співробітнику. У цьому випадку в базу даних необхідно буде ввести одночасно декілька змін.

Якщо під час виконання транзакції відбудеться збій, наприклад, із-за виходу зі строю комп'ютеру, база даних потрапить в суперечливий стан, так як деякі зміни вже будуть здійснені, а решта – ще ні. Тому всі часткові зміни не будуть збережені для повернення бази даних в колишній несуперечливий стан.

ACID – ***вимоги*** гарантують правильність та надійність роботи системи.

Atomic (атомарність) - транзакція не може бути виконана частково, або все, або нічого.

Consistency (узгоджуванність) - після виконання транзакції всі дані повинні знаходитися в узгоджуваному стані. Тобто транзакція не руйнує взаємної узгоджуваності даних.

Isolation (ізольованність) - означає, що транзакції, які конкурують за доступ до бази даних, фізично обробляються послідовно, ізольовано одна від одної.

Durability (стійкість) - після завершення транзакції зміни, які були внесені, залишуться незмінними.

ACID - фундаментальні властивості систем обробки транзакцій

3. <u>сервіси управління паралельністю</u>

СУБД повинна мати механізм, який гарантує коректне відновлення бази даних при паралельному виконанні операцій відновлення багатьма користувачами.

Головною метою створення та використання СУБД є те, що багато користувачів мають змогу здійснювати паралельний доступ до даних, які спільно опрацьовуються. Паралельний доступ порівняно легко організувати, якщо всі користувачі виконують лише читання даних, так як в цьому випадку вони не зможуть зашкодити одне одному. Але, коли два або більш користувачів одночасно отримують доступ до бази даних, конфлікт з небажаними наслідками легко може виникнути, наприклад, якщо хоча б один з них спробує відновити дані. СУБД повинна гарантувати те, що при одночасному доступі до бази даних багатьох користувачів схожих конфліктів не відбудеться. Для цього треба використовувати термін «блокування».

4. <u>сервиси відновлення</u>

СУБД повинна надати засоби відновлення бази даних на випадок будь-яких ушкоджень або знищень.

При збої транзакція повинна бути повернена в несуперечливий стан. Такий збій може трапитися у випадку виходу зі строю системи чи запам’ятовуючого пристрою, помилки апаратного чи програмного забезпечення, які можуть привести до зупинки СУБД. Крім цього, користувач має змогу знайти помилку під час виконання транзакції та вимагати її відміни.

У всіх цих випадках СУБД повинна забезпечити механізм відновлення бази даних та повернення її в несуперечливий стан.

Кожна база даних має журнал транзакцій, у якому фіксуються всі зміни даних, зроблені в кожній із транзакцій.

Журнал транзакцій - це важлива складова бази даних. Якщо система дасть збій, цей журнал допоможе повернути базу даних в узгоджений стан.

5. **сервіси контролю доступу до даних**

СУБД повинна мати механізм, який гарантує можливість доступу до бази даних лише санкціонованих користувачів.

Іноді необхідно приховувати деякі відомості, які зберігаються в базі даних від інших користувачів. Наприклад, менеджерам відділень компаній можна було б надавати інформацію, яка пов'язана з заробітною платою співробітників, але бажано приховати її від інших користувачів. Крім того, базу даних необхідно було б захистити від будь-якого несанкціонованого доступу.

Термін *«безпека»* відноситься до захисту бази даних від навмисного чи випадкового несанкціонованого доступу. Припускається, що СУБД забезпечує механізми подібного захисту даних.

6. **служби підтримки цілісності даних**

СУБД повинна володіти інструментами контролю за тим, щоб дані та їх зміни відповідали вказаним правилам.

Цілісність бази даних визначає коректність та несуперечливість зберігаємих даних. Вона може розглядатися як ще один тип захисту бази даних.

Крім того, що дане питання пов'язане з забезпеченням захисту, воно має більш широкий зміст, оскільки цілісність пов'язана з якістю самих даних. Цілісність звичайно зображується у вигляді обмежень або правил зберігання несуперечливості даних, які не повинні порушуватися в базі даних.

Наприклад, можна вказати, що співробітник немає змогу відповідати одночасно більш ніж за 10 об'єктів нерухомості. В даному випадку при спробі закріпити черговий об'єкт нерухомості за деяким співробітником, СУБД повинна перевірити, чи не перевищений встановлений ліміт, та у випадку виявлення подібного перевищення закріпити новий об'єкт за іншим співробітником.

Тема 1.1.2 Переваги та недоліки СУБД

(2 години)

Мета: визначення переваг та недоліків кожної СУБД та ознайомлення з їх змістом.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;

b. Психоемоційний настрій;

c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План:

1. Переваги СУБД.
2. Недоліки СУБД.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 1.1.2.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Якими перевагами володіє СУБД?
2. Що потрібно зробити для того, щоб вилучити надмірність даних?
3. В яких випадках виникають несуперечливі дані?
4. Завдяки чому описується цілісність бази даних?
5. У вигляді чого може бути представлена система забезпечення безпеки баз даних?
6. Якими недоліками володіє СУБД?
7. Чому СУБД вважається складним програмним продуктом?
8. Продуктивність відноситься до переваг чи недоліків СУБД? Пояснити свою відповідь.

СУБД володіють як потенціальними перевагами, так й недоліками.

Переваги СУБД:

1. контроль за надмірністю даних;
2. несуперечливість даних;
3. спільне використання даних;
4. підтримка цілісності даних;
5. підвищена безпека.

Контроль за надмірністю даних

При використані бази даних передбачається спроба вилучення надмірності даних за рахунок інтегрування файлів для того, щоб уникнути зберігання декількох копій того ж самого елементу інформації. Але повністю надмірність даних не виключається, а лише контролюється її ступінь. В одних випадках ключові елементи даних треба дублювати для моделювання зв'язків, а в інших випадках деякі дані треба буде продублювати з міркувань підвищення продуктивності системи.

Несуперечливість даних

Вилучення надмірності даних або контроль за нею дозволяють скоротити ризик виникнення суперечливих станів.

Якщо елемент даних зберігається в базі даних в одному примірнику, то для зміни його значення треба буде виконати лише одну операцію оновлення, до того ж нове значення стане припустимим одразу всім користувачам бази даних. А якщо цей елемент даних зберігається в базі даних в декількох примірниках, то така система має можливість стежити за тим, щоб копії не суперечили одна одній.

На жаль, в багатьох сучасних СУБД такий тип несуперечності даних автоматично не підтримується.

Спільне використання даних

Файли звичайно належать окремим особам або цілим відділам, які використовують їх в своїй роботі. В той же самий час, база даних належить всій організації в цілому та має можливість спільно використовуватися всіма зареєстрованими користувачами. При такій організації роботи більша кількість користувачів може працювати з великим об'ємом даних. До того ж, при цьому можна створювати нові додатки на підставі вже існуючої в базі даних інформації та додавати до неї тільки ті дані, які в теперішню мить ще не зберігаються в ній, а не визначати знов вимоги до всіх даних, які потрібні новому додатку.

Нові додатки можуть також використовувати такі функціональні можливості, як визначення структур даних та керування доступом до даних, організація паралельної обробки та забезпечення засобів копіювання / відновлення, виключаючи необхідність реалізації цих функцій зі свого боку.

Підтримка цілісності даних

Цілісність бази даних визначає коректність та несуперечливість зберігаємих в ній даних. Цілісність звичайно описується за допомогою обмежень, тобто правил підтримки несуперечливості, які не повинні порушуватися в базі даних. Обмеження можна пристосовувати до елементів даних усередині одного запису або до зв'язків поміж записами.

Наприклад, обмеження цілісності може казати, що заробітна плата співробітника на місяць не повинна перевищувати 50000 грн. або те, що в запису з даними про співробітника номер відділення, в якому він працює, повинен відповідати існуючому відділенню компанії.

Таким чином, інтеграція даних дозволяє адміністратору баз даних завдавати вимоги по підтримці цілісності даних, а СУБД використовувати їх.

Підвищена безпека

Безпека бази даних міститься в захисті бази даних від несанкціонованого доступу з боку користувачів. Без залучення відповідних заходів безпеки інтегровані дані стають найбільш уразливими, ніж дані в файловій системі.

Система забезпечення безпеки може бути висловлювана в формі обліку імен та паролів для ідентифікації користувачів, які зареєстровані в цій базі даних. Доступ до даних з боку зареєстрованого користувача може бути обмежений деякими операціями (видобуванням, додаванням, оновленням та знищенням). Наприклад, адміністратору баз даних може бути надано право доступу до всіх даних бази даних, менеджеру відділення компанії – до всіх даних, які належать до його відділення, а інспектору відділу реалізації – тільки до даних про нерухомість, в результаті цього він немає доступу до конфіденційних даних, наприклад, заробітної плати співробітників компанії.

Недоліки СУБД:

1. важкість;
2. розмір;
3. вартість СУБД;
4. витрати на перетворення;
5. продуктивність.

Важкість

Забезпечення функціональності, якою повинна володіти кожна добра СУБД, супроводжується значним ускладнюванням програмного забезпечення СУБД.

Для того, щоб скористатися всіма перевагами СУБД, проектувальники та розробники баз даних, адміністратори даних та адміністратори баз даних, а також кінцеві користувачі повинні добре розуміти функціональні можливості СУБД. Нерозуміння принципів роботи системи може призвести до невдалих результатів проектування, що буде мати дуже серьйозні наслідки для всієї організації.

Розмір

Важкість та широчінь функціональних можливостей призводить до того, що СУБД стає надзвичайно важким програмним продуктом, який може займати багато місця на диску та вимагати великого об'єму оперативної пам'яті для ефективної роботи.

Вартість СУБД

В залежності від існуючого обчислюваного середовища та потребуємих функціональних можливостей, вартість СУБД може знаходитися в дуже широких межах.

Витрати на перетворення

В деяких ситуаціях вартість СУБД та додаткового апаратного забезпечення може виявитися несуттєвою у порівнянні з вартістю перетворення існуючих додатків для роботи з новою СУБД та новим апаратним забезпеченням. Ці витрати також містять вартість підготовки персоналу для роботи з новою системою, а також сплачування послуг фахівців, які будуть здійснювати допомогу в перетворенні та запуску нової системи.

Все це є однією з головних причин, за якою деякі організації залишаються прибічниками колишніх систем та не бажають здійснювати перехід до найбільш сучасних технологій управління бази даних.

Продуктивність

Звичайно файлова система створюється для деяких спеціалізованих додатків, наприклад, для оформлення рахунків, а тому саме її продуктивність може бути дуже високою.

Але СУБД призначені для вирішення найбільш загальних задач та обслуговування одразу декількох додатків, а не якогось одного з них. В підсумку багато які додатки в новому середовищі будуть працювати не так швидко, як раніше.

Змістовий модуль 1.2 - Моделі даних

Тема 1.2.1 Ієрархічна, мережева та реляційна модель даних

(2 години)

Мета: засвоєння поняття «модель даних», її призначення, види моделей даних, властивості ієрархічної, мережевої та реляційної моделей даних.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Поняття «Модель даних» та її призначення. Види моделей даних.
2. Призначення ієрархічної моделі даних. Головні інформаційні одиниці та властивості ієрархічної моделі даних.
3. Недоліки ієрархічної моделі даних.
4. Призначення мережевої моделі даних. Головні інформаційні одиниці мережевої моделі даних.
5. Переваги мережевої моделі даних.
6. Структура реляційних даних.
7. Властивості реляційної таблиці.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

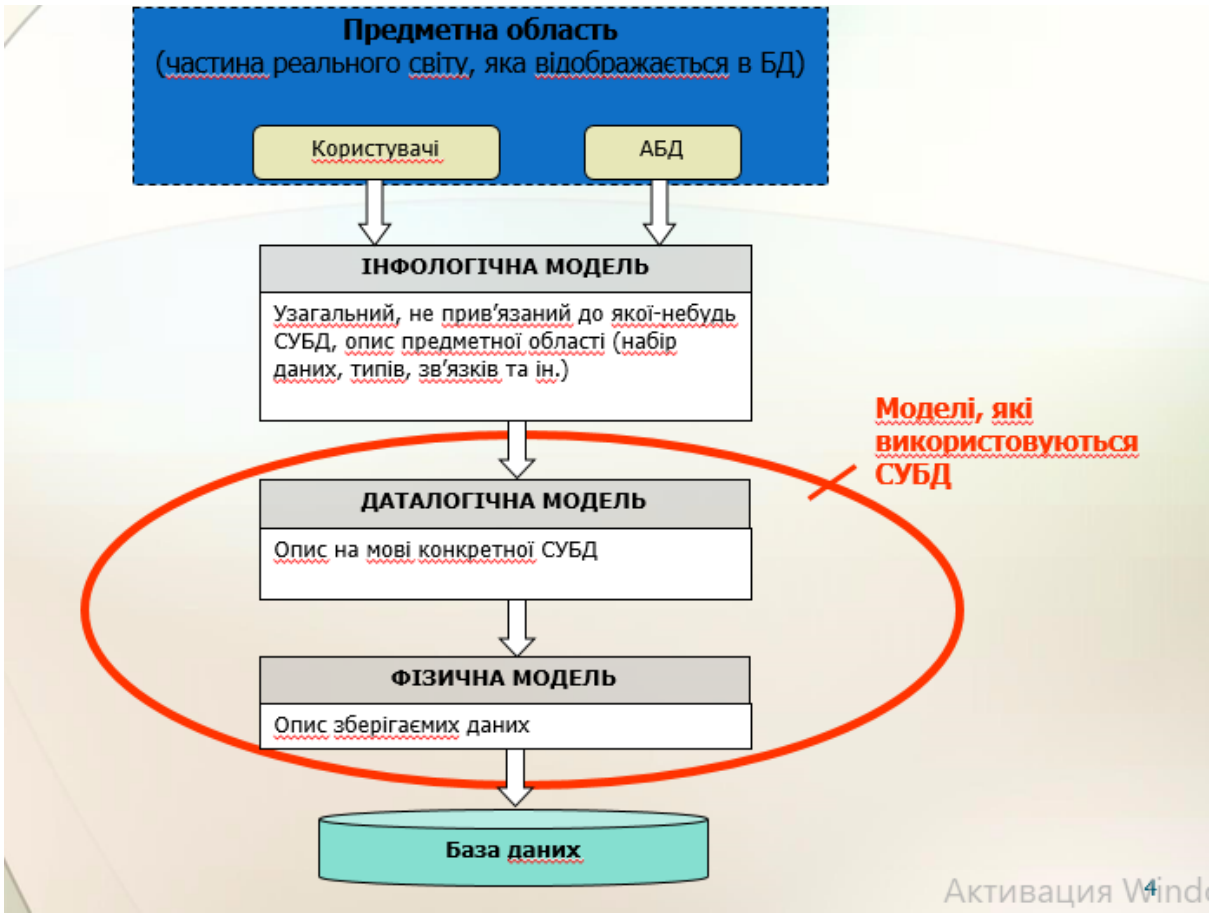
VI. Домашнє завдання:
   1. Ознайомитись з теоретичними відомостями теми 1.2.1.
   2. Вивчити основні поняття лекції.
   3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що розуміється під поняттям «модель даних»?
2. Які існують моделі даних?
3. Що таке «ієрархія»?
4. Що є головними інформаційними одиницями ієрархічної моделі даних?
5. Якими властивостями володіє ієрархічна модель даних?
6. Що розуміється під поняттям «елемент даних», «запис» та «набір даних»?
7. Чим відрізняється мережева модель даних від ієрархічної моделі даних?
8. Якими перевагами володіє мережева модель даних?
9. Ким була запропонована реляційна модель даних?
10. Чим реляційна модель даних відрізняється від інших моделей даних?
11. Чому реляційна модель даних стала найпоширенішою?
12. Які головні поняття використовуються при роботі з реляційною моделю даних? Дати їм визначення.
13. Яка характеристика реляційної моделі даних визначається кількістю атрибутів, які містяться в базі даних?
14. Для чого в базі даних треба створювати первинний ключ?
15. Навіщо при роботі з пов'язаними таблицями використовується зовнішній ключ?
16. Які вимоги надаються до таблиць реляційної моделі даних?

Дані, які зберігаються в базі даних, повинні бути організовані за визначеними правилами. Організація даних та способи доступу до них, які забезпечуються конкретною СУБД, називається **моделю даних**.

Рівні моделей даних (послідовність розробки БД)



Існуючі бази даних побудовані на основі ієрархічної, мережевої, реляційної та об'єктно-орієнтованої моделей даних або їх підмножин.

Ієрархічна модель даних

Ієрархія - розташування елементів від найвищого елементу до найнижчого.

Ієрархічна модель даних (ІМД) представляє собою древовидну (ієрархічну) структуру. В вершинах дерева знаходяться сукупності властивостей даних, які описують деякий об’єкт. В термінології ІМД ці сукупності називаються **сегментами**. Сегмент, у якого немає вище за нього рівня ієрархії, називається **кореневим (головним)**, а інші знаходяться на найнижчих рівнях - **підпорядковані**. Сегменти на найвищих рівнях мають назву - **логічно початкові**, **батьківські** або **предки**. Сегменти на найнижчих рівнях мають назву – **логічно підпорядковані**, **дочірні** або **потомки**.

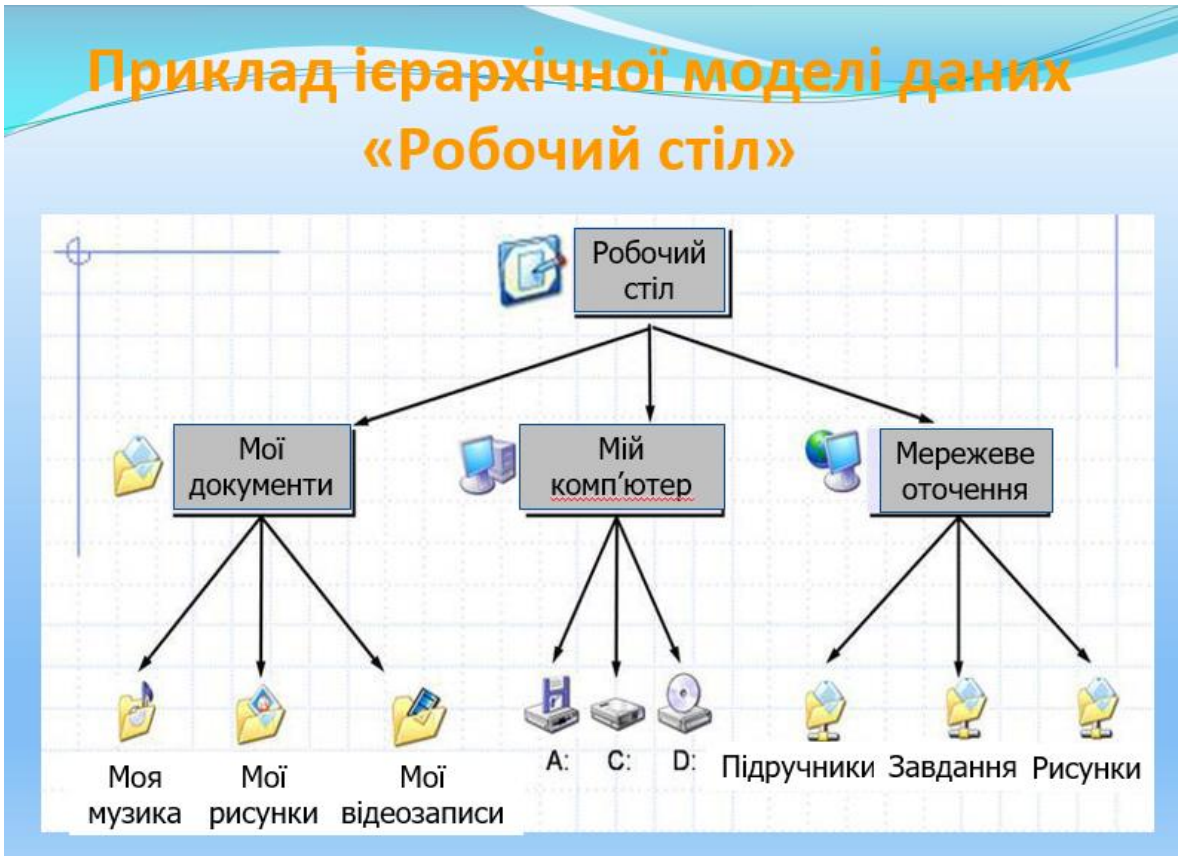
Головними інформаційними одиницями моделі є:

1) **поле** - мінімальна нерозподілена одиниця даних (прізвище, посада);
2) **сегмент (запис)** - сукупність полів, яка характеризує об’єкт.

