**Поняття Бази даних та системи управління базами даних (СУБД). Створення структури бази даних.**

**Основні поняття**

Базами даних (БД) називають електронні сховища інформації, доступ до яких здійснюється за допомогою одного або декількох комп'ютерів. База даних складається з таблиць. Таблиці складаються з полів (стовпців) і записів (рядків). Розробник бази даних визначає структуру БД, т. Е. Створює поля (задає ім'я, визначає тип і властивості полів), а користувач наповнює її, т. Е. Вводить, змінює або видаляє записи.

Системи Управління Базами Даних (СУБД) - це програмні засоби, призначені для створення, наповнення і видалення баз даних. За призначенням СУБД поділяються на три види: Промислові універсального призначення, Промислові спеціалізовані і розробляються під конкретного замовника. Універсальні розраховані «на всі випадки життя» і, як наслідок, або дуже складні у використанні і вимагають від користувача спеціальних знань, або прості, але обмежені в можливостях. Прикладом універсальних СУБД можуть служити Access, FoxPro, Oracle, DB2. Спеціалізовані спрямовані на виконання вузьких задач і тому створюються так, щоб вони були прості у використанні для професіоналів у своїй галузі. Прикладом таких СУБД можуть служити різні бухгалтерські або складські програми (БЕСТ, 1С Підприємство, Правова система Гарант). СУБД, що розробляються під конкретного замовника, максимально враховують потреби споживача, його ситуацію і не вимагають додаткових знань від користувача. Але вони досить дорогі і вимагають часу для створення, налагодження та впровадження, тоді як Універсальні і спеціалізовані порівняно дешеві і вводяться в експлуатацію за порівняно короткий термін (від тижня до місяця).

По розташуванню СУБД поділяються на локальні і розподілені (віддалені). Всі частини локальної СУБД розташовані на одному комп'ютері. Локальні СУБД можуть працювати в мережі, але в будь-якому випадку все її частини знаходяться на одному комп'ютері (локально). На відміну від локальних, розподілені СУБД працюють тільки при наявності комп'ютерної мережі і розташовуються, як мінімум, на двох комп'ютерах. Значна частина програмно-апаратних засобів розподіленої СУБД централізована і розташована на досить потужному комп'ютері (сервері). На комп'ютерах користувачів розташована тільки невелика частина СУБД (клієнт), що дозволяє зв'язуватися з головною частиною. Розподілені СУБД ще називають Клієнт-серверні СУБД.

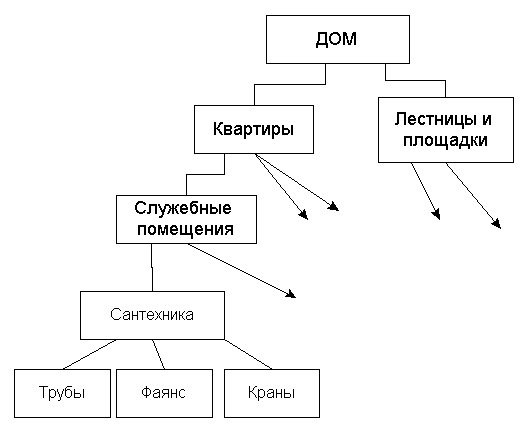
У кожній таблиці БД повинен існувати первинний ключ - одне або кілька полів, однозначно визначають кожен запис таблиці. Значення первинного ключа повинно бути унікальним, тобто в таблиці не повинно бути двох або більше записів з однаковим значенням первинного ключа. Наприклад, якщо є таблиця за накладними якогось складу і нумерація накладних кожен місяць починається з 1, то первинним ключем можуть виступати набір з двох полів: Номер і Дата накладної. Тому як можуть бути записи з однаковим номером накладної або з однаковою датою, але не може бути записи, в якій будуть однакові і номер, і дата.

