Житомирський державний університет імені Івана Франка
Кафедра прикладної математики та інформатики

БАЗИ ДАНИХ ТА ІНФОРМАЦІЙНІ СИСТЕМИ

Житомир
Вид-во ЖДУ ім. І. Франка

УДК 004.65
З 34
ББК 32.937.26-018.2

Рекомендовано до друку вченою радою Житомирського державного університету імені Івана Франка (протокол № 5 від 27.11.09).

Рецензенти:

Спірін О. М. – доктор педагогічних наук, доцент, заступник директора з наукової роботи Інституту інформаційних технологій і засобів навчання;

Триус Ю. В. – доктор педагогічних наук, професор кафедри комп'ютерних технологій Черкаського державного технологічного університету;

Крижанівський В.Б. – кандидат фізико-математичних наук, доцент кафедри програмного забезпечення обчислювальної техніки Житомирського державного технологічного університету.

Зарицька О.Л.

З 34 Бази даних та інформаційні системи: Методичний посібник. – Житомир: Вид-во ЖДУ ім. І. Франка, 2010. – 132 с., ил:.

У методичному посібнику у стислій та систематизованій формі викладено теоретичний курс дисципліни.

Практичні роботи присвячені закріпленню на практиці умінь та навичок використання основних понять теорії баз даних; складанню ієрархічних, мережних та реляційних моделей; перетворенню елементів концептуальної моделі в реляційну схему; виконанню запитів засобами реляційної алгебри, реляційного числення та мови SQL.

Метою лабораторного практикуму є реалізація баз даних у системі Microsoft Access

Методичний посібник підготовлено відповідно до програми навчальної дисципліни "Бази даних та інформаційні системи". Для студентів вищих навчальних закладів спеціальностей "Інформатика", "Фізика та інформатика".

УДК 004.65
ББК 32.937.26-018.2

ЗМІСТ

ВСТУП

"Бази даних та інформаційні системи" входить до циклу дисциплін професійної науково-предметної підготовки, що формують кваліфікованих фахівців високого рівня. Засвоєння студентами цієї дисципліни поряд з освітньо-пізнавальними має й науково-прикладне значення як на початковому етапі навчання, так і в процесі формуванні фахівця загалом. Сучасний рівень інформаційних технологій вимагає від студентів, майбутніх фахівців, ґрунтовних знань з обліку й ефективної обробки даних засобами систем керування базами даних (СКБД).

У методичному посібнику простежено історію розвитку інформаційних систем на основі баз даних, особливості спільного використання даних, типи баз даних. Окремі розділи присвячені концептуальному проектуванню баз даних та переведенню інфологічної моделі в реляційну. Розглянуто процес нормалізації реляційних таблиць та мови обробки даних – реляційну алгебру й еквівалентного їй реляційного числення. Наведено приклади запису запитів та створення схем і таблиць мовою SQL.

Метою вивчення курсу є:

- формування ґрунтовної теоретичної бази знань студентів про об'єкти системи керування базами даних; властивості цих об'єктів; проектування інформаційних систем, баз даних і систем їх керування;
- формування практичних навичок із розробки логічної структури бази даних, створення таблиць баз даних та подальшої роботи з ними.

До *теоретичної бази знань* належить: етапи розвитку технології баз даних, ієрархічні та мережні моделі інформаційних систем, принципи концептуального проектування баз даних, реляційна модель даних, реляційна алгебра та реляційне числення, керування реляційною базою даних, мова SQL, хеш-організація.

До *практичних навичок* належать: навички розробки логічної структури бази даних у процесі технічного проектування; навички розробки таблиць баз даних, введення, модифікації, вилучення, відображення даних, використовуючи форми введення та модифікації даних і табличні режими, використання мови SQL під час роботи з базою даних.

Мета курсу досягається через вивчення теоретичного матеріалу за кожною з визначених тем та поступове практичне оволодіння студентами навичками розробки, створення та експлуатації баз даних. Засвоєння теоретичних знань та оволодіння практичними навичками дасть студенту знання та вміння, що послужить суттєвим підґрунтям для вдосконалення майстерності роботи на ПЕОМ, а також подальшого використання комп'ютера у повсякденній діяльності висококваліфікованого фахівця.