Для відмінності окремих сегментів кожен тип сегменту має ключ.

Ключ - сукупність елементів, які значним чином визначають сегмент (номер співробітника, номер студентського квитка, номер залікової книжки, ідентифікаційний код, номер паспорту).

Прикладом ієрархічної БД є робочий стіл.



Ієрархічною БД є файлова система, яка складається з кореневої директорії, в якій є ієрархія піддиректорій та файлів.

Сукупність конкретних значень полів сегмента називається **екземпляром сегмента** або **записом**: м. Київ, вул. Перемоги, 3; 18-67-53. З цього можна сказати, що один блок на схемі відображає множину фактичних записів (екземплярів сегмента).

Пошук необхідного запису в ієрархічній БД завжди починається від кореневого запису. Наприклад, необхідно обрати запис з характеристиками конкретної партії товарів (кількість, колір, розмір, вартість та т. д.). Треба послідовно вказати філіал, магазин, склад, де ці товари знаходяться. Для спрощення пошуку записи кожного сегмента підпорядковуються за даними деякого поля.

Головні властивості ієрархічної моделі даних

1. Кожний з підлеглих сегментів пов'язаний лише з одним сегментом рівня ієрархії, який знаходиться вище нього. Зв'язки між сегментами одного уровня не допускаються.

2. Між сегментами двох рівнів можуть підтримуватися лише зв’язки «один до багатьох» (один філіал - багато магазинів, один склад - багато товарів) або «один до одного» (один філіал - один директор).

Недоліки ієрархічної моделі даних:

1. ***Асиметричність запитів***. В цьому можна впевнитися, порівнюючи два запити. За допомогою першого з них обирається запис з інформацією про партії товарів, для якої відомі назви філіала, магазина та склада. В рамках другого запиту за відомостями про партії товарів треба визначити назву філіала, в якому вона знаходиться.

2. ***Складність змін***. Наприклад, при закритті деякого складу товари перерозподіляються поміж іншими складами. Ця операція потребує кардинальної зміни структури БД.

3. ***Жосткість структури***, яка не дозволяє врахувати в БД окремі реальні ситуації. Наприклад, товари, які зберігаються на одному складі, призначені декільком магазинам.

4. ***Складність експлуатації***. Робота з ієрархічними БД потребує значної кваліфікації користувачів в області програмування. В частності, в СКБД IMS фірми IBM для опису загальної схеми БД та блоку зв’язка кожного користувача з БД використовувалась мова програмування Assembler. Для відбору даних з БД - спеціалізована мова DL/1, для обробки отриманої інформації – мови PL/1 або Кобол.

Перелічені причини призвели до того, що ієрархічна модель даних в теперішній час практично не використовується.

Мережева модель даних

Одним із основоположників мережевої моделі даних є американський учений Чарльз Бахман. В 1973 р. за керування роботою Data Base Task Group (робоча група по базам даних, США) він був нагороджений премією Тьюрінга – найпрестижнішою премією в галузі гнформатики.

Приклад: СУБД Integrated Database Management System (IDMS) компанії Cullinet Software, Inc. (1972 р.)

Мережева модель бази даних являє собою сукупність об'єктів різного рівня, проте схема зв'язків між об'єктами може бути будь-якою.

Мережеві БД більш гнучкі: немає явно вираженого головного елемента й існує можливість встановлення горизонтальних зв'язків. Наприклад, організація інформації в Інтернеті (WWW).

Елементами цієї моделі є: **рівень, вузол та зв'язок.**

В мережевій моделі даних не існує підпорядкованості рівнів «один до одного».

В мережевій моделі даних не накладаються жодні обмеження на кількість зв'язків, які входять до однієї вершини. Тобто, зв'язки можна встановлювати не лише між вузлами сусідніх за підлеглістю рівнів, але й різних рівнів.

Приклади мережевої моделі даних

Перевагами мережевої моделі даних у порівнянні з ієрархічною моделю є її гнучкість, можливість утворення довільних зв'язків, економічність.

Недоліки - висока складність, яка практично виключає можливість її експлуатації користувачами, які не є спеціалістами в області інформаційних технологій, слабкий контроль цілісності зв'язків між об'єктами БД.

Реляційна модель даних

Враховуючи всю складність ієрархічної та мережевої моделей, найбільше поширення отримала **реляційна модель даних** (**relation** - відношення -

математичний термін для визначення непідпорядкованої сукупності записів одного типу або таблиць визначеного специфічного вигляду).

Вся інформація в реляційній моделі даних організована у вигляді таблиць, які пов'язані між собою за допомогою зв'зків.

До реляційних СУБД можна віднести - MS Access, Firebird, SQL Server, Oracle, DB2.

Реляційні системи беруть початок в математичній теорії множин. Вони були запропоновані наприкінці 1968 р. доктором Е. Ф. Коддом з фірми IBM, який першим усвідомив, що можна використовувати математику для надання надійної основи та суворості області керування базами даних.

Нечіткість багатьох термінів, які використовуються в сфері обробці даних, змусила Е. Ф. Кодда відмовитися від них та вигадати нові або дати найбільш точні визначення існуючим. Таким чином, він не міг використовувати поширений термін «Запис», який в різних ситуаціях може визначати екземпляр запису, або тип записів, запис у стилі Коболу (який припускає групи, які півторюються) або плоский запис (який їх не припускає), логічний запис або фізичний запис, зберігаємий запис або віртуальний запис та т.д. Замість цього він використовував термін **«Кортеж довжини n»** або просто **«Кортеж»**.

Реляційна модель даних (РМД) покладена в основу більшості сучасних СУБД. Перевагами моделі вважаються простота розміщення даних та зручність їх інтерптетації.

Реляційна модель орієнтована на організацію даних у вигляді таблиць (відношень).

Для РМД існує достатньо суворе теоретичне обгрунтування. Представлення даних у вигляді відношень дозволяє використовувати для обробки даних формальний математичний апарат реляційної алгебри відношень та реляційного числення. Поняття таблиці та відношення з практичної точки зору представляють собою одне й теж саме, тому в подальшому будуть використовуватися обидва ці терміни.

Кожна таблиця реляційної БД має ім'я та рядок заголовків.

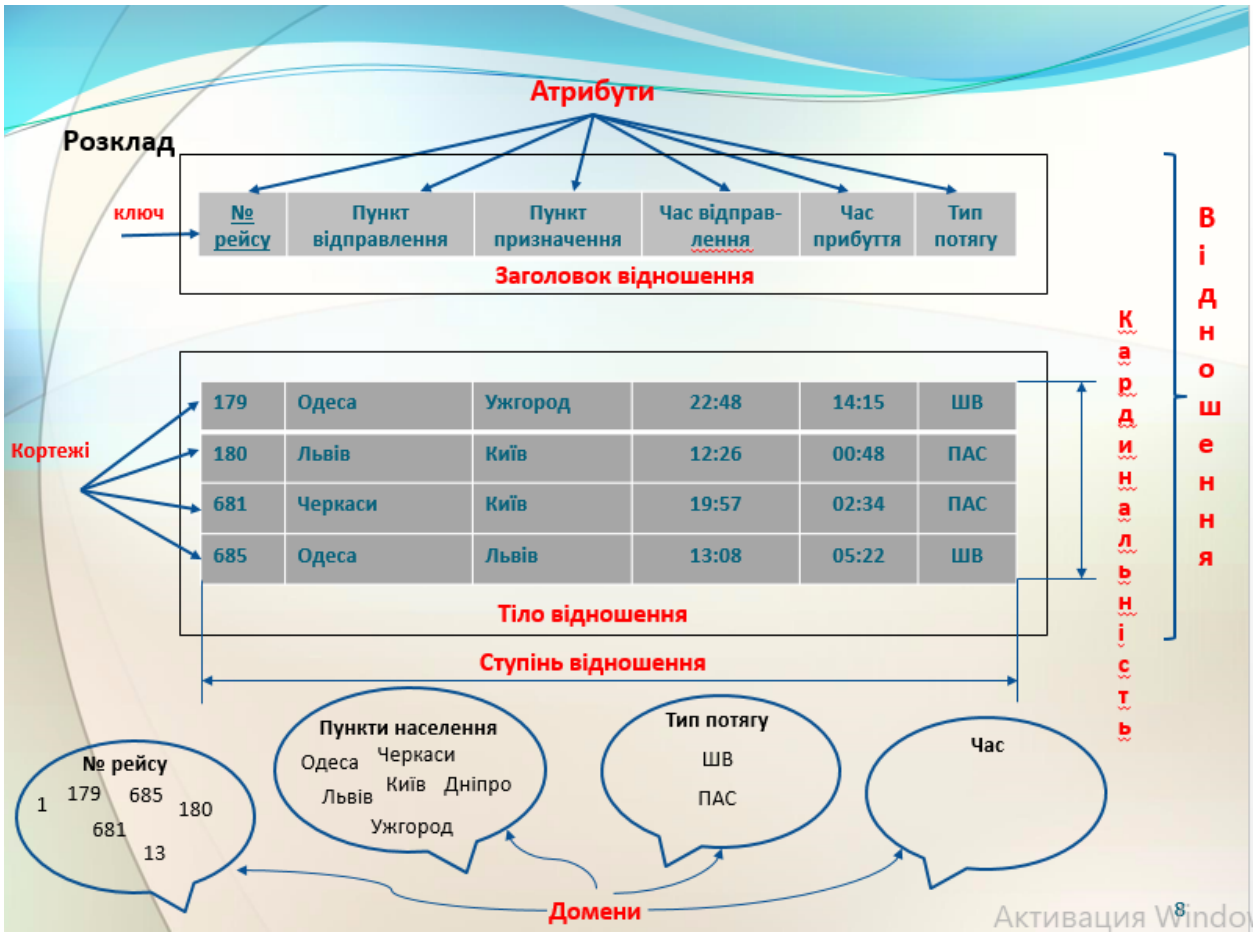
Структура реляційних даних

Відношення - плоска таблиця, яка складається з рядків та стовпців.

В будь-якій реляційній СУБД передбачається, що користувач сприймає базу даних як набір таблиць.

Атрибут - стовпець відношення з власним іменем.

В реляційній моделі відношення використовується для зберігання інформації про об’єкти, які зображуються в базі даних. Відношення має вигляд двомірної таблиці, в якій рядки відповідають окремим записам, а стовпці - атрибутам. При цьому атрибути можуть бути розташовані в будь-якому порядку - незалежно від їх перерозташування відношення залишиться тим самим та матиме той самий зміст.



Кортеж - рядок відношення.

В відношенні «Відділення» кожен рядок містить шість значень, по одному для кожного атрибуту. Кортежі можуть розтошовуватися в будь-якій черзі, при цьому відношення буде залишатися тим самим та мати той самий зміст.

Кортежі називають ***поширенням***, ***станом*** або ***тілом відношення***, яке постійно змінюється.

Ступінь - визначається кількістю атрибутів, яку містить відношення.

Відношення «Відділення» має 6 атрибутів та його ступінь дорівнює 6. Це означає, що кожен рядок таблиці є 6-арним кортежем, тобто кортежем, який містить 6 значень.

Відношення лише з одним атрибутом має ступінь 1 та називається ***унарним відношенням*** (чи 1-арним кортежем). Відношення з двома атрибутами має назву ***бінарне***, відношення з трьома атрибутами - ***тернарне***, а для відношень з більшою кількістю атрибутів використовується термін ***n-арний***.

Кординальність - кількість кортежів, яку містить відношення.

Ця характеристика змінюється при кожному додованні або знищенні кортежів. Кординальність є властивістю тіла відношення та визначається поточним станом відношення в окрему мить.

Для того щоб відрізняти один рядок від іншого використовується поняття *первинного ключа*.

Первинним ключем (Primary key) називається атрибут відношення, значення якого унікальним чином ідентифікує кожен кортеж відношення. У відношення може бути лише один первинний ключ.

Припустимо, що ми маємо два відношення в базі даних: перше відношення містить дані про продавців, а друге - про покупців. У відношенні з даними про покупців є атрибут, в якому записане ім’я продавця, який продав товар.

Було б дуже непродуктивно писати по декілька разів теж саме ім’я продавця. Тут і виникає питання: може існує інша, найвигідніша можливість заповнення цього поля? Зараз ми й познайомимося ближче з поняттям «зовнішнього ключу».

Атрибут одного відношення, значення в якому співпадають зі значеннями атрибуту, який є первинним ключем іншого відношення, має назву **зовнішній ключ (Foreign key).**

Продавці

Код1	ПІБ
1	Симонова С.І.
2	Ветрова В.Р.

Покупці

Код2	ПІБ
1	Петров А.В
2	Коваль М.Л.

Угода

Код	Код1	Код2	Сума
1	2	2	300
2	1	2	500

В цьому випадку Код1 та Код2 в відношенні «Угоди» є зовнішніми ключами, які відповідають первинним ключам в відношенні «Продавці» (ключове поле Код1) та в відношенні «Покупці» (ключове поле Код2).

Поняття **тип даних** в реляційній моделі даних повністю адекватне поняттю типу даних в мовах програмування.

Звичайно в сучасних реляційних базах припускається зберігання символьних, числових даних (таких як «гроші»), а також спеціальних «темпоральних» даних (дата, час, часовий інтервал).

Домен - набір припустимих значень одного або декількох атрибутів.

Кожен атрибут реляційної бази даних визначається на деякому домені. Домени можуть відрізнятися для кожного з атрибутів, але два або більша кількість атрибутів можуть визначатися на тому ж самому домені.

Атрибут	**Ім'я домену**	**Зміст домену**	**Визначення домену**
Номер_ відділення	Номер_ відділення	Безліч усіх припустимих номерів відділень компанії	Символьний; розмір 3; діапозон „В1 – В99"
Місто	Назва_міста	Безліч усіх назв міст в Україні	Символьний; розмір 15.
Поштовий_ індекс	Поштовий_ індекс	Безліч усіх почтових кодів в Україні	Цілий
Стать	Стать	Визначення статі людини	Символьний; розмір 1; Значення „ч" чи „ж"
Дата_ народження	Дата_ народження	Усі можливі значення дат народження співробітників компанії	Дата; діапозон від 1 січня; формат дд-мм-рррр.
З_П	З_П	Усі можливі значення з_п співробітника	Грошовий; 7 цифр

Поняття **домену** має велике значення, так як завдяки йому користувач має централізовано визначити глузд та джерело значень, які можуть отримувати атрибути. В підсумку при виконанні реляційної операції системі стає доступне більше інформації, що дозволяє їй уникнути семантично некоректних операцій.

Наприклад, безглуздо порівнювати назву вулиці з номером телефону, навіть якщо для двох цих атрибутів визначеннями доменів є символьні рядки.

Властивості реляційної таблиці:

1. Таблиця являє собою двовимірну структуру, що складається з рядків і стовпців
2. Кожен рядок таблиці (кортеж) являє собою окрему сутність усередині набору сутностей
3. Кожен стовпець таблиці являє собою атрибут, і в кожного стовпця є своє ім'я
4. На кожному перетині рядка і стовпця є єдине значення
5. Кожна таблиця повинна мати атрибут, що унікально ідентифікує кожен рядок
6. Усі значення в стовпці мають відображатися в однаковому форматі. Наприклад, якщо атрибуту присвоюється формат цілого, то всі значення в стовпчику, що представляє цей атрибут, мають бути цілими
7. Кожен стовпець має певний діапазон значень, званий доменом атрибута (attribute domain)
8. Порядок слідування рядків і стовпців для СУБД не суттєвий.

Тема 1.2.2 Реляційна цілісність БД. З'язки в реляційній моделі даних

(2 години)

Мета: засвоєння двох важливих правил цілісності даних, які існують в реляційній моделі даних; визначення типів зв’язків, які підтримуються в реляційній моделі даних.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Поняття «цілісність бази даних» та її призначення.
2. Цілісність сутностей.
3. Цілісність за посиланням.

4. Поняття зв’язків в реляційній базі даних. Типи зв’язків в реляційній базі даних.
5. Приклади зв’язків в реляційній базі даних.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 1.2.2.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що таке цілісність бази даних?
2. Які дві базові вимоги забезпечення цілісності існують в реляційній моделі?
3. Що повинне мати відношення для здійснення унікальності записів?
4. Зі значенням чого повинен співпадати зовнішній ключ?
5. Які вимоги та призначення цілісності на рівні сутності?
6. Які вимоги та призначення цілісності на рівні посилань?
7. Які типи зв’язків існують в реляційній моделі даних?
8. Яку роль виконують первинні та зовнішні ключі при створенні зв’язків в реляційній моделі даних?
9. Які приклади зв’язків Ви можете навести?

Цілісність даних (цілісність відношення) - це механізм підтримки відповідності бази даних, за якого обрані дані завжди дають один і той самий результат і відповідають певним правилам.

В реляційній моделі даних визначено дві базові вимоги забезпечення цілісності:

- цілісність сутностей (категорна сутність)
- цілісність посилань.

Обмеження цілісності

1. **цілісність сутностей (категорна сутність)**, згідно з якою відношення повинне мати первинний ключ (PRIMARY KEY) для здійснення унікальності записів, отже, у таблиці не може бути 2-х однакових об'єктів;

2. **цілісність посилань** ґрунтується на механізмі вторинних ключів (зовнішніх - Foreign Key), сенс якої полягає в тому, що деякому атрибуту (групі атрибутів) одного відношення призначається посилання на первинний ключ іншого відношення, отже закріплюються зв'язки підпорядкованості між цими відношеннями.

Цілісність на рівні сутності

У межах таблиці первинний ключ має бути унікальним, щоб однозначно ідентифікувати кожен рядок. Коли справа йде так, то кажуть, що таблиця проявляє **цілісність на рівні сутності**.

Для забезпечення цілісності на рівні сутності в первинному ключі неприпустимі порожні значення, null (тобто повна відсутність даних).

Правило категорної цілісності: жоден ключовий атрибут будь-якого рядка реляційної таблиці не може мати порожнього значення.