Бази даних змінюються в рамках транзакції. Під транзакцією розуміється вплив на базу даних, що переводить її з одного цілісного стану в інший. Вплив виражається в зміні даних в таблицях бази. Якщо одна зі змін, що вносяться до БД в рамках транзакції, завершується неуспішно, повинен бути проведений відкат до стану БД, яка мала місце до початку транзакції. Отже, зміни в рамках однієї транзакції або одночасно підтверджуються, або не підтверджується жодне з них.

**Коротка історія розвитку баз даних.**

До появи СУБД всі дані в комп'ютерних системах зберігалися у вигляді окремих файлів. Система управління файлами (СУФ), яка зазвичай була частиною операційної системи, стежила за іменами файлів і їх розташуванням. У системах управління файлами нічого не знали про внутрішню будову файлів даних. Цю інформацію мали тільки програми, які працювали з цими даними. Тобто кожен додаток для роботи з даними містило опис структури даних. Такий підхід призводив до того, що навіть при незначних змінах в структурі файлу даних доводилося змінювати всі програми, які використовували цей файл. Згодом кількість і обсяг файлів росли і на супроводу СУФ, а також розробку нових додатків потрібно усе більше і більше зусиль.

Проблеми з супроводом великих систем. заснованих на файлів привели в кінці 60-х до появи створення СУБД. Ідея СУБД полягала у вилученні визначення структури файлів даних з додатків, що працюють з ними.

Перші СУБД мали ієрархічну структуру (ієрархічні СУБД). Оскільки СУБД застосовувалися в основному в економіці, а їх застосування було пов'язане з плануванням виробництва, то така структура найбільш повно відповідала потребам промислових підприємств. Наприклад, нехай є якась будівельна компанія, яка отримала підряд на будівництво житлового комплексу в одному з мікрорайонів. Для того, щоб знати скільки необхідно закупити будівельних матеріалів, необхідно мати уявлення про те, з чого складається будинок, потім з яких частин складаються його складові, далі з чого складаються ці складові і так далі. Наприклад, в будинку є службові приміщення (туалет, ванна кімната), для них необхідні труби, фаянс і крани, труби в свою чергу діляться на водопровідні, опалення і каналізаційні. 

Такий список є за своєю природою ієрархічним. Для зберігання даних, що мають таку структуру, були розроблені ієрархічна модель даних. У такій моделі для описаного випадку кожна запис являє собою конкретний будівельний вузол. Між записами існують відносини предок / нащадок, що зв'язують кожен конкретний вузол з його складовими. Однією з найбільш популярних ієрархічних СУБД була Information Managment System (IMS) від компанії IBM, що з'явилася в 1986 році. Перевагами даної СУБД була простота моделі даних, використання відносин предок / нащадок (що дозволяло робити висновки «А є частиною Б» або «А належить Б»), а також прекрасна швидкодія.

Якщо структура даних виявлялася складніше, ніж традиційна ієрархія, то простота організації ієрархічної бази даних ставала її недоліком. Наприклад, якщо розглянути роботу торгової компанії, то одне замовлення може брати участь в декількох аспектах предок / нащадок: з замовником, з менеджером або торговою точкою, що відпустила товар, а також з самим товаром. Однак ієрархія допускає наявність лише одного відносини між її записами. У зв'язку з цим для таких додатків була розроблена мережева модель даних, що допускала множинні відносини типу предок / нащадок. Такі відносини називалися множинами. У 1971 році на конференції з мов обробки даних (Conference on Data System Languages ​​- CODASYL) був опублікований стандарт мережевих баз даних, який відомий як модель CODASYL.

З точки зору програміста, доступ до мережевої бази даних був дуже схожий з доступом до ієрархічної бази даних. Прикладна програма могла

• знайти певний запис предка по ключу,

• перейти до першого нащадка в конкретній множені,

• перейти в сторону від одного нащадка до іншого,

• перейти вгору від нащадка до його предка.

І знову програмісту доводилося шукати інформацію в базі даних, послідовно перебираючи всі записи, але тепер він можу вказати не тільки напрямок, але і необхідне відношення.

Плюсами мережевих СУБД були:

• Гнучкість. Мережеві СУБД дозволяли працювати з даними, що мають досить складну структуру.