Розраховано на студентів вищих навчальних закладів спеціальностей "Інформатика", "Фізика та інформатика".

ЧАСТИНА 1
БАЗИ ДАНИХ. ІНФОРМАЦІЙНІ СИСТЕМИ, ЩО ВИКОРИСТОВУЮТЬ БАЗИ ДАНИХ. ФІЗИЧНА ОРГАНІЗАЦІЯ БАЗ ДАНИХ

§ 1.1. ПОНЯТТЯ БАЗИ ДАНИХ. ІСТОРІЯ РОЗВИТКУ БАЗ ДАНИХ. ФАЙЛОВІ СИСТЕМИ. ІНФОРМАЦІЙНІ СИСТЕМИ, ЩО ВИКОРИСТОВУЮТЬ БАЗИ ДАНИХ

В історії розвитку обчислювальної техніки можна простежити два ключові напрямки її практичного застосування. Один із них передбачав використання обчислювальної техніки для здійснення складних розрахунків, що сприяло інтенсифікації методів розв'язування непростих математичних задач та розвитку групи мов програмування, орієнтованих на зручний запис алгоритмів. Другий напрямок окреслював шляхи використання засобів обчислювальної техніки для автоматичної обробки даних. Саме цей напрямок і є предметом вивчення дисципліни "Бази даних та інформаційні системи".

Бази даних, системи керування базами даних, системи обробки даних

У ході свого розвитку людство століттями накопичувало знання. Проте, лише у другій половині XX століття надзвичайної актуальності набуло питання сортування та обробки наявної інформації. Зберігання та обробка інформації – найважливіші функції комп'ютера. Саме в процесі розв'язування конкретних прикладних задач відбувається обробка потрібних даних за заданим алгоритмом. Дані можуть бути найрізноманітнішими: числа, прізвища, адреси, назви тощо. Точкою відліку нової інформаційної епохи стала саме поява перших баз даних (у подальшому – БД). У найзагальнішому випадку *БД – це файл спеціального формату, що містить певним чином структуровану інформацію.*

Протягом значного часу для розв'язування окремої задачі використовувалася власна сукупність даних (ця сукупність не могла бути використаною в іншій задачі). Такий спосіб використання даних є незручним. Зокрема, його недоліками є надлишковість та неузгодженість даних (інші недоліки буде розглянуто далі). Наприклад, для вищого навчального закладу розроблено програму підрахунку щорічного навантаження викладачів та програму обліку використання викладачами фондів бібліотеки. Для обох програм використовуються такі спільні дані: прізвище, ім'я, по батькові співробітника, його посада, належність до певної кафедри. Проте кожна зі створених програм використовувала ці однакові дані про викладача з власного файлу, тобто дані дублювалися (надли-

шковість). Зрозуміло, що зберігання одних і тих самих даних в одному файлі, незалежно від програми, що їх використовує, значно економить ресурси ПК. Неузгодженість даних полягає в тому, що технологічно важко простежити внесення змін одночасно у всі файли (наприклад, зміну прізвища викладача, його посаду тощо). Так, зміна прізвища викладача тимчасово могла бути відображена лише в одному з файлів даних, – у тому, що використовується для розрахунку навантаження, а в іншому файлі, – тому, що обслуговує бібліотечну програму, – залишитися без змін (зміни вчасно не внесені з технічних причин). Таким чином виникає неузгодженість даних, адже в обох випадках йтиметься про одну й ту саму особу, хоч комп'ютер буде оперувати двома різними прізвищами. Такі недоліки й спричинили на початку 60-х років минулого століття появу БД.

БД є проявом сучасності, оскільки створити аналог БД поза комп'ютером неможливо. Прототипами БД можна вважати бібліотечні каталоги або телефонні довідники. Проте, відомості в них є незмінними до чергового оновлення каталогу або перевидання довідника.

БД – значний упорядкований комплекс інформації, який зберігається на комп'ютерних носіях зазвичай разом із програмою, що дозволяє здійснювати швидку обробку даних (пошук, оновлення, друк тощо).