Вимога цілісності на рівні сутності - Всі первинні ключі унікальні й не можуть містити порожнього значення (null).

Призначення цілісності на рівні сутності - Гарантує, що кожна сутність (логічний об'єкт) матиме унікальну ідентифікацію, а значення зовнішнього ключа можуть належним чином посилатися на значення первинного ключа.

Приклад - Рахунок не може мати кілька значень, що дублюються, і не може мати порожнє значення (null). Тобто, всі рахунки унікально ідентифікуються своїм номером.

Цілісність на рівні посилань

Якщо зовнішній ключ містить значення, що збігаються з первинним ключем або порожні значення (null), кажуть, що таблиця (або таблиці), яка використовує такий ключ, проявляє **цілісність на рівні посиланнь**.

Іншими словами цілісність на рівні посилання означає, що в тому разі, якщо зовнішній ключ містить якесь значення, то це значення посилається на наявний дійсний кортеж (рядок) в іншому відношенні.

Вимога цілісності на рівні посилань - Зовнішній ключ може мати або порожнє значення, або значення, що збігається зі значенням первинного ключа у пов'язаній таблиці (Кожне непорожнє значення зовнішнього ключа повинно посилатися на наявне значення первинного ключа).

Призначення цілісності на рівні посилань - Допускається, що атрибут не має відповідного значення, але атрибут не може приймати неприпустимі значення. Виконання правила цілісності на рівні посилання унеможливлює видалення рядка в одній таблиці, де первинний ключ має обов'язкову відповідність зі значенням зовнішнього ключа в іншій таблиці.

Приклад - Клієнту може бути не призначений (ще) торговий агент, але неможливо призначити клієнту неіснуючого агента.

Зв’язки в реляційній моделі даних

У кожному зв'язку одне відношення може виступати як основне, а інше відношення виступає в ролі підлеглого.

Таким чином, один кортеж основного відношення може бути пов'язаний з кількома кортежами підлеглого відношення.

Для підтримки цих зв'язків обидва відносини мають містити набори атрибутів, за якими вони пов'язані.

В основному відношенні це **первинний ключ** відношення, який однозначно визначає кортеж основного відношення.

У підлеглому відношенні для моделювання зв'язку має бути присутнім набір атрибутів, що відповідає первинному ключу основного відношення.

Однак тут цей набір атрибутів уже є **вторинним ключем** або **зовнішнім ключем**, тобто він визначає безліч кортежів відношення, які пов'язані з єдиним кортежем основного відношення.

Типи зв’язків в реляційній моделі даних

- зв'язок "один до одного" (1:1) - кожному значенню сполучного поля відповідає один запис по обидва боки.
- зв'язок "один до багатьох" (1:N) - кожному значенню сполучного поля з одного боку відповідає декілька записів по інший бік.
- зв'язок "багато до багатьох" (M:N) - значення в полях неодноразово зустрічаються у пов'язаних відносинах. Такий тип зв'язку необхідно обходити і розділяти його на зв'язки 1:N (створюється третє відношення, яке пов'язується з початковими відносинами зв'язками 1:N).

Приклади зв’язка «один до одного»:

- у людини може бути лише один чоловік або дружина (традиційне подружжя);
- на факультеті може бути лише один декан, та навпаки, один декан може керувати лише одним факультетом.

Приклади зв’язка «один до багатьох»:

- в групі навчається багато студентів, але кожен з цих студентів навчається лише в одній групі;
- у кожного студента лише один класний керівник, а у класного керівника багато студентів в його класній групі;
- у керівника деякого підприємства багато підлеглих, але у кожного підлеглого може бути лише один керівник;
- у чоловіка може бути декілька дружин;
- у дружини може бути декілька чоловіків.

Приклади зв’язка «багато до багатьох»:

- викладач працює в різних групах, та в тій самій групі працюють різні викладачі;
- у чоловіка може бути декілька дружин та у дружини може бути декілька чоловіків.

Блок змістових модулів 2 – Теорія проектування баз даних

Змістовий модуль 2.1 - Інфологічне моделювання предметної області

Тема 2.1.1 Етапи проектування бази даних. Побудова концептуальної моделі предметної області

(2 години)

Мета: знати 3 етапи, які використовуються при проектуванні бази даних та навчитися створювати високорівневу концептуальну модель даних «Сутність – зв'язок».

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Етапи проектування БД.

2. Концептуальна модель даних або модель «Сутність – зв'язок».

3. Побудова ER-діаграм.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 2.1.1.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Які етапи виділяють в процесі проектування БД?

2. Яке призначення мають етапи проектування БД?

3. Для чого призначена модель «Сутність – зв’язок»?

4. Що є головними поняттями моделі «Сутність-зв’язок»?

5. Що таке «сутність»? Навести приклади сутностей.

6. Які існують типи сутностей? Як вони зображуються на діаграмах?

7. Що таке «атрибут»? Навести приклади атрибутів.

8. Які існують типи атрибутів? Як вони зображуються на діаграмах?

9. Як зображується на діаграмах атрибут, який є первинний ключем?

10. Для чого потрібні зв’язки між сутностями? Навести приклади зв’язків.

11. Як зображуються на діаграмах зв’язки?

12. Кількість сутностей,охоплених зв’язком – це .....? Продовжити визначення.

Проектуванню баз даних традиційно приділяється велика увага, так як ця робота в більшості визначає успішність експлуатації бази даних, яка створюється, можливості її модернізації та удосконалення в подальшому.

В процесі проектування БД часто виділяють три етапи.

Етап 1. Побудова концептуальної моделі предметної області

В рамках цього этапу досліджується предметна область – частина реального світу, для якої створюється БД. Вивчаються інформаційні потреби користувачів, виявляються інформаційні об’єкти та зв’язки між ними. Виходячи з отриманої інформації будується концептуальна модель предметної області, незалежна від моделі даних та програмих засобів (включаючи СУБД).

Етап 2. Логічне проектування – перетворення створенної концептуальної моделі в концептуальну схему, яка реалізується конкретною СУБД

На цьому етапі на основі концептуальної моделі розробляється структура БД, яка відповідає обраній для її створення СУБД. Для реляційної БД інформація розбивається на відношення (таблиці); для кожного відношення (таблиці) визначаються атрибути (поля), первинні ключі; відношення приводяться до нормалізованого вигляду; ідентифікуються зв’язки між відношеннями.

Етап 3. Фізичне проектування бази даних

На цьому етапі вирішуються проблеми фізичного розташування БД в зовнішній пам’яті та організації доступу до неї. Фізичне проектування БД реалізується адміністратором банку даних при конфігуруванні та налаштуванні системи. Від спеціалістів, які брали участь в проектуванні БД на попередніх етапах, цей процес може бути повністю прихований.

Розглянемо більш детально кожен з етапів.

Побудова концептуальної моделі предметної області

В рамках концептуальної моделі інформаційний зміст предметної області виражається деякими абстрактними засобами. Основною вимогою, яка висувається до концептуальної моделі, є вимога адекватного відображення предметної області.

Модель має бути несуперечливою, відображати погляди і потреби всіх користувачів системи. Вона повинна володіти властивістю легкої розширюваності, що забезпечує введення нової інформації.

Розглянемо деякі засоби концептуального моделювання.

ER-модель

ER-модель (Entity-Relationship – сутність - зв'язок) була запропонована П. Ченом в 1976 р. Інформація про зміст предметної області в межах моделі зображується в структурованому графічному вигляді (ER-діаграма).

ER-модель (**E**ntity **R**elationship model) або модель «Сутність – зв’язок» – це високорівнева концептуальна модель даних, яка була розроблена з метою спрощування задач проектування баз даних. Ця модель даних являє собою набір концепцій, які описують структуру бази даних та пов’язані з нею транзакції оновлення та вилучення даних.

Для ER-моделі не існує єдиної стандартизованої системи позначень, тому характеристики ER-діаграм можуть дещо відрізнятися.

Головними поняттями моделі «Сутність - зв’язок» вважаються сутності, атрибути та зв’язки.

Під сутністю в ER-моделі розуміються об’єкт або явище, інформація про яких буде зберігатися в базі даних. При цьому розрізняють **тип сутності** та **екземпляр сутності**.

Під типом сутності розуміють набір однорідних об’єктів, який відображається як єдине ціле.

Під екземпляром сутності розуміється конкретний об’єкт.

На ER-діаграмі сутність зображується прямокутником, в якому вказане його ім’я**.**

Типи сутностей можна класифікувати як сильні та слабкі.

Слабкий тип сутності - тип сутності, існування якого залежить від якого-небудь іншого типу сутності.

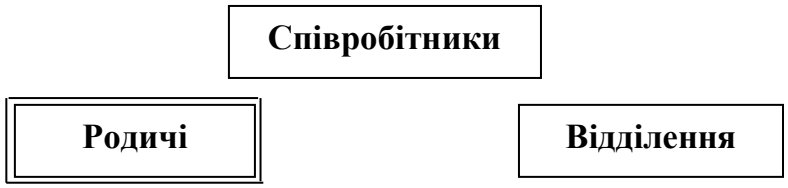
Сильний тип сутності - тип сутності, існування якого не залежить від якого-небудь іншого типу сутності.

Наприклад, сутність **«Родичі»** є сутністю слабкого типу, яка надає відомості про родичів співробітника. Сутність **«Родичі»** не може існувати в цій моделі без наявності сутності **«Співробітники»**.

Прикладами сильних сутностей є сутності **«Співробітники»** та **«Відділення»**.

Слабкі типи сутностей іноді називають ***дочірніми*** (child), ***залежними*** (dependent) або ***підпорядкованими*** (subordinate), а сильні типи сутностей - ***батьківськими*** (parent), ***сутностями-володорями*** (owner) або ***домінантними*** сутностями (dominant).

Кожен сильний тип сутності зображується у вигляді прямокутника з іменем сутності всередині нього, а кожен слабкий тип сутності - у вигляді прямокутника з подвійним контуром.



Сутності мають властивості, які називаються **атрибутами**. Атрибути повинні дозволяти розрізняти екземпляри сутності. На ER-діаграмі атрибути зображуються овалами, в яких вказуються їх імена. Атрибути поєднуються з сутностями прямими лініями.

Атрибути, які однозначно ідентифікують сутність, називаються **ключовими атрибутами**.

Ключові атрибути на ER-діаграмі підкреслюються.

В деяких ситуаціях з декількох простих атрибутів може бути сформований **складовий (складений) атрибут.**

Атрибути поділяються на прості та складові.

Простий атрибут – атрибут, який складається з одного компоненту з незалежним існуванням.

Прості атрибути не можуть бути розподілені на більш дрібні компоненти. Прикладами є атрибути статі («Ч» або «Ж») або «ЗП співробітника».

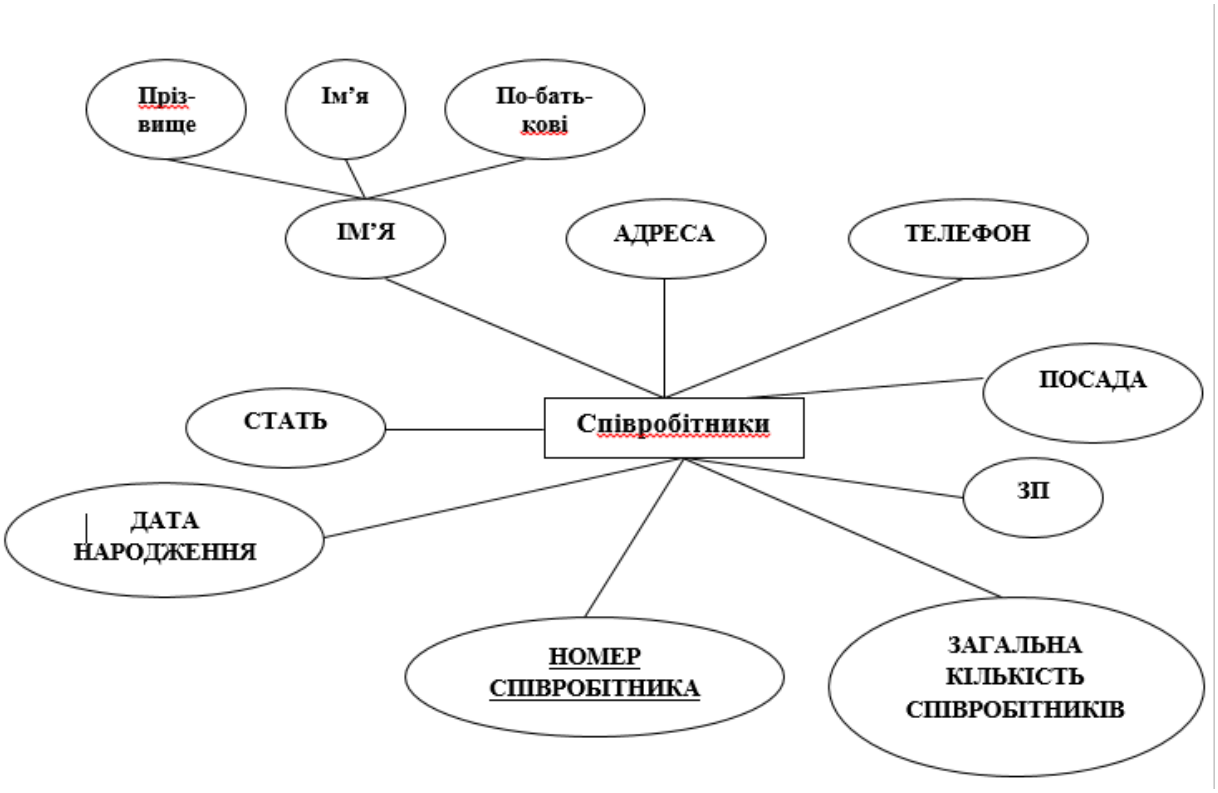
Прості атрибути іноді називають **атомарними**.

Складовий атрибут – атрибут, який складається з декількох компонентів, кожен з яких характеризується незалежним існуванням.

Деякі атрибути можуть бути розподілені на більш дрібні компоненти, які характеризуються незалежним існуванням. Наприклад, атрибут «Адреса» сутності «Відділення» зі значенням «Одеса, вул. Гер. Сталінграда 14, Суворівський, 65120, 568247» може бути розподілений на окремі атрибути «Місто» зі значенням «Одеса», «Вулиця» - зі значенням «Гер. Сталінграда 14», «Район» - зі значенням «Суворівський», «Поштовий індекс» - зі значенням «65120» та «Телефон» – зі значенням «568247».

На рисунку зображені атрибути, які пов’язані з сутностями **«Співробітники»** та **«Відділення»**.

Сутність «Співробітники»

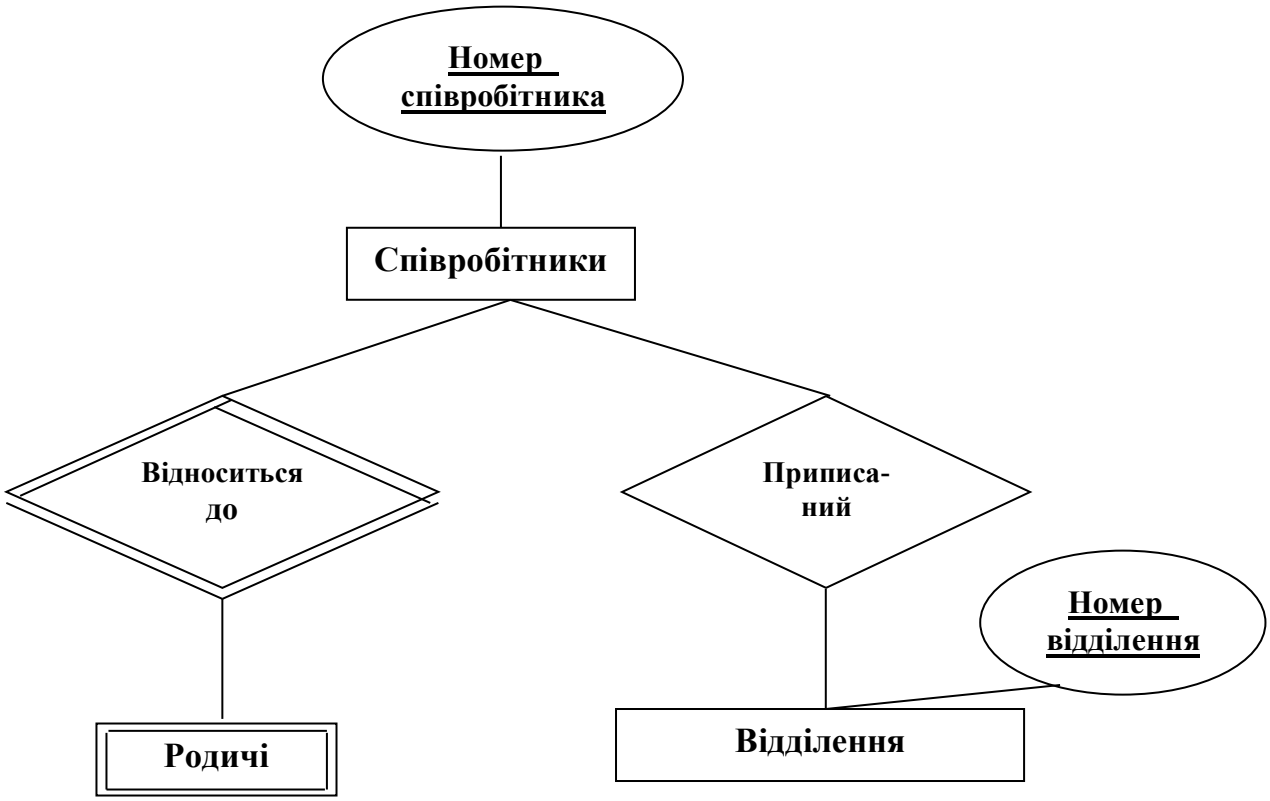


Якщо атрибут є складовим, його атрибути-компоненти зображуються у вигляді еліпсів, які приєднуються до нього (атрибут «Ім’я» сутності **«Співробітники»** є складовим атрибутом, який складається з атрибутів «Прізвище», «Ім’я», «По-батькові»).

Ім’я атрибуту, який є первинним ключем цього типу сутності, підкреслюється.

За допомогою **зв’язків** на ER-діаграмі відображається взаємодія між сутностями. Зв’язок зображується ромбом, який поєднує сутності. Всередині ромбу

вказується вид зв’язка. Ромб має подвійний контур, якщо зв’язок пов’язує слабку сутність з сильною сутністю, від якої ця слабка сутність залежить.



На рисунку взаємозв’язок між сутностями «Відділення» та «Співробітники» зображений за допомогою зв’язка «Приписаний до», а між сутностями «Родичі» та «Співробітники» - за допомогою зв’язка «Відноситься до», який зображений у вигляді ромбу з подвійним контуром та вказує на те, що він встановлений між слабкою (сутність «Родичі») та сильною (сутність «Співробітники») сутностями.

Для зниження рівня деталізації на окремій ER-діаграмі часто вказують лише ті атрибути, які уявляють первинні ключі зображених сутностей, а в деяких випадках атрибути зовсім не відображаються.

Наприклад, на рисунку зображені лише ті атрибути, які є первинними ключами сильних сутностей, а саме: «Номер співробітника» та «Номер відділення».

Ступінь зв’язка – кількість сутностей, які охоплені цим зв’язком.