• Стандартизація. Прийняття стандарту CODASYL призвело до полегшення створення нових додатків і переносимості даних.

• Швидкодія. Не дивлячись на складність моделі даних, мережеві СУБД дозволяли досягати швидкодії, який можна порівняти з швидкодією ієрархічні СУБД.

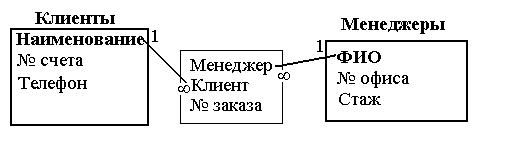
І все-таки, як і ієрархічні СУБД, мережеві мали безліч недоліків. Так зміна структури бази даних означало перебудову всього програми. Набори відносин і структуру записів випливало ставити наперед. Для того щоб отримати дані, програмісту необхідно було писати програму для навігації по базі даних, що могло зайняти від декількох днів до декількох тижнів, а інформація на той час могла виявитися марною.

Недоліки мережевих і ієрархічних СУБД привели до підвищення інтересу до нової реляційної моделі даних, описаної доктором Коддом в 1970 році. Реляційна модель була спробою спростити структуру бази даних. У ній були відсутні явні покажчики на предків і нащадків, а всі дані були представлені у вигляді простих таблиць, розбитих на рядки і стовпці, на перетині яких знаходяться дані. У кожної таблиці є певне ім'я, яке описує її вміст (слід зазначити, що грамотне іменування таблиць значно впливає на простоту сприйняття бази даних розробниками). Між таблицями існують зв'язку. Розрізняють три види зв'язків: один-до-одного, один-ко-многим, багато-до-багатьох. Саме від англійського слова зв'язок (relation) і пішла назва реляційні бази даних.

**Зв'язок один-до-одного** пов'язує дві таблиці по їх ключових полях. Це може застосовуватися в тих випадках, коли для одного ключового поля існує два великі блоки інформації, що відрізняються за змістом. Наприклад, успішність студента (перший блок) і його паспортні дані (другий блок). У цьому випадку створюються дві таблиці і потім вони зв'язуються по ключовому полю, припустимо за номером залікової книжки. При цьому запису в таблиці-нащадку відповідає одна запис в таблиці-предка.

**Зв'язок** **один-до-багатьох** є найпоширенішою в реляційних базах даних. Наведемо простий приклад. Нехай є база даних торгової компанії, в якій зберігаються дані з відпуску товару. Можна створити кілька таблиць: таблиця-довідник про товари (назва, одиниці виміру. Ціна за одиницю і т. Д.), Таблиця-довідник клієнтів і так далі. Однією записи в таблиці-довіднику може відповідати кілька записів у таблиці обліку відпуску товарів. Наприклад, одному клієнту відповідає кілька замовлень, зроблених ним. При цьому таблиця-довідник є предком (батьківської таблицею), а таблиця обліку відпуску товару є нащадком (дочірньої таблицею). У зв'язку беруть участь ключове поле батьківської таблиці і не ключове поле дочірньої.

**Зв'язок** **багато-до-багатьох** також використовується досить часто. Наведемо такий приклад: нехай при обліку відпуску товару записується інформація про менеджера, відпустили товар. Оскільки клієнт робить замовлення в різний час, то він може оформити замовлення у різних менеджерів, а кожен менеджер може відпустити товар кільком клієнтам. Таким чином, однією записи з таблиці-довідника менеджерів може відповідати кілька записів з таблиці-довідника клієнтів і одночасно навпаки, одного запису з таблиці-довідника клієнтів може відповідати кілька записів у таблиці-довіднику менеджерів. Відповідно до теорії реляційних баз даних для зберігання взаємозв'язку "багато-до-багатьох" потрібні три об'єкти: по одному для кожної сутності і один для зберігання зв'язків між ними (проміжний об'єкт). Проміжний об'єкт буде містити ідентифікатори пов'язаних об'єктів. На наведеному малюнку для зв'язку між двома таблицями-довідниками використовуєтьсятретя.