Без власної БД тепер не обходиться майже жоден навчальний заклад, державна установа, приватна фірма чи корпорація.

Сучасні технології БД є результатом розвитку протягом кількох останніх десятиліть способів обробки даних та керування наявною інформацією. Обробка даних розвивалася від примітивних методів 50-х років XX століття до складних інтегрованих систем сьогодення. Перші системи обробки даних виконували лише канцелярську роботу, скорочуючи тим самим паперовий обіг. Новіші системи накопичували інформацію та керували нею. Сьогодні для роботи з файлами баз даних використовують спеціальне програмне забезпечення – **системи керування базами даних (СКБД)**. Засобами СКБД здійснюється наповнення бази даних, пошук необхідної інформації, отримання звітів, створення нових або видалення існуючих БД, експортування та імпортування даних тощо. Фактично під базою даних частіше розуміють не саму базу даних як сховище інформації, а й супровідне програмне забезпечення, тобто СКБД+БД.

Перші комп'ютери використовувалися в основному в комерційних структурах для ведення бухгалтерського обліку, створення відомостей зарплатні тощо. Витрати ручної праці для ведення подібних відомостей або виписуванню рахунків були настільки значними, що автоматична система, яка могла б виконувати ці функції, швидко окупалася. Оскільки така система виконувала звичайні функції роботи з документами (які досі виконувалися вручну), використовуючи для цього дані комерційної структури, вона отримали назву системи обробки даних. Комп'ютерні файли виконували роль звичайних папок для паперів і містили певну інформацію. **Система обробки даних – автоматична система для роботи з даними установи.**

Дані та інформація. Інформаційні системи.

Наприкінці 60-х – на початку 70-х років минулого століття відбувся перехід від обробки даних до обробки інформації. Це було визнанням того, що інформація не є простим записом даних. У цьому контексті важливо розрізняти дані та інформацію. Під даними зазвичай розуміють розрізнені факти. Інформація – це організовані та оброблені дані або висновки з них.

На малюнку 1 а. у вигляді таблиць частково представлено дані файлів деякої уявної фірми (КЛІЄНТ, ТОРГОВИЙ_АГЕНТ, ТОВАР, УГОДИ, ВИРОБНИК). Зрозуміло, що кожен файл містить набагато більше рядків (записів), аніж представлено, та може містити додаткові стовпчики (поля). Приклад даних: Містер Клінтон проживає у Чикаго. Цей факт міститься в одному із записів файлу КЛІЄНТ. Подібних фактів файл може містити сотні. Приклад інформації: відомості про суму всіх угод клієнтів з України. Для отримання такої інформації потрібно проаналізувати дані з таблиць УГОДИ та КЛІЄНТ.

Отже, інформацію отримують у результаті обробки великої кількості даних. Інформація – це ресурс, що має велику цінність, а тому важливо навчитися впорядковувати його та керувати ним. Поступове визнання цього факту призвело до розуміння цінності інформації та потенціалу комп'ютерних систем у процесі підтримання такого значного ресурсу, як інформація. Так наприкінці 60-х з'явилися інформаційні системи. Такі системи використовували вже наявні в комп'ютері дані та давали відповіді на низку запитань.

Інформаційна система (ІС) – це програмний комплекс, функціями якого є: 1) підтримка надійного зберігання інформації в пам'яті комп'ютера; 2) виконання специфічних для конкретного додатка перетворень інформації й(або) обчислень; 3) надання користувачам зручного інтерфейсу.

Типовим прикладами ІС є банківські системи, системи резервування авіаційних або залізничних квитків, місць у готелях тощо.

Файлові системи

Перші комп'ютерні системи обробки даних використовували не зв'язані між собою локальні інформаційні файли лінійної структури. Суть такого підходу до організації інформаційного забезпечення полягала в тому, що файли проектувалися для кожної окремої задачі. Такі системи назвали файловими.

Файлова система – організація інформаційного забезпечення, за якої файли лінійної структури проектувалися нарізно для кожної окремої задачі.