Охоплені деяким зв’язком сутності мають назву ***учасники цього зв’язка***.

Кількість учасників деякого зв’язка має назву ступінь цього зв’язку. Тобто, ступінь зв’язка вказує на кількість типів сутностей, охоплених цим зв’язком.

Зв’язок зі ступнем 2 називається бінарним, зі ступенем 3 – тернарним, зі ступенем 4 – кватернарним.

Попередній приклад є прикладом тернарного зв’язка (приймають участь 3 типи сутності: «Співробітники», «Відділення» та «Родичі»).

Тема 2.1.2 Логічне проектування бази даних

(2 години)

Мета: навчитися здійснювати логічне проектування бази даних, тобто перетворювати модель «Сутність – зв'язок» в набір таблиць та пов'язувати ці таблиці зв'язками.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Поняття та призначення логічного проектування бази даних.
2. Приклади перетворення концептуальної моделі, представленої у вигляді ER – діаграми в логічну модель реляційної бази даних

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичними відомостями теми 2.1.2.
   2. Вивчити основні поняття лекції.
   3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Для чого призначене логічне проектування бази даних?
2. Яким чином здійснюється логічне проектування бази даних?

Логічне (даталогічне) проектування - створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних.

Для реляційної моделі даних даталогічна модель - набір схем відношень, зазвичай із зазначенням первинних ключів, а також "зв'язків" між відношеннями, що являють собою зовнішні ключі. Перетворення концептуальної моделі на логічну модель, як правило, здійснюється за формальними правилами. Цей етап може бути значною мірою автоматизовано.

На етапі логічного проектування враховується специфіка конкретної моделі даних, але може не враховуватися специфіка конкретної СУБД.

Фаза логічного проектування передбачає такі дії:

✕ перетворення концептуальної моделі даних на логічну модель, унаслідок якого буде визначено схему реляційної моделі даних;

✕ перевірка моделі за допомогою концепцій послідовної нормалізації;

✕ перевірка моделі щодо транзакцій користувачів.

Спрощення концептуальної моделі даних

Перетворення концептуальної моделі даних на логічну модель, у результаті якого буде визначено схему реляційної моделі даних, може мати два підходи.

1) Проектувальник працює з концептуальною моделлю безпосередньо, не вдаючись до її попереднього перетворення. У цьому випадку йому доведеться зіткнутися з необхідністю перетворення різноманітних структур даних
2) Проектувальник, перш ніж розпочати процес переходу від концептуальної моделі до логічної моделі, прагне спочатку цей перехід максимально спростити, провівши попередні перетворення концептуальної моделі, перетворення деяких її структур даних, які не підходять для реляційних СУБД.

До таких структур даних належать:

1. зв'язки типу "багато до багатьох";
2. складні зв'язки;
3. рекурсивні зв'язки;
4. зв'язки з атрибутами;
5. множинні атрибути;
6. надлишкові зв'язки.

Якщо в моделі присутні перелічені небажані структури, то їх потрібно виключити шляхом тотожної їхньої заміни на структури даних, допустимі для логічної моделі.

Виключення зв'язку типу "багато до багатьох"

Перетворення зв'язку типу "багато до багатьох" здійснюється шляхом введення деякого проміжного об'єкта із заміною одного зв'язку двома зв'язками типу "один до багатьох" із новоствореним об'єктом. Під час організації нових зв'язків необхідно стежити за тим, щоб максимальна потужність зв'язку "один" була спрямована до вихідного об'єкта, а максимальна потужність зв'язку "багато" - до новоствореного об'єкта.

Розглянемо основні правила перетворення ER-моделі в логічну модель бази даних на прикладі діаграми, побудованої за темою курсової роботи. Для прикладу розглянемо тему «Розробити автоматизоване робоче місце менеджера з продажу магазину комп’ютерної техніки».

На рис. 1 зображена модель «Сутність – зв'язок» для даної предметної області, а на рис. 2 ця модель вже перетворена в логічну модель реляційної бази даних.

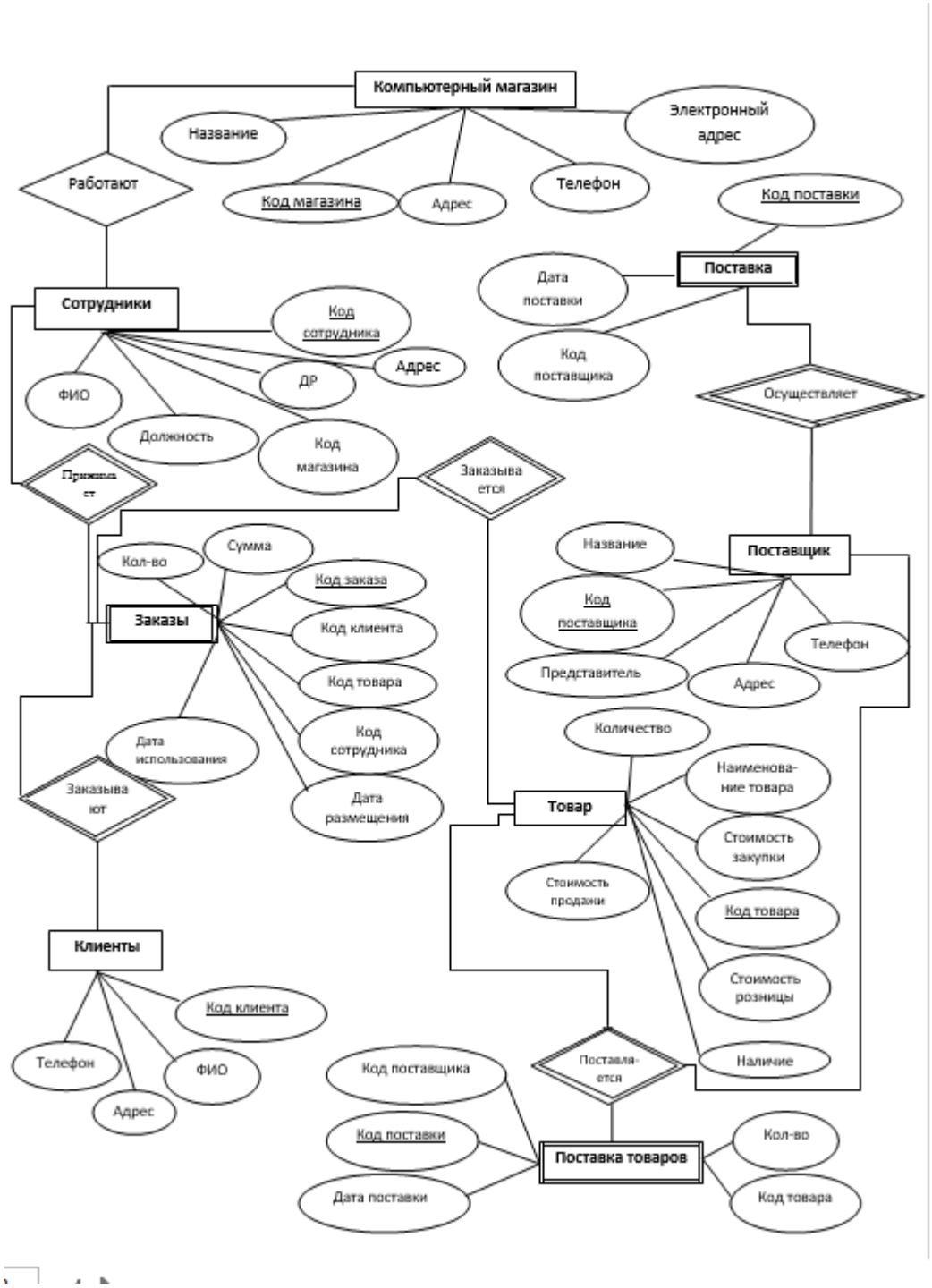


Рис. 1 - Модель «Сутність – зв'язок» для предметної області «Комп'ютерний магазин»

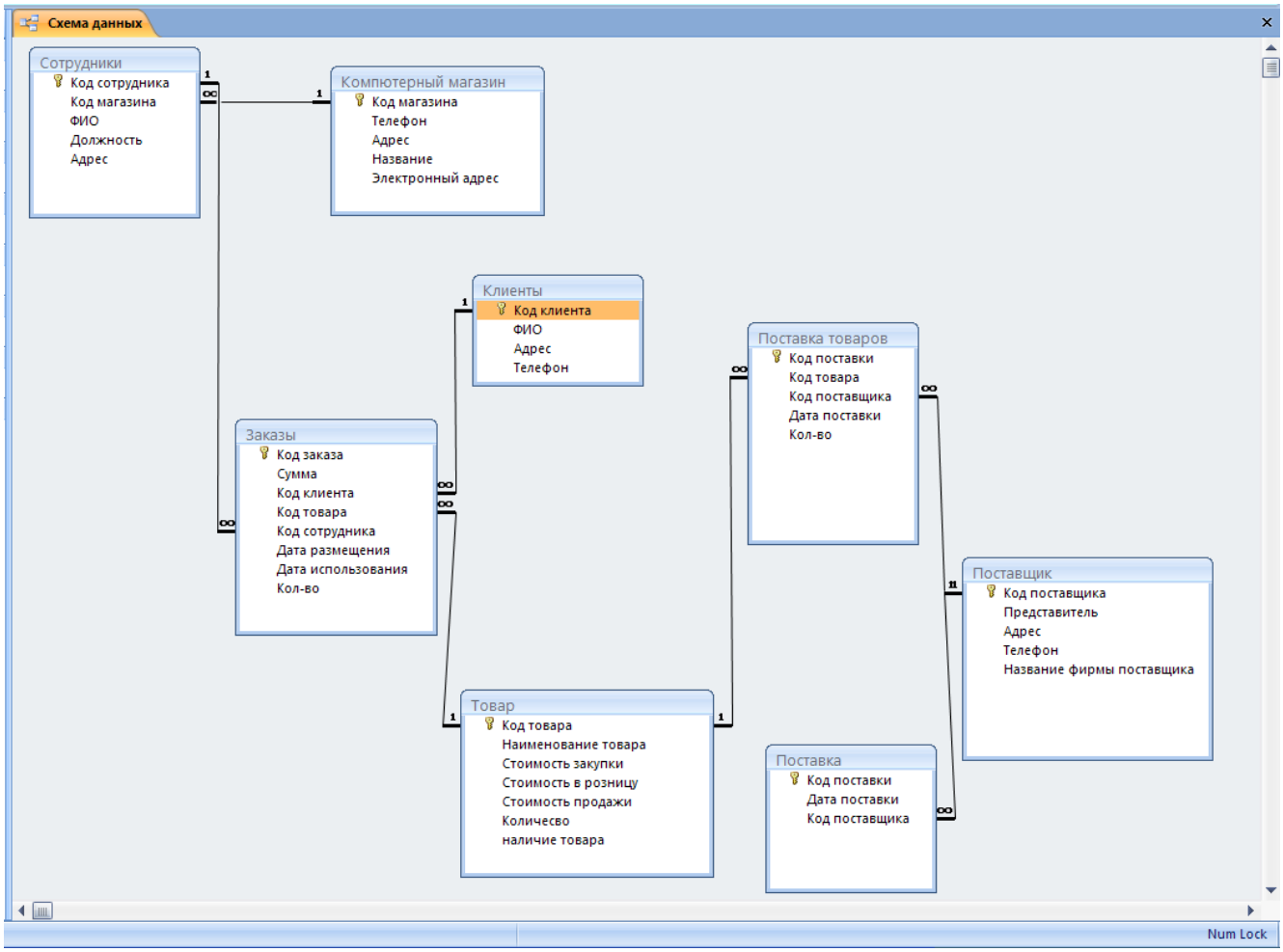


Рис. 2 Структурна схема реляційної БД для предметної області «Комп'ютерний магазин»

Розглянуті правила перетворення ER-моделі в логічну модель бази даних досить прості, наочні і дозволяють отримати добрі результати.

Змістовий модуль 2.2 - Процес нормалізації

Тема 2.2.1 Нормалізація відношень

(2 години)

Мета: знати про процес нормалізації та призначення кожної нормальної форми.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Поняття та призначення нормалізації.
2. Нормальні форми.
3. Переваги та недоліки нормалізації.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

4. Ознайомитись з теоретичними відомостями теми 2.2.1.
5. Вивчити основні поняття лекції.
6. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що розуміється під поняттям нормалізації?
2. Які нормальні форми існують в реляційній теорії?
3. В чому зміст 1 нормальної форми. Навести приклад бази даних, доведеної до 1 нормальної форми.
4. В чому зміст 2 нормальної форми. Навести приклад бази даних, доведеної до 2 нормальної форми.
5. В чому зміст 3 нормальної форми. Навести приклад бази даних, доведеної до 3 нормальної форми.
6. Якими перевагами володіє нормалізація?
7. Завдяки якій перевазі нормалізації спрощується структура бази даних та економиться дисковий простір?
8. Якими недоліками володіє нормалізація?

Під **нормалізацією** розуміється процес приведення відношення до однієї з нормальних форм (НФ). Але перед тим як розглядати нормальні форми, треба розповісти, для чого потрібна нормалізація.

При проектуванні баз даних в першу чергу робиться наголос на вірогідність та несуперечливість зберігаємих даних, до того ж, ці властивості не повині втрачатися в процесі роботи з даними, тобто після багаточисельних змін, знищень та доповнень даних по відношенню до першопочаткового стану бази даних.

Для підтримки бази даних в стійкому стані використовується ряд механізмів, які отримали узагальнену назву **засобів підтримки цілістності**. Ці механізми використовуються **статично** (на етапі проектування бази даних), так й **динамічно** (в процесі роботи з базою даних).

База даних повинна володіти обмеженнями, які будуть задовільняти її в процесі створення, незалежно від її наповнення даними. Доведення структури бази даних до відповідності цим обмеженням – це **нормалізація**. В цілому суть цих обмежень дуже проста: кожен факт, який зберігається в базі даних, повинен зберігатися один єдиний раз, так як дублювання може призвести до незгоди між копіями однакової інформації. Треба уникати невизначеностей, а також надмірності зберігаємої інформації. Отже, наша **мета** – **доведення відношень до НФ**. Треба відмітити, що в процесі нормалізації постійно зустрічається ситуація, коли відношення необхідно розкласти на декілька інших відношень. Тому коректніше було б казати про нормалізацію не окремих відношень, а всієї їх сукупності в базі даних.

Нормалізація може не викликати проблему, якщо база даних проектується одразу ж за визначеними правилами. Іншими словами, можна спочатку створити базу даних як-небудь, а потім нормалізувати її, або ж з самого початку будувати її за правилами для того, щоб в подальшому не треба було б її переробляти.

Нормалізація відношень забезпечує ефективність структур даних в реляційній БД. Цей процес зменшує надмірність даних (зберігання однакових даних в декількох місцях). В результаті більш раціонально використовується зовнішня пам’ять, зменшується ймовірність порушення узгоджуваності даних.

Нормалізація - це послідовне перетворення початкової бази даних до НФ, при цьому кожна наступна НФ обов’язково містить у собі попередню.

В реляційній теорії нараховують 6 НФ:

- 1 НФ;
- 2 НФ;
- 3 НФ;
- НФ Бойса-Кодда (НФБК);
- 4 НФ;
- 5 НФ.

На практиці обмежуються 3 НФ, її достатньо для створення надійної схеми бази даних. НФ більш високих порядків уявляють академічну зацікавленість із-за надмірної складності. Крім того, при реалізації абстрактної схеми бази даних у вигляді реальної бази іноді розробники змушені зробити крок назад – провести денормалізацію з метою підвищення ефективності, так як ідеальна з точки зору теорії структура може статися занадто накладною на практиці.

Основні властивості нормальних форм:

• кожна наступна нормальна форма покращує властивості попередньої нормальної форми;

• при переході до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

Перша нормальна форма (1НФ)

Відношення знаходиться в 1 НФ, якщо на перетині кожного стовпця з кожним рядком міститься лише атомарне (нерозподілене) значення. Іншими словами, кожен атрибут відношення повинен зберігати одне єдине значення та не бути ні списком, ні множиною значень. Атрибут, який є атомарним в одному додатку, може виявитися складовим в іншому.

Приклад. В базі даних відділу кадрів фірми в таблиці, в якій зберігаються особисті відомості про співробітників, є атрибут «Домашня адреса», в якому адреса зберігається в форматі: місто, вулиця, будинок[/корпус], квартира (в даному випадку адреса зберігається у вигляді єдиного текстового рядка, так як малоймовірно, щоб необхідно було б обрати співробітників, наприклад, за номером квартири). Таким чином, в контексті бази даних відділу кадрів адреса є атомарним поняттям, та його ділення на складові частини немає сенсу, так як лише привнесе до бази даних надмірну громіздкість. Але та сама адреса для додатку, призначеного для сортування пошти у поштовому відділенні компанії, не є атомарним, тому що бажано було б згрупувати конверти в окремі купи по вулицям, так як кожну вулицю обслуговує свій листоноша. Крім цього, з метою оптимізації переміщення листоноші в межах вулиці, кожну купу бажано було б відсортувати за номерами будинків, для того щоб зробити можливим рознесення пошти за один прохід по вулиці без повернення.

Доведення відношення до 1 НФ – проста операція. Треба проглянути відношення та розподілити складові атрибути на різні рядки/стовпці. Можливо, цю

ситуацію доведеться повторити декілька разів до тих пір, доки кожен із атрибутів не стане атомарним.

Приклад. В базі даних міститься таблиця підприємств, в якій зберігаються такі відомості:

- найменування підприємства;
- місто;
- адреса;
- електронна адреса;
- веб-сторінка;
- вид агенту (постачальник або клієнт);
- контактні особи (може бути декілька), посада контактної особи, телефон контактної особи.

Табл. 1

Найм-ня	Місто	Адреса	Ел. пошта	Веб-стор.	Вид	Конт. особа
Поршневий з-д	Київ	Вул. 2-а Кольцева, 17	info@ plunger.gmail.com	www. plunger.ua	Постачаль-ник	Іванов І. І., заступник директора, тел (044)76-15-95 Петров П. П., начальник від. збуту, тел (044)76-15-35
ПП«Вимпел»	Одеса	Вул. Гоголя, 25	pennon@ gmail.com		Клієнт	Сидоров С. С., директор, тел. (048)66-65-38
ПП «Альфа»	Київ	Вул. Пушкінська, 37, оф. 565	alpha@ gmail.com		Клієнт	Васильєв В.В., директор, тел (044)74-57-45

В даному випадку атрибут «Контактна особа» не є атомарним, тому що в ньому зустрічаються списки з декількох осіб. Розподілимо ці кортежі таким чином, щоб кожен кортеж містив дані тільки про одну особу:

Табл. 2

Найм-ня	Місто	Адреса	Ел. пошта	Веб-стор.	Вид	Конт. особа
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@ plunger.gmail.com	www. plunger.ua	Постачальник	Іванов І. І., заступник директора, тел (044)76-15-95
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@ plunger.gmail.com	www. plunger.ua	Постачальник	Петров П. П., начальник від. збуту, тел (044)76-15-35
ПП «Вимпел»	Одеса	Вул. Гоголя, 25	pennon@ gmail.com		Клієнт	Сидоров С.С., директор, тел. (048)66-65-38
ПП «Альфа»	Київ	Вул. Пушкінська, 37, оф. 565	alpha@ gmail.com		Клієнт	Васильєв В.В., директор, тел (044)74-57-45

Трохи краще, хоча якщо придивитися виявляється, що атрибут «Контактна особа» може бути атомарним знов з натягуванням, так як містить дані різного роду, хоча й про одну особу. Розподілемо його на декілька атрибутів:

Табл. 3

Найм-ня	Місто	Адреса	Ел. пошта	Веб-сторінка	Вид	Посада	ПІБ	Код міста	Тел.
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@ plunger.gmail.com	www. plunger.ua	Постачальник	Заст. дир.	Іванов І.І.	044	76-15-95
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@ plunger.gmail.com	www. plunger.ua	Постачальник	Нач. від. збуту	Петров П.П.	044	76-15-35
ПП «Вимпел»	Одеса	Вул. Гоголя, 25	pennon@ gmail.com		Клієнт	Директор	Сидоров С.С.	048	66-65-38
ПП «Альфа»	Київ	Вул. Пушкінська, 37, оф. 565	alpha@ gmail.com		Клієнт	Директор	Васильєв В.В.	044	74-57-45

Зараз можна вважати, що кожне значення кожного з атрибутів нашого відношення є атомарним. Отже, відношення знаходиться в 1 НФ.