**Проектування баз даних. Нормалізація.**

Процес проектування БД в чималому ступені залежить від досвіду і інтуїції розробника. Є творчим. Але як бути новачкові, у якого немає досвіду створення баз даних? Крім того, бази даних зберігають величезну кількість інформації, тому можуть займати багато місця в пам'яті комп'ютера, в зв'язку з чим виникає необхідність їх оптимізації. Для цих цілей були розроблені певні правила створення структури БД - нормальні форми. Процес приведення БД до нормальної формі називається нормалізацією. Всього існує сім нормальних форм, але ми розглянемо тільки три з них.

**Перша нормальна форма (1НФ)** вимагає, щоб кожне поле таблиці БД було неподільним і не містило повторюваних груп.

Неподільність поля означає, що міститься в ньому інформація не повинна ділитися на більш дрібні.

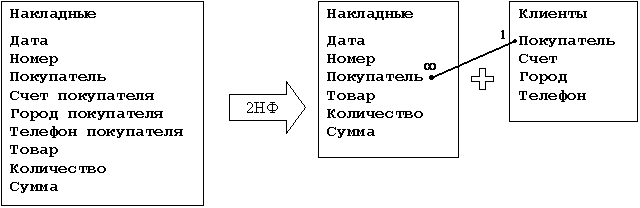
Прімер1.1: поле ПІБ можна розділити на три окремих поля Прізвище, Ім'я, По батькові.

Прімер1.2: не можна використовувати поле Адреса, потрібно використовувати набір полів: Вулиця, Номер будинку, Квартира, Місто.

Для чого це робиться? Припустимо, у вашій БД існує поле Адреса, тоді вам буде дуже важко знайти всіх людей, що живуть на певній вулиці. Вам доведеться писати спеціальну конструкцію на мові запитів для аналізу кожного запису, що надзвичайно сповільнить пошук. При поділі поля ви просто в одне натискання включаєте фільтр по полю і отримуєте бажані записи.

Повторюваними є поля, що містять однакові за змістом значення.

Прімер1.3: Нехай є якесь підприємство, що продає крупи. І для них була створена база даних містить, крім даних про накладної та клієнта ще й такі поля: Цукор, Сіль, Борошно, Гречка, Рис, Пшоно в яких зберігається кількість відпущеного товару. На перший погляд все добре, але кожне поле бази даних незалежно від того містить воно інформацію чи ні, все одно займає місце в пам'яті, тобто розмір, займаний порожній записом і повністю заповненою записом однаковий.

Припустимо, що справи у компанії пішли добре і вона змогла розширитися і стала продавати ще і молокопродукти. Якщо залишити структуру не пошкоджене, то в таблицю необхідно додати нові поля, причому вже наявні записи будуть змінені. А якщо далі число продуктів, що продаються фірмою перевищить хоча б сотню, то чи буде зручно орієнтуватися в такій таблиці? У подібній ситуації слід зауважити, що всі наведені поля суть найменування товару і зберігають вони лише кількість товару, тому досить використовувати всього два поля: ***Товар, Кількість.***

Приклад 1.4. Приведення таблиці до 1НФ.

*Неприведенна*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ФИО** | **Тетради** | **Карандаши** | **Ручки** | **Ластики** |
| Иванов Иван Иванович |  | 15 |  |  |
| Петров Петр Петрович | 10 |  |  |  |

*Приведенна к 1НФ*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Фамилия** | **Имя** | **Отчество** | **Канц. тов.** | **Кол-во** |
| Иванов | Иван | Иванович | Карандаши | 15 |
| Петров | Петр | Петрович | Тетради | 10 |

**Друга нормальна форма (2НФ)** вимагає, щоб всі поля таблиці залежали від первинного ключа, тобто, щоб первинний ключ однозначно визначав запис і не був надмірний. Ті поля, які залежать тільки від частини первинного ключа повинні бути виділені в окремі таблиці. Припустимо у нас є таблиця для відпуску товару за накладними. Цю таблицю ми повинні привести до другої нормальної форми. Розглянемо це тільки схематично:  
Первинним ключем тут є поля Дата і Номер. Поля: Рахунок, Місто і Телефон - залежать тільки від поля Покупець, але не від первинного ключа, тому їх можна виділити в окрему таблицю під назвою Клієнти. Таким чином, ми привели БД Накладні до другої нормальної форми.