Друга нормальна форма (2НФ)

Зараз наше відношення має коректний вигляд, але не досконалий. Значення, які повторюються в таблиці, є потенційним джерелом проблем.

По-перше, при заповненні таких записів легко помилитися. Наприклад, достатньо змінити лише один символ в атрибуті «Адреса», та це стане зовсім другою адресою, яка немає нічого спільного з першою адресою. Знайти такі помилки в величезній таблиці – дуже складна задача.

По-друге, назва вулиці може змінитися. Може змінитися й адреса веб-сайту фірми. Тоді треба буде проглядати всю таблицю та змінювати відповідні значення.

По-третє, не можна зневажати ефективністю зберігання інформації. З нашої таблиці можна було б як мінімум двічі дізнатися адресу поршневого заводу, хоча достатньо було б одного разу. Таким чином, боротьба з надмірністю даних є найбільш вигідною з усіх боків – як з боку підвищення зручності користування даними та підтримки їх в цілісному вигляді, так й з боку ефективності обробки та зберігання даних апаратними засобами серверу баз даних. Саме на усунення цієї надмірності спрямована подальша нормалізація відношень.

Відношення знаходиться в другій нормальній формі, якщо воно знаходиться в першій нормальній формі, і кожен неключовий атрибут (тобто що не є складовою частиною первинного ключа) функціонально повно залежить від первинного ключа.

Припустимо, що при постановці задачі замовник повідомив нам, що в межах кожного міста найменування підприємств є унікальним, але в різних містах найменування можуть співпадати. Таким чином, підприємство характеризується складовим ключем «Найменування + Місто». Подивимося на наше відношення з точки зору того, як ми тільки що дізналися про 2 НФ. Явно, що телефонний код міста залежить виключно від самого міста та жодним чином не пов’язаний з найменуванням підприємства. Звідси й одне з джерел надмірності даних – скільки разів в таблиці зустрічається рядок, який містить відомості про контактну особу будь-якого підприємства, яке знаходиться в даному місті, стільки й повторюється інформація про код міста. Для усунення цієї надмірності, треба розподілити наше відношення на декілька відношень:

Табл. 4а

Найм-ня	Місто	Адреса	Ел. пошта	Веб-сторінка	Вид	Посада	ПІБ	Тел.
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@plunger.gmail.com	www.plunger.ua	Постачальник	Заст. дир.	Іванов І.І.	76-15-95
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@plunger.gmail.com	www.plunger.ua	Постачальник	Нач. від. збуту	Петров П.П.	76-15-35
ПП «Вимпел»	Одеса	Вул. Гоголя, 25	pennon@gmail.com		Клієнт	Директор	Сидоров С.С.	66-65-38
ПП «Альфа»	Київ	Вул. Пушкінська, 37, оф. 565	alpha@gmail.com		Клиієт	Директор	Васильєв В.В.	74-57-45

Табл. 4б

Місто	Код міста
Київ	044
Одеса	048

Отже, ми усунули часткову залежність атрибута «Код міста» від складового ключа, перемістивши коди міст в окреме відношення з ключем «Місто». Таким чином, ми отримали два відношення, кожне з яких знаходиться в 2 НФ.

Не дивлячись на здійснені зусилля, таблиця 4а все ж таки містить надмірність – достатньо поглянути на значення в стовбцях «Адреса», «Ел. пошта» та «Веб. сторінка», які повторюються. Це значить, що процес нормалізації не є завершуваним, та треба приступати до його наступного етапу.

Третя нормальна форма (3НФ)

Існує функціональна залежність між атрибутами «ПІБ», «Посада» та «Телефон». Є очевидним, що на підприємстві деяка людина займає визначену посаду та розпоряджається визначеним робочим телефоном. Зворотнє в спільному випадку не є вірним – на підприємстві може існувати декілька аналогічних штатних одиниць, наприклад, менеджери за збутом, та декілька чоловік мають можливість користуватися одним робочим телефоном.

Для того, щоб уникнути цю функціональну залежність, треба зробити декомпозицію таблиці 4а на дві таблиці. Перша з них буде зберігати факти, які відносяться безпосередньо до самого підприємства:

Табл. 5а

Найм-ня	Місто	Адреса	Ел. пошта	Веб-сторінка	Вид
Поршневий з-д	Київ	Вул. 2-я Кольцева, 17	info@ plunger.gmail.com	www.plunger.ua	Постачаль-ник
ПП «Вимпел»	Одеса	Вул. Гоголя, 25	pennon@gmail.com		Клієнт
ПП «Альфа»	Київ	Вул. Пушкінська, 37, оф. 565	alpha@ gmail.com		Клієнт

Друга таблиця буде зберігати факти, які відносяться до конкретної особи, яка виконує обов'язки на цьому підприємстві:

Табл. 5б

Найм-ня	Місто	ПІБ	Посада	Тел.
Поршневий з-д	Київ	Іванов І.І.	Заступник директора	76-15-95
Поршневий з-д	Київ	Петров П.П.	Начальник від. збуту	76-15-35
ПП «Вимпел»	Одеса	Сидоров С.С.	Директор	66-65-38
ПП «Альфа»	Київ	Васильєв В.В.	Директор	74-57-45

Разом з таблицею 4б цей набір таблиць уявляє собою нашу першопочаткову базу даних, наведену до 3 НФ.

Відношення знаходиться в третій нормальній формі, якщо воно знаходиться в другій нормальній формі та кожен його неключовий атрибут безпосередньо (не транзитивно) залежить від первинного ключа.

В більшості випадків досягнення третьої нормальної форми вважається достатнім для реальних проектів баз даних, проте в теорії нормалізації існують нормальні форми вищих порядків (НФБК, 4НФ, 5НФ), деякі з яких пов'язані вже не з функціональними залежностями між атрибутами стосунків, а відображають тонші питання смислового вмісту наочної області.

Переваги нормалізації:

- краща загальна організація бази даних;
- скорочення кількості непотрібного повторювання даних;
- узгоджування даних в базі даних;
- більш гнучка структура бази даних;
- ефективні можливості забезпечення безпеки та надійності бази даних.

Процес нормалізації покращує організацію бази даних, полегшуючи роботу з нею усім, починаючи з простих користувачів до адміністратора, який відповідає за загальне керування об'єктами бази даних. Зменшується кількість повторювань

даних, що спрощує структуру даних та економить дисковий простір. Через скорочення дублювання даних зменшується ймовірність їх неузгоджування. Так як в підсумку нормалізації база даних поділяється на більш дрібні таблиці, модифікувати існуючі таблиці стає легше. Набагато легше змінювати таблицю з невеликою кількістю даних, ніж велику таблицю, яка містить усі важливі для бази даних значення. Нарешті, підвищується безпека у тому розумінні, що адміністратор бази даних отримує можливість дозволити різним користувачам доступ лише до обмеженого переліку таблиць. Нормалізація спрощує керування безпекою.

Недоліки нормалізації

Хоча більшість вдало працюючих баз даних в деякому ступені є нормалізованими, нормалізація має один суттєвий недолік: уповільнення роботи бази даних. Виконання запиту або транзакції передбачає використання центрального процесору комп'ютера, пам'яті та операцій введення-виведення. Тобто, в нормалізованій базі даних для виконання транзакцій або запитів найбільш інтенсивно використовується центральний процесор, вимагається більше пам'яті та більша кількість операцій введення-виведення, ніж в ненормалізованій базі даних. В нормалізованій базі даних треба знаходити відповідні таблиці та пов'язувати дані для того, щоб вилучати потрібну інформацію та обробити її.

Блок змістових модулів 3 – Структурована мова запитів SQL

Змістовий модуль 3.1 - Структурована мова запитів SQL

Тема 3.1.1 Мова SQL. Оператор Select. Запити на читання даних. Використання агрегатних функцій.

(2 години)

Мета: призначення та функціональні можливості мови SQL, засвоєння головних операторів SQL; навчитися створювати запити з використанням функцій мови SQL.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Мова SQL. Функціональні можливості SQL.
2. Оператори мови SQL
3. Інструкція оператора Select.
4. Приклади з використанням операторів мови SQL.
5. Агрегатні функції SQL.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 3.1.1.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Для чого призначена мова SQL?
2. Що таке «Запит»?
3. Якими функціональними можливостями володіє мова SQL?
4. На які категорії можна поділити команди мови SQL?
5. Для чого використовується мова маніпулювання даними? Який оператор цієї мови Ви можете навести як приклад?
6. До якої категорії команд мови SQL можна віднести оператор Create table?
7. Які головні оператори мови маніпулювання даними Ви знаєте?
8. Які оператори є обов’язковими при виконанні будь-якого запиту?
9. Які оператори є необов’язковими при виконанні будь-якого запиту?
10. Які існують головні типи умов пошуку?
11. Для чого призначене ключове слово Like?
12. Для чого призначені логічні оператори **And**, **Or** або **Not**?
13. Які функції мови SQL Ви знаєте?
14. Для чого призначена кожна функція мови SQL?
15. В яких операторах мови SQL можна використовувати функції, а в яких ні?

В цьому розділі ми познайомимося з мовою структурованих запитів **SQL** (**S**tructure **Q**uery **L**anguage).

SQL є інструментом для відбору та обробки інформації, яка міститься в базі даних. Унікальність цієї мови полягає в тому, що вона є єдиною стандартною мовою баз даних. Мову SQL підтримують більш ніж 100 СУБД, які працюють на мейнфреймах та звичайних персональних комп'ютерах. Та саме головне: не дивлячись на те, що кожна СУБД привносить свої доповнення, стандартну SQL підтримують всі працюючі бази даних.

Яким чином працює SQL? Коли користувачу необхідно отримати інформацію з бази даних, він просить цю інформацію в СУБД за допомогою SQL. СУБД опрацьовує запит, знаходить (або не знаходить) необхідні дані та повертає результат користувачу.

Запит – команда, яку надає користувач базі даних, та яка повідомляє їй щоб вона вивела визначену інформацію з таблиць в пам'ять. Ця інформація звичайно посилається безпосередньо на екран комп'ютеру або терміналу, яким користується користувач.

Функціональні можливості SQL

SQL використовується для реалізації всіх функціональних можливостей, які надаються СУБД. До них належать:

1. **Організація даних.** SQL надає користувачу можливість змінювати структуру зображення даних, а також встановлювати відношення між елементами бази даних.
2. **Обрання даних.** SQL надає користувачу або додатку можливість вилучати з бази даних інформацію, яка міститься в ній та користуватися нею.
3. **Обробка даних.** SQL надає користувачу або додатку можливість змінювати базу даних, тобто додавати до неї нові дані, а також знищувати або модифікувати існуючі дані.
4. **Керування доступом.** За допомогою SQL можна обмежити можливості користувача по обранню та зміні даних та захистити їх від несанкціонованого доступу.

5. **Сумісне використання даних.** SQL дозволяє координувати сумісне використання даних користувачами, які працюють паралельно, для того щоб вони не заважали одне одному.
6. **Цілісність даних.** SQL дозволяє забезпечити цілісність бази даних, захищаючи її від руйнування з-за неузгоджуваних змін або відмови системи.

SQL не є комп'ютерною мовою. SQL є невід'ємною частиною СУБД та працює лише з реляційними базами даних.

Оператори мови SQL

Команди мови SQL можна поділити на три категорії:

- **DDL - D**ata **D**efinition **L**anguage (мова визначення даних) – складається з команд, які створюють об'єкти (таблиці, індекси, представлення, та т. д.) в базі даних. Приклад – оператор CREATE TABLE.
- **DML - D**ata **M**anipulation **L**anguage (мова маніпулювання даними) – це набір команд, які визначають які значення зображені в таблицях в будь-яку мить часу. Приклад – SELECT.
- **DCL - D**ata **C**ontrol **L**anguage (мова керування даними) – складається з засобів, які визначають, дозволити користувачу виконувати визначені дії або ні. Приклад – оператор GRANT або REVOKE.

Головні оператори DML:

1. **Select** – обрання даних з бази даних;
2. **Insert** – додавання даних до таблиці;
3. **Update** – оновлення (зміна) даних в таблиці;
4. **Delete** – знищення даних з таблиці.

Інструкція Select

SELECT [Distinct| All] список полів

FROM ім'я таблиці або список імен таблиць [alias] [, ...]

[WHERE умова обрання або з'єднання]

[GROUP BY групування даних за полями]

[HAVING умова для групи]

[ORDER BY список полів, за якими треба підпорядкувати виведення]

Тут **список таблиць** є іменем існуючих в базі даних таблиць або представлень, до яких треба отримати доступ. Необов'язковий параметр **alias** – скорочення, яке встановлюється для імені таблиці **ім'я таблиці**.

Обробка елементів оператору **Select** здійснюється у наступній послідовності:

1. **FROM** – визначається ім'я використовуємої таблиці або декількох таблиць. FROM завжди йде слідом за SELECT, порядок таблиць немає значення.
2. **WHERE** – виконується фільтрація рядків об'єкта у відповідності з поставленими умовами.
3. **GROUP BY** – утворюються групи рядків, які мають однакові значення у вказаному стовпці.
4. **HAVING** – фільтруються групи рядків об'єкта у відповідності з поставленою умовою.
5. **SELECT** – встановлюється, які стовпці повинні бути присутніми в вихідних даних.
6. **ORDER BY** – визначається порядок разташування результатів виконання оператора.

Порядок речень та фраз в Select не можна змінити. Лише два речення оператора - **Select** та **From** - є обов'язковими, всі інші речення та фрази можуть не використовуватися.

Операція виконання оператора Select є зачиненою: результат запиту до таблиці уявляє собою іншу таблицю.

Зауваження: SQL є регістронезалежною мовою, тобто немає різниці між великими та маленькими літерами.

Для побудови прикладів SQL-операторів будуть використовуватися такі таблиці:

1. **Відділення** (Номер відділення, Місто, Вулиця, Район, Поштовий індекс, Телефон);
2. **Співробітники** (Номер співробітника, Прізвище, Ім'я, По-батькові, Адреса, Телефон, Посада, Стать, Дата народження, ЗП, Номер відділення);
3. **Об'єкт нерухомості** (Номер об'єкта, Місто, Вулиця, Район, Тип, Кімнати, Орендна плата, Номер володаря);
4. **Володар** (Номер володаря, Прізвище, Ім'я, По-батькові, Адреса, Телефон);

5. **Орендатор** (Номер орендатора, Прізвище, Ім’я, По-батькові, Адреса, Телефон, Тип, Максимальна орендна плата, Номер відділення);
6. **Огляд** (Номер огляду, Номер орендатора, Номер объекта, Дата огляду, Коментарі).

Обрання рядків

Приклад 1.Обрання всіх рядків та стовпців.

Скласти список відомостей про кожного зі співробітників.

Так як в наведеному запиті не вказано жодного обмеження, то розміщувати в оператор речення Where немає потреби. Крім того, необхідно обрати всі існуючі в таблиці стовпці.

```
Select [Номер співробітника], Прізвище, Ім’я, По-батькові, Адреса, Телефон,
Посада, Стать, [Дата народження], ЗП, [Номер відділення]
From Співробітники;
```

Так як обрання всіх існуючих в таблиці стовпців виконується часто, в SQL визначений спрощений варіант запису значення «всі стовпці» - замість імен стовпців вказується *:

```
Select *
From Співробітники;
```

Приклад 2. Обрання деяких рядків та стовпців.

Створити звіт про заробітну плату всіх співробітників з вказівкою лише їх номерів, ПІБ, а також відомостей про заробітну плату.

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

Приклад 3. Використання ключового слова Distinct

Створити список номерів всіх здаваємих в оренду об’єктів нерухомості, які були оглянуті клієнтами.

```
Select [Номер об’єкту]
From Огляд;
```

Якщо б таблиця «Огляд» містила б декілька дублюючих значень об’єкту нерухомості, то для їх знищення використовується ключове слово Distinct.

```
Select Distinct [Номер об’єкту]
From Огляд;
```

Приклад 4. Поля, які обчислюються

Створити звіт про заробітну плату за місяць всіх співробітників з вказівкою лише їх номерів, ПІБ та заробітної плати.

Select [Номер співробітника], Прізвище, Ім'я, По-батькові, ЗП /12

From Співробітники;

В таблиці, яка була отримана в підсумку виконання запиту, п'ятий стовпець має назву col5. В одних діалектах мови SQL імена таким стовпцям призначаються у відповідності з порядком їх розташування в таблиці (наприклад, col5), в інших діалектах у такого стовпця ім'я може бути зовсім відсутнім.

Стандарт ISO дозволяє явним чином задавати імена стовпців підсумкової таблиці, для чого застосовується фраза AS. При її використанні наведений трохи вище оператор Select прийме такий вигляд:

Select [Номер співробітника], Прізвище, Ім'я, По-батькові, З_П /12 AS [ЗП за місяць]

From Співробітники;

Обрання рядків з використанням оператора WHERE

В наведених вище прикладах в підсумку виконання оператора Select були обрані всі рядки таблиці. Але дуже часто треба тим чи іншим чином обмежити набір рядків, які розташовуються в підсумковій таблиці запита. Це досягається за допомогою оператора **Where**.

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** – перевіряється, потрапляє або ні результат обчислення виразу в заданий діапазон значень;
3. **належність до множини** – перевіряється, належить або ні результат обчислення виразу до заданої множини значень;

4. **відповідність шаблону** – 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;

> - більше;

<= - менше або рівно;

>= - 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, використовуємих для визначення порядку обчислення виразу.

Обчислення виразу в умовах пошуку виконується за такими правилами:

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

Приклад 5. Порівняння умов пошуку.

Перелічити весь персонал з розміром заробітної плати більш ніж 3500 грн.

```
Select [Номер співробітника], Прізвище, Ім’я, По-батькові, Посада, ЗП
From Співробітники
Where ЗП>3500;
```

Приклад 6. 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 7. Використання діапазону Between в умовах пошуку.

Перелічити весь персонал з заробітною платою від 13000 до 15000 грн.

```
Select [Номер співробітника], Прізвище, Ім'я, По-батькові, Посада, ЗП
From Співробітники
Where ЗП Between 13000 and 15000;
```

Цей запит можна записати ще таким чином:

```
Select [Номер співробітника], Прізвище, Ім'я, По-батькові, Посада, ЗП
From Співробітники
Where ЗП>=13000 and ЗП<=15000;
```

Приклад 8. 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 9. 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 з нуля або більшої кількості символів;
2. **_** - символ підкреслювання являє 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 10. 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;
```

Використання агрегатних функцій

За допомогою запитів можна узагальнити значення одного поля. Для того щоб здійснити таку операцію використовуються **агрегатні функції**. Вони повертають тільки одне значення для цілої групи рядків таблиці. Наприклад, сума значень одного поля або найбільше значення зі всіх обраних значень одного поля - все це можливо реалізувати за допомогою агрегатних функцій.

В стандартній мові SQL існує п'ять агрегатних функцій: **COUNT(), SUM(), AVG(), MAX(), MIN().**

Функції використовуються як імена полів в операторі SELECT, але з одним виключенням: імена полів застосовуються як аргументи.

Функції **SUM()** та **AVG()** можуть працювати лише з цифровими полями. Функції **COUNT(), MAX()** та **MIN()** працюють як з цифровими так й з символьними полями. При застосуванні до символьних полів функції **MAX()** та **MIN()** працюють з ACSII еквівалентами символів.

Агрегатні функції можуть використовуватися лише в списку оператора Select та в складі речення Having. У всіх інших випадках використання цих функцій є неприпустимим.

Функція **COUNT() –** це функція, яка повертає кількість значень у вказанному стовпці.

Приклад 1. Обчислити кількість співробітників в компанії.

```
Select Count([Номер співробітника]) AS [Кількість співробітників]
From Співробітники;
```

Приклад 2. Визначити кількість здаваємих в оренду об’єктів нерухомості мають ставку орендної плати менш ніж 10000 грн. на місяць:

```
Select COUNT(*) AS [Кількість об’єктів]
From [Об’єкт нерухомості]
Where [Орендна плата] < 10000;
```

COUNT() із зірочкою містить як NULL-значення, так й значення, які півторюються.

SUM() - ця функція дозволяє отримати суму значень одного або декількох полів.

Приклад 3. Визначити загальну кількість менеджерів компанії та обчислити суму їх заробітної плати.

```
Select Count ([Номер співробітника])AS Кількість, Sum(ЗП) AS [Сумма зп]
From Співробітники
Where Посада=‘Менеджер’;
```

AVG() - функція вираховує середнє арифметичне значення одного або декількох полів.

MIN() – повертає мінімальне значення у вказанному стовпці.

MAX() - повертає максимальне значення у вказанному стовпці.

Приклад 4. Обчислити значення мінімальної, максимальної та середньої заробітної плати.

Select MIN(ЗП) AS Мінімальна, MAX(ЗП) AS Максимальна, AVG(ЗП) AS Середня From Співробітники;

Примітка: в Microsoft Access існує ще 4 додаткових агрегатних функції: **StDev** – обчислює зміщуване значення середньоквадратичного відхилення, **StDevP** - обчислює незміщуване значення середньоквадратичного відхилення, **Var** – повертає значення зміщуваної дисперсії, **VarP** - повертає значення незміщуваної дисперсії.

Всі вказані функції повертають результат, який обчислюється з набору значень, які містяться у вказаному полі запиту.

Тема 3.1.2 Використання операторів Group by та Having. Робота з багатотабличними запитами.

(2 години)

Мета: навчитися створювати запити з використанням операторів мови SQL Group By та Having; знати якими способами можна отримати інформацію з декількох таблиць та вміти створювати багатотабличні запити.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Оператор Group By.
2. Оператор Having.
3. Способи з’єднання таблиць.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичними відомостями теми 3.1.2.
   2. Вивчити основні поняття лекції.

3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Який оператор мови SQL дозволяє застосовувати агрегатні функції до отриманої підмножини значень?
2. Для чого призначений оператор мови SQL Having?
3. В чому різниця між операторами Where та Having?
4. Чому оператор Having треба використовувати спільно з оператором Group By?
5. Як отримати інформацію з декількох таблиць?
6. Якими способами можна з'єднати декілька таблиць?
7. Який синтаксис має операція Inner Join?
8. В чому полягає різниця між способами з'єднання декількох таблиць?
9. Що таке «псевдонім» та для чого він призначений?

Використання операторів Group by та Having.

Робота з функціями істотньо відрізняється від обрання поля тим, що вихідні дані містять єдине значення незалежно від кількості рядків в таблиці. За цією причиною агрегатні функції та поля не можуть обиратися одночасно.

Для реалізації такої можливості існує оператор **GROUP BY**, який визначає підмножину значень окремого поля в термінах іншого поля та дозволяє застосовувати агрегатні функції до отриманої підмножини. Це надає можливість комбінувати поля та агрегатні функції в одному операторі Select. **GROUP BY** використовує агрегатні функції окремо до кожної серії груп, які визначаються загальним значенням поля. Це означає, що поле, до якого застосовується GROUP BY по визначенню має на виході лише одне значення на кожну з груп, що відповідає застосуванню агрегатних функцій. Таке сполучання результатів дозволяє комбінувати агрегати з полями вказаним способом.

Всі імена стовпців, які наведені в операторі Select, повинні бути присутніми в операторі GROUP BY – за виключенням тих випадків, коли ім'я стовпця використовується в агрегатній функції.

Стандартом ISO визначено, що при проведенні групування всі відсутні значення розглядаються як рівні. Якщо два рядка таблиці в одному й тому ж стовпці,

який групується, містять значення Null та ідентичні значення у всіх інших непустих стовпцях, які групуються, вони розміщуються в одну й ту ж групу.

Приклад 1. Використання оператора GROUP BY

Визначити кількість персоналу, який працює в кожному з відділень компанії, а також їх загальну заробітну плату. Результат розташувати в порядку збільшення номера відділення компанії.

```
Select [Номер відділення], Count([Номер співробітника]) AS Кількість, Sum(ЗП) AS [Загальна ЗП]
From Співробітники
Group By [Номер відділення]
Order By [Номер відділення];
```

Немає потреби розміщувати імена стовпців «Номер співробітника» та «ЗП» у список оператора Group By, так як в списку оператора Select вони використовуються лише в агрегатних функціях. В той самий час стовпець «Номер відділення» в списку Select не пов'язаний з жодною агрегатною функцією та за цією причиною обов'язково повинен бути вказаний в операторі Group By.

Номер відділення	Кількість	Загальна ЗП
3	3	40050
5	2	27250
7	1	14300

При обробці цього запиту будуть виконані такі дії:

1. рядки таблиці «Співробітники» розподіляються в групи у відповідності зі значеннями в стовпці «Номер відділення». В межах кожної з груп виявляються дані про весь персонал одного з відділень компанії. Тобто будуть створені три групи.

Номер відділення	Номер співробітника	ЗП
3	37	12400
3	14	14500
3	5	13150
5	21	13150
5	41	14100
7	9	14300

Номер Відділення	Count (Номер співробітника)	Sum(З_П)
3	3	40050
5	2	27250
7	1	14300

2. Для кожної з груп обчислюється загальна кількість рядків, яка дорівнює кількості співробітників відділення, а також сума значень в стовпці «ЗП», яка є сумою заробітної плати всіх співробітників відділення компанії, яка нас цікавить. Потім генерується єдиний підсумковий рядок для всієї групи початкових рядків.
3. Наприкінці, отримані рядки підсумкової таблиці пересортовуються в порядку збільшення номера відділення, який вказаний в стовпці «Номер відділення».

Оператор Having призначений для використання спільно з оператором Group By для завдання обмежень, які вказані з метою відбору тих груп, які будуть розміщені в підсумковій таблиці запиту.

Хоча оператори Having та Where мають схожий синтаксис, їх призначення відрізняються. Оператор Where призначений для фільтрації окремих рядків, які використовуються для групування або які розміщуються в підсумковій таблиці запита, тоді як оператор Having використовується для фільтрації груп, які розміщуються в підсумковій таблиці запита.

Стандарт ISO вимагає, щоб імена стовпців, які використовуються в операторі Having, обов’язково були присутніми в списку оператора Group By або використовувались в агрегатних функціях.

На практиці умови пошуку в операторі Having завжди містять хоча б одну агрегатну функцію, в протилежному випадку ці умови пошуку повинні бути розміщені в операторі Where та використовуватися для відбору окремих рядків.

Треба пам’ятати, що агрегатні функції не можуть використовуватися в операторі Where. Оператор Having не є необхідною частиною SQL – будь-який

запит, написаний з використанням оператору Having, може бути зображений в іншому вигляді без його використання.

Приклад 2. Використання оператора Having

Для кожного відділення компанії з кількістю персоналу більш ніж 1 особа визначити кількість співробітників та суму їх заробітної плати.

```
Select [Номер відділення], Count([Номер співробітника]) AS Кількість, Sum(ЗП) AS [Загальна ЗП]
From Співробітники
Group By [Номер відділення]
Having Count([Номер співробітника])>1
Order By [Номер відділення];
```

Цей приклад є аналогом попереднього, але тут використовуються додаткові обмеження, які вказують на те, що нас цікавлять відомості лише про ті відділення компанії, в яких працює більш ніж 1 особа. Така вимога накладається на групи, тому в запиті треба використовувати оператор Having.

Номер відділення	Кількість	Загальна ЗП
3	3	40050
5	2	27250

Робота з багатотабличними запитами

Користувачу звичайно треба вилучати як можна більш повну інформацію з бази даних, а в реляційних базах даних для цього іноді треба звертатися до декількох таблиць одночасно. Для цього треба вказати які стовпці та з якої таблиці треба обрати. В SQL синтаксис звертання до стовпців таблиці зображується таким чином: **Ім’я таблиці.Ім’я стовпця** (якщо стовпці мають унікальні назви в межах запиту, то ім’я таблиці необов’язково вказувати).

Припустимо, що нам треба виконати такий запит: скласти список імен всіх клієнтів, які вже оглянули хоча б один здаваємий в оренду об’єкт нерухомості та повідомили свою думку.

```
Select Орендатор.[Номер орендатора], Орендатор.Прізвище, Орендатор.Ім’я,
Орендатор.По-батькові, Огляд.[Номер об’єкта], Огляд.Коментар
From Орендатор, Огляд
```

Where Орендатор.[Номер орендатора]=Огляд.[Номер орендатора];

В цьому запиті треба показати відомості як з таблиці **Орендатор**, так й з таблиці **Огляд**, тому при побудові запиту ми скористувалися механізмом **простого з'єднання таблиць**. В Select перелічуються всі стовпці, які повинні бути розміщені в підсумковій таблиці запиту.

Висновок. Якщо треба отримати інформацію з більш ніж однієї таблиці, то можна або застосувати підзапит, або виконати з'єднання таблиць. Для виконання з'єднання є достатнім в операторі From вказати імена двох або більшої кількості таблиць, роз'єднавши їх комами, після чого включити в запит оператор Where з визначенням стовпців, які використовуються для з'єднання вказаних таблиць.

Реалізувати з'єднання таблиць можна за допомогою операції ***Inner Join*** та підзапиту.

Inner Join поєднує записи з двох таблиць, якщо сполучні поля цих таблиць містять однакові значення.

Синтаксис операції Inner Join:

From таблиця1 **Inner Join** таблиця2 **On** таблиця1.поле1 **оператор** таблиця2.поле2

Аргументи операції Inner Join:

- ✓ **таблиця1, таблиця2** – імена таблиць, записи яких підлягають об'єднанню;
- ✓ **поле1, поле2** – імена полів, які об'єднуються. Якщо ці поля не числові, то вони повинні мати однаковий тип даних;
- ✓ **оператор** – будь-який оператор порівняння.

Операцію **Inner Join** можна використовувати в будь-якому операторі From. Вона створить еквіз'єднання, яке також відоме як внутрішнє з'єднання.

Приклад 3. Використання декількох способів з'єднання таблиць.

Скласти список персоналу, який працює в відділенні компанії, розташованому на вулиці Ген. Петрова в будинку 1.

<u>1-й спосіб: використання підзапиту</u>

```
Select [Номер співробітника], Прізвище, Ім'я, По-батькові, Посада
From Співробітники
Where [Номер відділення] =
   (Select [Номер відділення]
    From Відділення
```

Where Вулиця = «Ген. Петрова 1»);

2-й спосіб: використання простого з'єднання таблиць

Select Співробітники.Прізвище, Співробітники.Ім'я, Співробітники.По-батькові, Співробітники.Посада

From Співробітники, Відділення

Where Співробітники.[Номер відділення] = Відділення.[Номер відділення] and Відділення.Вулиця=«Ген. Петрова 1»;

3-й спосіб: використання операції Inner Join для з'єднання таблиць

Select Співробітники.Прізвище, Співробітники.Ім'я, Співробітники.По-батькові, Співробітники.Посада

From Відділення Inner Join Співробітники On Співробітники. [Номер відділення] = Відділення.[Номер відділення]

Where Відділення.Вулиця = «Ген. Петрова 1»;

Треба відмітити, що хоча назви таблиць перед іменами стовпців достатньо добре інформують користувача, постійно переписувати їх назви набридає. Для полегшання життя в SQL існує поняття **аліясів** (Alias) - псевдонімів імен таблиць. Псевдоніми використовуються для візуального спрощення тексту запитів. У випадку їх використання імена таблиць та псевдоніми, які їм призначаються, повинні розподілятися пробілами. Якщо для таблиці визначений псевдонім, він може бути використаний в будь-якому місці, де необхідне вказання імені цієї таблиці.

Для прикладу псевдонімів скористуємося другим способом (використання простого з'єднання таблиць) попереднього прикладу.

Приклад 4. Використання псевдонімів.

Select С.Прізвище, С.Ім'я, С.По-батькові, С.Посада

From Співробітники С, Відділення О

Where С. [Номер відділення] = О.[Номер відділення] and О.Вулиця = «Ген. Петрова 1»;

Тема 3.1.3 Зміна змісту бази даних. Оператори Insert, Update та Delete.

(2 години)

Мета: знати призначення та синтаксис операторів мови SQL, за допомогою яких можна додавати, знищувати та оновлювати дані в базі даних.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Оператор нових даних до таблиці - Insert.
2. Оператор модифікації даних в таблиці - Update.
3. Оператор знищення даних з таблиці - Delete.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:
   1. Ознайомитись з теоретичними відомостями теми 3.1.3.
   2. Вивчити основні поняття лекції.
   3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Які оператори SQL призначені для модифікації змісту бази даних?
2. Який формат має оператор додавання даних до таблиці?
3. Яким чином можна додати одразу групу записів до таблиці?
4. Можна оновити не всі, а лише деякі записи таблиці? Як це можна зробити?
5. За допомогою якого оператора можна однією дією знищити всі дані з таблиці?

SQL є повнофункціональною мовою маніпулювання даними, яка може використовуватися не лише для обрання даних з бази, але й для модифікації її змісту. Існує 3 оператори SQL, які призначені для модифікації змісту бази даних: Insert, Update та Delete.

Додавання нових даних до таблиці - оператор INSERT

Для додавання єдиного рядка у вказану таблицю використовується наступний формат оператору Insert:

Insert Into Ім’я таблиці [(Список полів таблиці)] Values (Список значень полів таблиці);

1. Параметр **«Ім’я_таблиці»** - ім’я таблиці, до якої додаються дані.
2. **«Список полів таблиці»** являє собою список, який складається з імен одного або більшої кількості стовпців, розподілених комами. Параметр **«Список полів таблиці»** є необов’язковим. Якщо він відсутній, то передбачається використання списку з імен всіх стовпців таблиці, вказаних в тому порядку, в якому вони були описані в операторі Create table. Якщо в операторі **Insert** вказується конкретний список імен стовпців, то будь-які відсутні в ньому стовпці повині бути описані при створенні таблиці як припускаючі значення Null – за виключенням випадків, коли при описі стовпця використовується параметр Default.
3. Параметр **«Список значень полів таблиці»** повинен наступним чином відповідати параметру **«Список полів таблиці»:**
    - кількість елементів в обох списках повинна бути однаковою;
    - повинне існувати пряме співвідношення між позицією одного й того ж елементу в обох списках, тому перший елемент списка **«Список значень полів таблиці»** відповідає першому елементу списка **«Список полів таблиці»**, другий елемент списка **«Список значень полів таблиці»** до другого елементу списка **«Список полів таблиці»** та т. д.;
    - типи даних елементів списка **«Список значень полів таблиці»** повинні бути сумісні з типом даних відповідних стовпців таблиці.

Приклад. Використання конструкції Insert ... values.

Додати до таблиці «Співробітники» новий запис, який містить дані у всіх стовпцях.

Insert Into Співробітники values (50,‘Савченко’,‘Сергій’, ‘Олександрович’, ‘Одеса, Архітекторська 35,24’ 481534, ‘Заступник директора’, ‘Ч’, #12/03/1973#, ‘23450’, 3);

Додавання групи записів до таблиці

Інструкцію Insert Into синтаксису запита для додавання декількох записів можна також використовувати з Select ... From для копіювання набору записів іншої таблиці або запиту. В цьому випадку Select визначає поля, які треба додати до специфікованої таблиці-адресату (тобто таблиці, яка змінюється).

Формат цієї операції має такий вигляд:

Insert Into Таблиця, яка змінюється Select (Список полів таблиці)

From Початкова таблиця;

Дані обираються з початкової таблиці та розміщуються в таблицю, в якій відбуваються зміни.

В наступному прикладі з пропонуємої таблиці **«Студенти»** обираються всі студенти, які поступили на роботу в період 2021 р, та їх записи додаються до таблиці «Співробітники».

Insert Into Співробітники Select Студенти.*

From Студенти

Where [Дата прийому] Between #01/01/2011# and #12/31/2011#;

Модифікація даних в базі - оператор UPDATE

Ця інструкція дозволяє змінити зміст вже існуючих рядків вказаної таблиці. Цей оператор має такий формат:

Update Ім’я таблиці

Set Ім’я стовпця1 = Значення стовпця1 [, Ім’я стовпця2 = Значення стовпця2 …]

[Where Умова пошуку];

В інструкції **Set** вказуються імена одного або більшої кількості стовпців, дані в яких треба змінити. Можна оновлювати одразу декілька стовпців, перелічуючи їх через кому.

Параметри **Значення стовпця** являють собою нові значення відповідних стовпців та повинні бути сумісними з ними за типом даних.

Оператор **Where** є необов’язковим. Якщо він відсутній, значення вказаних стовпців будуть змінені у всіх рядках таблиці. Якщо оператор **Where** присутній, то будуть оновлені лише ті рядки, які задовільняють умові пошуку, вказаній в параметрі **Умова пошуку.**

Приклад. Оновлення всіх рядків таблиці.

Всьому персоналу компанії підвищити заробітну плату на 3%.

Update Співробітники

Set ЗП = ЗП*1.03;

Приклад. Оновлення деяких рядків таблиці.

Всім менеджерам компанії підвищити заробітну плату на 5%.

```
Update Співробітники
Set ЗП = ЗП*1.05
Where Посада = «Менеджер»;
```

Приклад. Оновлення деяких стовпців таблиці.

Перевести Філатова Андрія Петровича - співробітника з номером 21 на посаду заступника директора та підвищити йому заробітну плату до 13450 грн.