**Третя нормальна форма (3НФ)** вимагає, щоб в таблиці не було транзитивних залежностей між неключових полями, т. Е. Щоб значення будь-якого поля, що не входить в первинний ключ, не залежало від значення іншого поля, також не входить у первинний ключ.

Типовим прикладом може служити зв'язка полів: Ціна за одиницю, Кількість, До оплати. Справа в тому, що значення поля До оплати залежить від полів Ціна за одиницю і Кількість. Значення поля До оплати можна обчислювати в процесі створення звіту або запиту, але зберігати його в Базі даних зовсім не обов'язково.

Нормалізація таблиць дозволяє усунути з них надлишкову інформацію, що призводить до зменшення розміру бази даних. У той же час надмірне захоплення нормалізацією може привести до того, що пошук по БД буде йти дуже повільно, оскільки СУБД повинна буде зчитувати дані з декількох таблиць, витрачаючи час на їх відкриття, перегляд і закриття. Тому ви завжди повинні визначити для себе що є найбільш важливим: розмір бази або швидкість пошуку. Слід зазначити, що перші дві нормальні форми застосовуються дуже часто, третя трохи рідше. При грамотному використанні нормалізації розробником, користувач отримує логічно вибудувану, зручну у використанні і компактну базу даних.

Нехай дано таке завдання:

Технічне завдання на створення бази даних. Варіант 0.

a) База Даних (БД) повинна бути приведена до третьої нормальної формі і містити наступну інформацію: Найменування клієнта, Адреса клієнта, Номер рахунку Клієнта, ПІБ обслуговуючого агента, Номер офісу обслуговуючого агента, Місцезнаходження компанії, Телефон офісу, Сума всіх контрактів агента, Номер договору, Дата укладення договору, Сума за контрактом.

b) Створити форми для введення інформації в кожну з створених таблиць.

c) Створити три запити:

a. Номер контракту, Найменування клієнта, ПІБ обслуговуючого агента, сума контракту. Сортування по номеру контракту в порядку зростання. Вивести інформацію для вказаного користувачем агента.

b. ПІБ обслуговуючого агента, Кількість контрактів, Загальна сума всіх контрактів. Вивести інформацію тільки для зазначеного користувачем офісу.

с. Номер офісу, Кількість контрактів, Загальна сума всіх контрактів, Середня сума всіх контрактів. Вивести інформацію тільки за минулий місяць.

d) Створити звіти на підставі наявних запитів.

e) Створити основну кнопкову форму.

**Почнемо виконання завдання.**

**Наведемо базу даних до першої нормальної формі.** У базі даних є кілька полів, які можна розділити на кілька неподільних. Це, перш за все, поля Адреса. Розіб'ємо їх, вважаючи, що всі клієнти і офіси знаходяться в місті.

Адреса клієнта (Місто Клієнта, Індекс клієнта, Вулиця клієнта, Номер будинку клієнта, Номер кімнати).

Адреса Офісу (Місто офісу, Індекс офісу, Вулиця офісу, Номер будинку офісу, Номер кімнати).

Наступним діленим полем є поле ПІБ обслуговуючого агента, ділимо його на три поля: Прізвище агента, Ім'я агента, По батькові агента.

У нашій таблиці немає полів, які можна було б розглядати як однотипні. Таким чином приведення до першої нормальної формі закінчено, і ми визначилися з набором полів нашої бази даних:

| **Первая Нормальная форма** |
| --- |
| Наименование клиента  Город Клиента  Индекс клиента  Улица клиента  Номер дома клиента  Номер комнаты клиента  Номер счета Клиента  Фамилия обсл. Агента  Имя обсл. Агента  Отчество обсл. Агента  Номер офиса обсл. Агента  Сумма всех контрактов агента  Город офиса  Индекс офиса  Улица офиса  Номер дома офиса  Номер комнаты офиса  Телефон офиса  Номер договора  Дата заключения договора  Сумма по договору |

**Наведемо нашу базу даних до другої нормальної форми.** Для цього необхідно визначити первинний ключ. Якщо уважно подивитися на наведений список полів, то стане ясно, що це база даних, що зберігає перелік договорів якоїсь фірми. Кожне з наведених полів може мати повторювані значення, тому первинний ключ буде складатися як мінімум з двох полів. В якості первинного ключа можна вибрати Дату і номер договору, оскільки в один і той же день не можна підписати договір з однаковим номером.

Поле Найменування клієнта залежить від первинного ключа, а вся інша інформація про клієнта від первинного ключа не залежить, але залежить від поля Найменування клієнта. У зв'язку з цим виділяємо ці поля в окрему таблицю Клієнти. Далі Прізвище обслуговуючого агента залежить від первинного ключа, а вся інша інформація про агента від первинного ключа не залежить, тому виділяємо ці поля в окрему таблицю Агенти. В навчальних цілях можна вважати, що в нашій фірмі немає однофамільців, тому в новій таблиці в якості первинного ключа виберемо прізвище агента. Решта поля таблиці Договору залежать від первинного ключа.

У таблиці Агенти від ключа (поля Прізвище) не залежить поля, що зберігають адресу офісу та телефон офісу, тому виділяємо їх в окрему таблицю Офіси.

Таким чином структура нашої бази даних набуде вигляду:

| **Договор** | **Клиенты** | **Агенты** | **Офисы** |
| --- | --- | --- | --- |
| **Номер договора**  **Дата заключения договора**  Наименование клиента  Фамилия обсл. Агента  Сумма по договору | **Наименование клиента**  Город Клиента  Индекс клиента  Улица клиента  Номер дома клиента  Номер комнаты клиента  Номер счета Клиента | **Фамилия обсл. Агента**  Имя обсл. Агента  Отчество обсл. Агента  Номер офиса обсл. Агента  Сумма всех контрактов агента | **Номер офиса**  Город офиса  Индекс офиса  Улица офиса  Номер дома офиса  Номер комнаты офиса  Телефон офиса |

Зв'язки між таблицями:

Клиенты (Наименование клиента) – Договор (Наименование клиента);

Агенты (Фамилия обсл. Агента) – Договор (Фамилия обсл. Агента);

Офисы (Номер офиса) – Агенты (Номер офиса обсл. Агента).

**Наведемо базу даних до третьої нормальної форми.** У таблиці Агенти є поле **Сумма всех контрактов Агента**. Значення цього поля залежить від значень поля **Сумма по договору** і може бути обчислена на етапі створення звіту або запиту. Таким чином, це поле прибираємо з бази даних. Інших таких полів немає.

Остаточний вигляд структури бази даних:

| **Договор** | **Клиенты** | **Агенты** | **Офисы** |
| --- | --- | --- | --- |
| **Номер договора**  **Дата заключения договора**  Наименование клиента  Фамилия обсл. Агента  Сумма по договору | **Наименование клиента**  Город Клиента  Индекс клиента  Улица клиента  Номер дома клиента  Номер комнаты клиента  Номер счета Клиента | **Фамилия обсл. Агента**  Имя обсл. Агента  Отчество обсл. Агента  Номер офиса обсл. Агента | **Номер офиса**  Город офиса  Индекс офиса  Улица офиса  Номер дома офиса  Номер комнаты офиса  Телефон офиса |

Зв'язки між таблицями:

Клиенты (Наименование клиента) – Договор (Наименование клиента);

Агенты (Фамилия обсл. Агента) – Договор (Фамилия обсл. Агента);

Офисы (Номер офиса) – Агенты (Номер офиса обсл. Агента).

Отже, розробка структури бази даних завершена. Переходимо до безпосередньої роботи з СУБД Access.