```
Update Співробітники
Set Посада = «Заступник директора», ЗП = 13450
Where [Номер співробітника] = 21;
```

Знищення даних з бази - оператор DELETE

Delete дозволяє знищувати рядки даних з таблиці. Цей оператор має такий формат:

Delete From Ім'я_таблиці
[Where Умова пошуку];

Приклад. Знищення визначених рядків таблиці.

Знищити всі записи про огляди здаваємого в оренду об'єкта нерухомості з номером 4.

```
Delete From Огляд
Where [Номер об'єкта]= 4;
```

Приклад. Знищення всіх рядків таблиці.

Знищити всі рядки з таблиці «Огляд»

```
Delete From Огляд;
```

Тема 3.1.4 Визначення структури даних в SQL

(2 години)

Мета: знати призначення та синтаксис операторів для створення таблиць, доменів та індексів. Навчитися здійснювати зміну структури таблиці за допомогою оператора Alter table та знищення об'єктів бази даних.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;

b. Психоемоційний настрій;

c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Оператор *Create table* та його синтаксис.

2. Оператор *Drop table* та його синтаксис.

3. Оператор *Alter table* та його синтаксис..

4. Визначення доменів.

5. Створення індексів.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 3.1.4.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що в реляційній БД вважається найважливішим елементом її структури?

2. Для чого призначений оператор CREATE TABLE?

3. Як визначається первинний ключ при створенні таблиці оператором CREATE TABLE?

4. Як визначається зовнішній ключ при створенні таблиці оператором CREATE TABLE?

5. В яких випадках при визначенні стовпця вказується NOT NULL?

6. В якому випадку виконання оператору DROP TABLE завершиться невдачею?

7. Для чого призначений параметр CASCADE при роботі з оператором DROP TABLE?

8. Для чого використовується оператор ALTER TABLE та що можна зробити з його допомогою?

9. Що означає поняття «домен»?

10. За допомогою якого оператора створюються домени?

11. Яке призначення в базах даних мають індекси та який синтаксис має оператор CREATE INDEX?

В реляційній БД найважливішим елементом її структури є таблиця. В багатокористувацькій БД основні таблиці, як правило, створює адміністратор, а користувачі просто звертаються до них в процесі роботи. При роботі з БД часто здається зручним створювати власні таблиці та зберігати в них власні дані, а також дані, прочитані з інших таблиць. Такі таблиці можуть бути тимчасовими та існувати лише впродовж одного інтерактивного SQL-сеансу, але можуть зберігатися й впродовж декількох тижнів або навіть місяців.

Створення таблиці (оператор CREATE TABLE)

Оператор CREATE TABLE визначає нову таблицю. Різноманітні речення оператору задають елементи визначень таблиці.

Після виконання оператору CREATE TABLE з'являється нова таблиця, якій привласнюється ім'я, вказане в операторі. Ім'я таблиці повинне бути ідентифікатором, припустимим в SQL, та не повинне конфліктувати з іменами існуючих таблиць.

CREATE TABLE ім'я_таблиці
(*визначення стовпців*
ім'я стовпця тип даних [DEFAULT значення] [NOT NULL],…
визначення первинного ключа
PRIMARY KEY(ім'я стовпця,…),
визначення зовнішніх ключів
FOREIGN KEY ім'я обмеження (ім'я стовпця,…)
REFERENCE ім'я таблиці
[ON DELETE CASCADE | SET NULL | SET DEFAULT | NO ACTION]
[ON UPDATE CASCADE | SET NULL | SET DEFAULT | NO ACTION],
умова унікальності даних
UNIQUE(ім'я стовпця,…),
умова перевірки
CHECK (умова пошуку))

Визначення стовпців. Стовпці нової таблиці задаються в операторі CREATE TABLE. Визначення стовпців представляють собою розміщений в дужках список, елементи якого відокремлені одне від одного комами. Послідовність визначень стовпців в списку відповідає розташуванню стовпців в таблиці. Кожне таке визначення містить таку інформацію:

1. **Ім'я стовпця**, яке використовується для посилання на стовпець в операторі SQL. Кожен стовпець в таблиці повинен мати унікальне ім'я, але в різних таблицях імена стовпців можуть співпадати.

2. **Тип даних стовпця**, який показує, дані якого виду зберігаються в стовпці.

3. **Вказання на те, обов'язково або ні стовпець повинен містити дані**. Речення NOT NULL попереджає занесення в стовпець значень NULL, в іншому випадку значення NULL припускаються.

4. **Значення за умовчуванням** для стовпця, яке заноситься в стовпець в тому випадку, якщо в операторі INSERT для таблиці не визначене значення даного стовпця.

5. **Визначення первинного та зовнішнього ключів.** Окрім визначень стовпців таблиці, в операторі CREATE TABLE вказується інформація про первинний ключ таблиці та про зв'язки таблиці з іншими таблицями БД. Ця інформація міститься в реченях PRIMARY KEY та FOREIGN KEY.

В реченні PRIMARY KEY задається стовпець або стовпці, які створюють первинний ключ таблиці. Цей стовпець (або комбінація стовпців) служить в якості унікального ідентифікатора рядків таблиці. СУБД автоматично стежить за тим, щоб первинний ключ кожного рядка таблиці мав унікальне значення. Окрім того, в визначеннях стовпців первинного ключа повинне бути вказане, що вони не можуть містити значення NULL.

В реченні FOREIGN KEY задається зовнішній ключ таблиці та визначається зв'язок, який він створює для неї з іншою таблицею (таблицею-предком). В ньому вказуються:

1. стовпець або стовпці таблиці, яка створюється, які створюють зовнішній ключ;

2. таблиця, зв'язок з якою створює зовнішній ключ. Це таблиця-предок;

3. необов’язкове ім’я для цього відношення; воно не використовується в операторах SQL, але може з’являтися в повідомленнях про помилки та треба буде в подальшому, якщо буде необхідно знищити зовнішній ключ;

4. як СУБД повинна вести себе зі значеннями NULL в одному або декількох стовпцях зовнішнього ключа при пов’язуванні його з рядками таблиці-предка;

5. необов’язкове правило знищення для данного відношення (CASCADE, SET NULL, SET DEFAULT або NO ACTION), яке визначає дію, яку треба виконати при знищенні рядка-предка;

6. необов’язкове правило оновлення для данного відношення, яке визначає дію, яку треба виконати при оновленні первинного ключа в рядку-предку;

7. необов’язкова умова перевірки, яка обмежує дані в таблиці так, щоб вони відповідали визначеній умові пошуку.

Наприклад, створити таблиці City, Groups та Students.

Таблиця City містить 2 стовпці: CityNo (Номер міста), CityName (Назва міста).

Таблиця Groups містить 2 стовпці: GrNo (Номер групи), GrName (Назва групи).

Таблиця Students містить такі стовпці: StNo (Номер студента), StName (ПІБ студента), CityNo (Номер міста, в якому мешкає студент), GrNo (Номер групи, в якій навчається студент).

Create table City (CityNo integer not null, CityName varchar(30), Primary Key (CityNo));

Create table Groups (GrNo integer not null, GrName varchar(30), Primary Key (GrNo));

```
CREATE TABLE Students
(StNo INT NOT NULL,
GrNo INT NOT NULL,
StName CHAR(30) NOT NULL,
CityNo INT,
PRIMARY KEY (StNo),
FOREIGN KEY Students_Groups(GrNo)
REFERENCES Groups,
FOREIGN KEY Students_Cities(CityNo)
REFERENCES City)
```

2-й варіант

```
CREATE TABLE Students(
StNo INT NOT NULL,
GrNo INT NOT NULL,
StName CHAR(30) NOT NULL,
CityNo INT,
Students_Groups computed by
((select a.NameGroup from groups where groups.GrNo=Students.GrNo)),
Students_Cities computed by
((select a.NameCity from City where City.CityNo=Students.CityNo)),
PRIMARY KEY (StNo))
```

Знищення таблиці (оператор DROP TABLE)

Таблиці можна знищувати з БД за допомогою оператора DROP TABLE.

DROP TABLE ім’я таблиці CASCADE | RESTRICT

Оператор містить ім’я таблиці, яка знищується.

Стандарт SQL2 вимагає, щоб оператор DROP TABLE містив в собі або параметр CASCADE, або RESTRICT, які визначають, як впливає знищення таблиці на інші об’єкти БД. Якщо заданий параметр RESTRICT та в БД є об’єкти, які містять ссилку на таблицю, яка знищується, то виконання оператора DROP TABLE завершиться невдачею. В більшості комерційних СУБД припускається застосування оператора DROP TABLE без будь-яких параметрів.

Зміна визначення таблиці (оператор ALTER TABLE)

В процесі роботи з таблицею іноді виникає необхідність додати до таблиці деяку інформацію. Для зміни структури таблиці використовується оператор ALTER TABLE, за допомогою якого можна:

1. додати до таблиці визначення стовпця;

2. змінити значення за умовчуванням для будь-якого стовпця;

3. додати або знищити первинний ключ таблиці;

4. додати або знищити новий зовнішній ключ таблиці;

5. додати або знищити умову унікальності;

6. додати або знищити умову перевірки.

ALTER TABLE ім’я таблиці

ADD визначення стовпця

ALTER ім’я стовпця SET DEFAULT значення | DROP DEFAULT

DROP ім’я стовпця CASCADE | RESTRICT

ADD визначення первинного ключа

ADD визначення зовнішнього ключа

ADD умова унікальності даних

ADD умова перевірки

DROP CONSTRAINT ім’я обмеження CASCADE | RESTRICT

Приклади:

1. alter table City add kolvo varchar(4);
2. alter table City alter column kolvo to kolvo_people;
3. alter table City drop kolvo_people;

Визначення доменів

Для змісту окремого стовпця таблиці в реляційній БД існує обмеження: значення стовпця у всіх рядках таблиці повинні бути одного й того ж типу. Наприклад, всі назви міст в стовпці CityName таблиці City є символьними рядками змінної довжини.

В стандарті SQL2 формально визначення домену реалізоване як частина визначення БД. Згідно цьому стандарту, домен є поіменованою сукупністю значень даних та широко використовується в визначенні БД як додатковий тип даних. Домен створюється за допомогою оператора CREATE DOMAIN на основі базових типів

даних. Після створення домену на нього можна буде ссилатися всередині визначення таблиці як на тип даних.

Індекси (оператори CREATE/DROP INDEX)

Одним із структурних елементів фізичної пам'яті, присутніх в більшості реляційних СУБД, є ***індекс*** - засіб, який забезпечує швидкий доступ до рядків таблиці на основі значення одного або декількох стовпців.

Для створення та знищення індексів застосовуються оператори CREATE INDEX та DROP INDEX, синтаксис яких описаний нижче.

Треба відмітити, що СКБД автоматично створює індекси для первинних ключів.

CREATE [UNIQUE] INDEX ім'я індексу ON ім'я таблиці (ім'я стовпця [ASC | DESC],…)

DROP INDEX ім'я индекса

Тема 3.1 5 Представлення та привілеї в SQL

(2 години)

Мета: засвоїти поняття «представлення», його призначення, переваги та недоліки; знати призначення та синтаксис оператора для надання користувачам привілеїв – GRANT, а також призначення та синтаксис оператора для відміни привілеїв – REVOKE.

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Представлення. Оператор CREATE VIEW.
2. Знищення представлень.
3. Переваги, недоліки та оновлення представлень.
4. Оператор *GRANT* та його синтаксис.
5. Оператор *REVOKE* та його синтаксис.

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

1. Ознайомитись з теоретичними відомостями теми 3.1.5.
2. Вивчити основні поняття лекції.
3. Дати відповіді на контрольні питання.

Контрольні питання:

1. Що розуміється під поняттям «представлення»?

2. Для чого призначений оператор CREATE VIEW?

3. Що означає термін «матеріалізація представлення»?

4. За допомогою якого оператора здійснюється знищення представлення?

5. Якими перевагами володіють представлення?

6. Якими істотніми недоліками володіють представлення?

7. В яких випадках можна оновлювати представлення?

8. Завдяки чому можна розподілити базу даних на частини таким чином, щоб деяка інформація була прихована від користувачів, які не мають прав для доступу до неї?

9. В якому випадку треба використовувати директиву WITH GRANT OPTION при роботі з оператором GRANT?

10. За допомогою якого оператору можна відмінити повноваження користувачів?

11. Яке призначення мають параметри RESTRICT та CASCADE при роботі з оператором REVOKE?

Представлення в SQL

Представлення - це динамічно сформований результат однієї або декількох реляційних операцій, виконаних над відношеннями бази даних з метою отримання нового відношення.

Представлення є віртуальним відношенням, яке не завжди реально існує в базі даних, але створюється за запитом визначеного користувача в ході виконання цього запиту.

З точки зору користувача БД представлення виглядає як реальна таблиця даних, яка містить набір поіменованих стовпців та рядків даних. Але на відміну від реальних таблиць представлення не завжди існують в базі як деякий набір

зберігаємих значень даних. В дійсності, доступні через представлення рядки та стовпці даних є результатом виконання запиту, який був заданий при визначенні представлення.

СУБД зберігає визначення представлення в базі даних. Знайшовши ссилку на представлення, СУБД застосовує один з двох наступних підходів для формування представлення. При першому підході СУБД знаходить визначення представлення та перетворює ісходний запит, який лежить в основі представлення, в еквівалентний запит до таблиць, який використовується в визначенні представлення, після чого модифікований запит виконується. При другому підході, який називається *матеріалізацією представлення*, готове представлення зберігається в базі даних у вигляді тимчасової таблиці, а його актуальність постійно підтримується по мірі оновлення всіх таблиць, які лежать в його основі.

Оператор CREATE VIEW використовується для створення представлення. В ньому вказуються ім’я представлення та запит, який лежить в його основі. Для вдалого створення представлення необхідно мати дозвіл на доступ до всіх таблиць, які входять в запит.

CREATE VIEW ім’я_представлення (ім’я_стовпця,…) AS запит

При необхідності в операторі CREATE VIEW можна задавати ім’я для кожного стовпця представлення, яке створюється. Якщо вказується список імен стовпців, то він повинен містити стільки елементів, скільки стовпців міститься в запиті. Зверніть увагу на те, що задаються лише імена. Наприклад, створити представлення, яке містить прізвища студентів группы CS-091.

```
CREATE VIEW GroupCS091 (Name) AS
SELECT StName
FROM Students INNER JOIN Groups ON Students.GrNo = Groups.GrNo
WHERE Groups.GrName = 'CS-091'
```

Знищення представлення (оператор DROP VIEW)

Для знищення представлення використовується оператор DROP VIEW. В ньому також деталізовані правила знищення уявлень, на основі яких були створені інші уявлення.

DROP VIEW ім’я_представлення CASCADE | RESTRICT

Переваги представлень

В БД представлення застосовуються для зручності та дозволяють спрощувати запити до бази даних.

Головні переваги представлення перелічені нижче.

1. ***Безпека***. Кожному користувачу можна дозволити доступ до невеликої кількості представлень, які містять лише ту інформацію, яку йому дозволено знати. Таким чином, можна здійснити обмеження доступу користувачів до зберігаємої інформації.
2. ***Простота запитів***. За допомогою представлень можна прочитати дані з декількох таблиць та представити їх як одну таблицю, перетворюючи тим самим запит до багатьох таблиць в однотабличний запит до представлення.
3. ***Простота структури***. За допомогою представлень для кожного користувача можна створити власну «структуру» БД, визначивши її як множину припустимих користувачеві віртуальних таблиць.
4. ***Захист від змін***. Представлення може повертати несуперечливий та незмінний образ структури БД, навіть якщо початкові таблиці розподіляються, реструктуризуються або переіменовиваються.
5. ***Цілісність даних***. Якщо доступ до даних або введення даних здійснюється за допомогою представлень, СУБД може автоматично перевіряти, виконуються або ні визначені умови цілісності.

Недоліки представлень

Представлення володіють двома істотніми недоліками:

1. **Продуктивність.** Представлення створює лише видимість існування відповідної таблиці, та СУБД доводиться перетворювати запит до представлення в запит до початкових таблиць. Якщо представлення відображає багатотабличний запит, то простий запит до представлення становиться складним об’єднанням та на його виконання може знадобитися багато часу.

2. **Обмеження на оновлення**. Коли користувач намагається оновити рядки представлення, СУБД повинна встановити їх відповідність рядкам початкових таблиць, а також оновити останні. Це можливе лише для простих представлень; складні представлення оновлювати неможна, вони припустимі лише для читання.

Оновлення представлень

Представлення можна оновлювати в тому випадку, якщо запит, який його визначає, відповідає наступним вимогам:

1. Повинен бути відсутнім предикат DISTINCT.
2. В FROM повинна бути задана лише одна таблиця, яку можна оновлювати; тобто в представлення повинна бути одна початкова таблиця, а користувач повинен мати відповідні права доступу до неї. Якщо початкова таблиця сама є представленням, то воно також повинне задовільняти цим умовам.
3. Кожне ім'я в списку стовпців, які повертаються, повинне бути ссилкою на простий стовпець; в цьому списку не повинні міститися вирази, стовпці, які обчислюються або агрегатні функції.
4. WHERE не повинне містити вкладений запит; в ньому можуть бути присутніми лише прості умови пошуку.
5. В запиті не повинні міститися оператори GROUP BY або HAVING.

Привілеї в SQL

Механізм представлень мови SQL дозволяє різними способами розподілити базу даних на частини таким чином, щоб деяка інформація була прихована від користувачів, які не мають прав для доступу до неї. Цей режим задається за допомогою параметрів операцій, на основі яких **санкціоновані** користувачі виконують ті або інші дії з заданою частиною даних. Ця функція виконується за допомогою директиви ***GRANT***.

Тому, хто створює будь-який об'єкт, автоматично надаються всі привілеї в відношенні цього об'єкта.

Стандарт SQL1 визначає наступні привілеї для таблиць:

1. **SELECT** – дозволяє зчитувати дані з таблиці або представлення;
2. **INSERT** – дозволяє додавати нові записи до таблиці або представлення;
3. **UPDATE** – дозволяє модифікувати записи в таблиці або представленні;
4. **DELETE** – дозволяє знищувати записи з таблиці або представлення.

Стандарт SQL2 розширив список привілеїв для таблиць та представлень:

1. **INSERT** для окремих стовпців;
2. **REFERENCES** – для підтримки зовнішнього ключа.

Також додана привілея **USAGE** – для інших об'єктів БД.

Окрім того, більшість комерційних СУБД підтримує додаткові привілеї, наприклад:

1. **ALTER** – дозволяє модифікувати структуру таблиць (DB2, Oracle);
2. **EXECUTE** – дозволяє виконувати зберігаємі процедури.

Той, хто створює об'єкт, також отримує право надати привілеї доступу будь-якому іншому користувачеві за допомогою оператора **GRANT**.

GRANT {SELECT|INSERT|DELETE|(UPDATE стовпець, ...)}, ...

ON таблиця TO {користувач | PUBLIC} [WITH GRANT OPTION]

Привілеї вставки (INSERT) та оновлення (UPDATE) можуть задаватися для спеціально заданих стовпців.

Якщо задана директива WITH GRANT OPTION, це значить, що вказані користувачі наділені особливими повноваженями для заданного об'єкта – **правом надання повноважень.** Це, в свою чергу, означає, що для роботи з даним об'єктом вони можуть наділяти повноваженнями інших користувачів.

Наприклад: надати користувачу з прізвищем Ivanov повноваження для здійснення відбору та модифікації прізвищ в таблиці Students з правом надання повноважень.

GRANT SELECT, UPDATE StName

ON Students TO Ivanov WITH GRANT OPTION

Якщо користувач ***A*** **наділяє** деякими повноваженнями іншого користувача ***B***, то в подальшому він може **відмінити** ці повноваження для користувача ***B***. Відміна повноважень виконується за допомогою директиви **REVOKE** з наведеним нижче синтаксисом:

REVOKE {{SELECT | INSERT | DELETE | UPDATE},...|ALL PRIVILEGES}

ON таблиця,... FROM {користувач | PUBLIC},... {CASCADE | RESTRICT}

Оскільки користувач, з якого знімається привілея, міг надавати її іншому користувачу (якщо володіє правом надання повноважень), можливе виникнення ситуації залишених привілеїв. Головне призначення параметрів RESTRICT та CASCADE міститься в запобіганні ситуацій з виникненням залишених привілегій. Завдяки завданню параметра RESTRICT не дозволяється виконувати операцію відміни привілеї, якщо вона призводить до появи залишеної привілегії. Параметр CASCADE вказує на послідовну відміну всіх привілей, проізводних від даної.

Наприклад: зняти з користувача з прізвищем Ivanov повноваження для здійснення модифікації прізвищ в таблиці Students. Також зняти цю привилегію зі всіх користувачів, яким вона була надана Івановим.

```
REVOKE UPDATE
ON Students FROM Ivanov CASCADE
```

При знищенні домена, таблиці, стовпця або представлення автоматично будуть знищені також й всі привілеї в відношенні цих об’єктів з боку всіх користувачів.

Створюючи представлення та надаючи користувачам дозвіл на доступ до них, а не до початкової таблиці, можна тим самим обмежити доступ користувачу, дозволивши його лише до заданних стовпців або записів. Таким чином, представлення дозволяють здійснити повний контроль над тим, які дані доступні тому або іншому користувачеві.

Приклад. Створити представлення для доступу до даних групи CS-091.

```
CREATE VIEW GroupCS091 AS
SELECT *
FROM Students INNER JOIN Groups ON Students.GrNo = Groups.GrNo
WHERE Groups.GrName = «CS-091»
```

Приклад. Надати повноваження користувачу з прізвищем Ivanov.

```
GRANT SELECT, INSERT, DELETE, UPDATE
ON GroupCS091 TO Ivanov
```

Зараз користувач Ivanov може продивлятися та модифікувати лише дані групи CS-091.

Блок змістових модулів №4

СУБД в архітектурі «Клієнт - сервер»

Змістовий модуль 4.1 - Архітектура «Клієнт - сервер»

Тема 4.1 1 Архітектура «Клієнт-сервер». Моделі «Клієнт-сервер»

(1 година)

Мета: Знати про архітектуру «Клієнт - сервер» та про моделі архітектури «Клієнт - сервер».

Структура заняття:

I. Організаційний момент:
   a. Готовність групи до заняття;
   b. Психоемоційний настрій;
   c. Перевірка присутніх.

II. Повідомлення теми та мети заняття;

III. Виклад нового матеріалу:

План

1. Архітектура «Клієнт - сервер».
2. Моделі архітектури «Клієнт - сервер».

IV. Узагальнення та систематизація знань;

V. Підведення підсумків занять;

VI. Домашнє завдання:

4. Ознайомитись з теоретичними відомостями теми 4.1.1.
5. Вивчити основні поняття лекції.
6. Дати відповіді на контрольні питання.

Контрольні питання:

1. Перелічіть основні функції стандартного інтерактивного додатка, пов'язаного з обробкою баз даних.
2. Перелічіть основні завдання презентаційної логіки.
3. Що містить у собі бізнес-логіка додатка?
4. Перелічіть варіанти розподілу функцій стандартного інтерактивного додатка в архітектурі клієнт-сервер.
5. Назвіть основні дворівневі моделі архітектури клієнт-сервер.
6. У чому полягають особливості моделі віддаленого доступу до даних?

7. Якими засобами реалізується бізнес-логіка при використанні моделі віддаленої презентації?

1. Моделі клієнт-сервер у технології БД

Обчислювальна модель клієнт-сервер початково пов'язана з парадигмою відкритих систем, яка з'явилася в 90-х роках і швидко розвивалася. Термін клієнт-сервер початково застосовували до архітектури, за якої клієнтський процес запитує деякі послуги, а серверний процес забезпечує їх виконання.

Реалізація архітектури клієнт-сервер, стосовно розробки БД, дає змогу більш повно використовувати ресурси мережі. Навантаження рівномірно розподіляється між комп'ютером сервером і комп'ютером клієнтом, який, як і сервер, володіє власними ресурсами.

Основний принцип технології клієнт-сервер стосовно технології БД полягає в поділі функцій стандартного інтерактивного додатка на 5 груп, що мають різну природу:

- Функції введення і відображення даних (Presentation Logic).

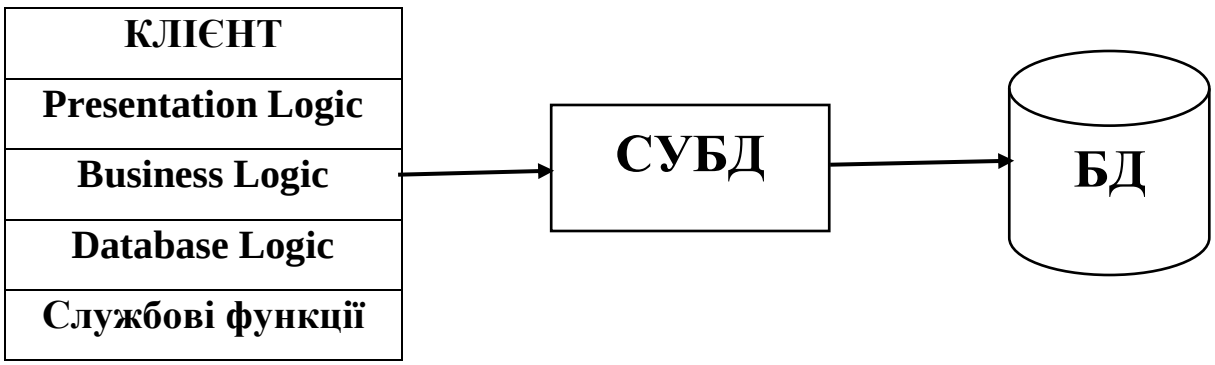
- Прикладні функції, що визначають основні алгоритми розв'язання задач програми (Business Logic).

- Функції опрацювання даних усередині програми (Database Logic).

- Функції управління інформаційними ресурсами (Database Manager System).

- Службові функції, що відіграють роль зв'язок між функціями перших 4-х груп.

Структуру типового інтерактивного додатка, що працює з БД, наведено на рисунку:



Презентаційна логіка - це частина програми, яка визначає те, що користувач бачить на екрані. Сюди відносяться інтерфейсні екранні форми, а також все, що

виводиться користувачеві на екран, як результати розв'язання проміжних завдань або довідкова інформація.

Основними завданнями презентаційної логіки є:

- формування екранних зображень;

- читання і запис в екранні форми інформації;

- управління екраном;

- обробка рухів миші та натискання клавіш клавіатури.

Бізнес-логіка або логіка застосунків - це частина коду застосунку, яка визначає власне алгоритми розв'язання задач застосунку. Зазвичай цей код пишеться за допомогою різних мов програмування: C, Cobol, Visual Basic.

Логіка опрацювання даних - це частина коду застосунку, яка пов'язана з опрацюванням даних усередині застосунку. Даними керує власне СУБД. Для забезпечення доступу до даних використовуються мова запитів і засоби маніпулювання даними мови SQL. Процесор управління даними - це власне СУБД, яка забезпечує зберігання та управління БД.

У централізованій архітектурі ці функції розташовуються в єдиному середовищі та комбінуються всередині виконуваної програми. У децентралізованій архітектурі ці завдання можуть бути по-різному розподілені між серверним і клієнтським процесами. Залежно від характеру розподілу можна виділити такі моделі розподілів:

- розподілена презентація; (частина подання на клієнті, частина на сервері, на сервері - усі інші частини)

- віддалена презентація; (уся презентація на клієнті - все інше на сервері)

- розподілена бізнес-логіка; (презентація і частина бізнес-логіки на клієнті)

- розподілене управління даними; (презентація, бізнес-логіка, і частина управління даними на клієнті);

- віддалене управління даними (презентаційна і бізнес-логіка на клієнті, решта на сервері).

- розподілена БД.

Ця класифікація показує, як завдання можуть бути розподілені між серверним і клієнтським процесами.

Дворівневі моделі

Дворівнева модель передбачає розподіл усіх зазначених функцій між 2-ма процесами, які виконуються на 2-х платформах - клієнті та сервері.

У чистому вигляді майже ніяка модель не існує.

2. Модель віддаленого доступу до даних

У цій моделі презентаційна логіка і бізнес-логіка розташовуються на клієнті. На сервері розташовується база даних і ядро СУБД (див. рисунку).

Звернення за сервісом управління даними відбувається за допомогою мови SQL. Перевага моделі - наявність великої кількості готових СУБД, що мають SQL-інтерфейси і набір інструментальних засобів, що забезпечують створення клієнтських додатків. У цій моделі мережею передаються SQL-запити, у відповідь на запити клієнт отримує не блоки файлів, а тільки дані, релевантні запиту. Основна перевага - уніфікація інтерфейсу клієнт-сервер, стандартом при спілкуванні клієнта і сервера стає мова SQL. Недоліки: досить високе завантаження системи передачі даних, внаслідок того, що вся логіка зосереджена в додатку, а оброблювані дані розташовані на віддаленому вузлі.

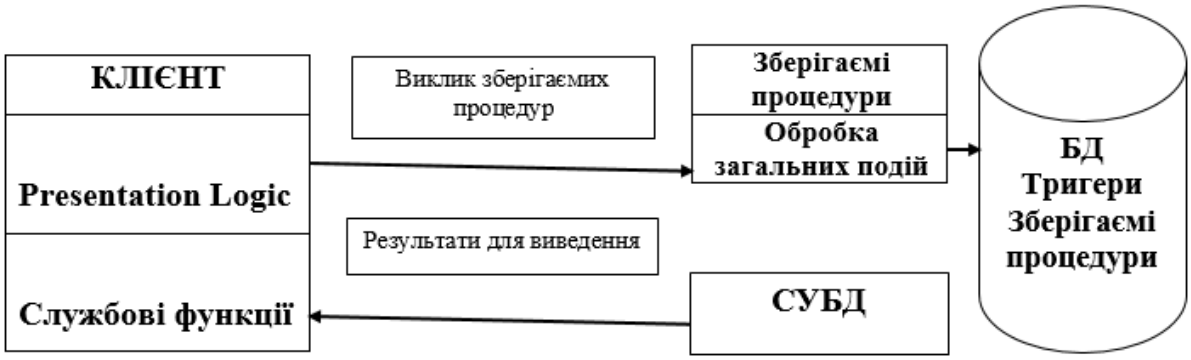
Ці системи незручні з точки зору модифікації та супроводу. Навіть у разі незначної зміни функцій системи потрібна переробка всієї прикладної частини. Оскільки на клієнті розміщена і презентаційна логіка, і бізнес-логіка застосунку, то в разі повторення аналогічних функцій у різних застосунках код відповідної бізнес-логіки має бути повторено для кожного клієнтського застосунку.

Сервер у цій моделі відіграє пасивну роль, тому функції інформаційного управління мають виконуватися на клієнті. Наприклад, якщо необхідно виконувати контроль страхових запасів на складі, то кожен застосунок, що пов'язаний зі зміною стану складу, після виконання операцій модифікації даних, що імітують продаж або видалення товару зі складу, має виконувати перевірку на обсяг залишку. У разі якщо

він менший за страховий запас, необхідно формувати відповідну заявку на поставку необхідного товару. Це може викликати необґрунтоване замовлення додаткових товарів кількома додатками.

3. Віддалена презентація (Модель сервера БД)

Модель сервера БД наведено на рисунку.



Модель сервера БД вирізняється тим, що функції комп'ютера клієнта обмежуються поданням інформації, тоді як прикладні функції забезпечуються додатком, який знаходиться на комп'ютері-сервері. Ця модель є більш технологічною, ніж модель віддаленого доступу.

Для того щоб позбутися недоліків моделі віддаленого доступу, мають бути дотримані такі умови:

- Необхідно, щоб БД у кожен момент відображала поточний стан предметної області.

- БД має відображати деякі правила предметної області, закони, за якими вона функціонує. Наприклад, завод може нормально функціонувати тільки в тому разі, коли є достатній запас деталей певної номенклатури, деталь можна запустити у виробництво тільки в тому разі, якщо на складі є достатньо матеріалу для її виготовлення тощо.

- Необхідний постійний контроль над станом БД, відстеження всіх змін і адекватна реакція на них. Наприклад, у разі зменшення товарного запасу нижче за критичний рівень має бути сформована заявка на поставку відповідного товару.

Таку модель підтримують більшість сучасних СУБД: Informix, Oracle, Sybase, MS SQL Server. Основу цієї моделі становить механізм збережених процедур як засіб програмування SQL сервера, механізм тригерів як механізм відстеження поточного

стану інформаційного сховища та механізм обмежень на призначені для користувача типи даних, який називається механізмом підтримки доменної структури. Процедури зазвичай зберігаються в словнику БД і поділяються кількома клієнтами. Збережені процедури можуть виконуватися в режимах інтерпретації та компіляції. Клієнтський додаток звертається до сервера з командою запуску збереженої процедури, а сервер виконує цю процедуру і реєструє всі зміни в БД, які в ній передбачені. Сервер повертає клієнту дані, що відповідають його запиту, які потрібні або для виведення на екран, або для виконання частини бізнес-логіки, яка розташована на клієнті. Трафік обміну інформацією між клієнтом і сервером помітно зменшується.

Централізований контроль цілісності даних у моделі сервера БД виконується з використанням механізму тригерів. Тригери також є частиною БД. Термін тригер узятий з електроніки і семантично точно відображає механізм відстеження спеціальних подій, які пов'язані зі станом БД. Ядро СУБД проводить моніторинг усіх подій, які спричиняють створені та описані тригери в БД, і при настанні відповідної події сервер запускає відповідний тригер. Тригер являє собою деяку програму, яка виконується над БД. Тригери можуть викликати збережені процедури.

У цій моделі сервер є активним, тому що не тільки клієнт, а й сам сервер, використовуючи механізм тригерів, може бути ініціатором обробки даних у БД. І збережені процедури, і тригери зберігаються в словнику БД, вони можуть бути використані кількома клієнтами, що істотно зменшує дублювання алгоритмів обробки даних у різних клієнтських додатках. Для написання збережених процедур і тригерів використовується розширення стандартної мови SQL.

Переваги моделі - можливість гарного централізованого адміністрування додатків на етапах розробки, супроводу та модифікації, а також ефективне використання обчислювальних і комунікаційних ресурсів. Один із недоліків моделі пов'язаний з обмеженнями засобів розробки збережених процедур. Основне обмеження - сильна прив'язка операторів збережених процедур до використовуваної СУБД. Мова написання збережених процедур, по суті, є процедурним розширенням мови SQL і не може змагатися за функціональними можливостями з традиційними мовами, такими як С або Паскаль. Інший недолік - дуже велике завантаження сервера.

Якщо ми переклали більшу частину бізнес-логіки застосунку на сервер, то вимоги до клієнтів у цій моделі різко зменшуються. Іноді таку модель називають моделлю з тонким клієнтом, на відміну від попередніх, де на клієнта покладалися набагато серйозніші завдання.

Можлива модель розподіленої бізнес-функції, у якій загальну частину бізнес-функцій реалізовано на сервері, а специфічні функції опрацювання інформації знаходяться на клієнті. Функції загального характеру можуть містити стандартне забезпечення цілісності даних, наприклад, у вигляді збережених процедур, а прикладні функції, що залишилися, реалізують спеціальне прикладне оброблення.

4. Модель розподіленої БД

Ця модель передбачає використання потужного комп'ютера клієнта, причому дані зберігаються і на комп'ютері-клієнті, і на комп'ютері-сервері. Взаємозв'язок обох БД може бути 2-х різновидів:

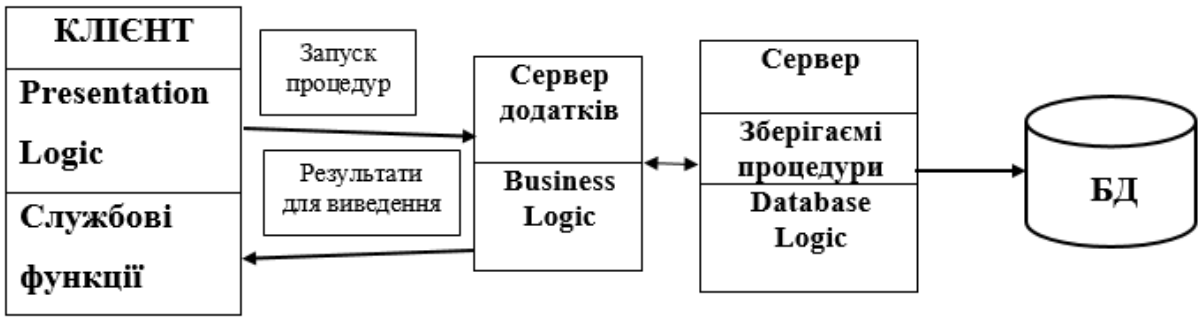
а) у локальній і віддаленій базах зберігаються окремі частини єдиної БД;

б) локальна і віддалена БД є копіями, що синхронізуються одна з одною.

Перевага - гнучкість розроблюваних ІС, що дають змогу комп'ютеру клієнту обробляти і локальні, і віддалені БД. Недолік - високі витрати при виконанні великої кількості однакових додатків на комп'ютерах клієнтах.

5. Модель сервера додатків

Ця модель є розширенням 2-рівневої моделі і в ній вводиться додатковий проміжний рівень між клієнтом і сервером. Цей проміжний рівень містить один або кілька серверів додатків.



У цій моделі компоненти діляться між трьома виконавцями:

6. Клієнт забезпечує логіку подання, включно з графічним користувацьким інтерфейсом; клієнт може запускати локальний код додатка клієнта, що може містити звертання до локальної БД, яка розміщена на комп'ютері-клієнті. Клієнт

виконує комунікаційні функції, які забезпечують доступ клієнту в локальну або глобальну мережу.

7. Сервери додатків являють собою новий додатковий рівень архітектури. На сервері додатків реалізується кілька прикладних функцій, кожна з яких оформлена як служба надання послуг усім програмам, які цього потребують. Серверів додатків може бути кілька, причому кожен з них надає свій вид сервісу. Будь-яка програма, що запитує послугу у сервера додатків, є для нього клієнтом. Запити, що надходять до серверів від клієнтів, поміщаються в чергу.

8. Сервери БД у цій моделі займаються винятково функціями СУБД: забезпечують створення і ведення БД, забезпечують функції сховищ БД, крім того, на них покладаються функції створення резервних копій, відновлення БД після збоїв, управління виконанням транзакцій.

Ця модель має більшу гнучкість, ніж дворівнева модель.