Розглянемо таку систему на прикладі. На малюнку 1 а. у вигляді таблиць представлено кілька файлів (КЛІЄНТ, ТОРГОВИЙ_АГЕНТ, ТОВАР, УГОДИ, ВИРОБНИК) уявної фірми та зразки деяких даних цих файлів. Кожна з таблиць являє собою один файл. Кожен рядок таблиці відповідає одному запису в файлі. Елементарні групи даних файлу (*Код_клієнта, Прізвище_клієнта* тощо) є полями файлу.

КЛІЄНТ

Код_клієнта	*Прізвище_клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_сума_боргу*	*Виплачено_на_сьогодні*
100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
2000	Клінтон	Чикаго, Full_Street, 910	США	33 333	30 000

ТОРГОВИЙ_АГЕНТ

Код_агента	*Прізвище_агента*	*Офіс*	*Комісійний_%*
2	Сомов	Москва	11
3	Дяченко	Київ	10
4	Акінава	Токіо	13

ТОВАР

Код_товару	*Назва_товару*	*Код_виробника*	*Ціна_закупівельна*	*Ціна_продажу*
1	Чайний сервіз	400	100	120
2	Фортепіано	300	800	1 000
3	Стіл	200	300	350
4	Картина	400	135	150

УГОДИ

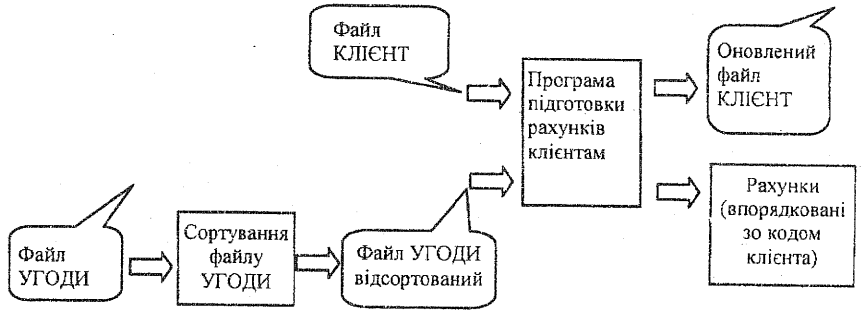
Дата	*Код_клієнта*	*Код_агента*	*Код_товару*	*Сума_угоди*
3.08.05	100	2	1	240
8.08.05	105	2	2	100
8.09.05	100	3	4	300
10.09.05	115	4	3	350
18.09.05	2000	2	4	150
28.10.05	115	3	3	350
2.11.05	2000	4	3	700
3.12.05	2000	2	3	1050

ВИРОБНИК

Код_виробника	*Назва_виробника*	*Адреса*	*Країна*
200	Фірма "Мебель"	Москва	Росія
300	Фабрика "Україна"	Харків	Україна
400	Артель "Усе разом"	Гонконг	Китай

Мал. 1 а. Дані файлів комерційної фірми

З файлами даних працювало багато різних програм. Зокрема, прикладна програма, що готує рахунки клієнтам, використовувала файли даних УГОДИ та КЛІЄНТ. Роботу такої прикладної програми зображено на мал. 1 б. Спочатку файл УГОДИ сортувався по полю *Код_клієнта* (файл КЛІЄНТ уже відсортовано за цим полем). Далі, об'єднуючи потрібні дані з цих файлів, програма роздруковувала рахунки. Під час роботи файл КЛІЄНТ змінювався. По-перше, оновлювалося поле *Загальна_сума_боргу*, по-друге, здійснені платежі (що підраховувалися іншою програмою) були внесені в поле *Виплачено_на_сьогодні*. Ці зміни відображалися також і в роздрукованих рахунках.



Мал. 1 б. Робота прикладної програми з підготовки рахунків клієнтам

Розглянемо цей приклад детальніше. Уже зазначалося, що обидва файли (УГОДИ та КЛІЄНТ) упорядковано за полем код клієнта. Проте після такого впорядкування файл УГОДИ абсолютно не впорядкований щодо коду товару. Таким чином прорахувати загальну суму продажу, звернувшись до файлу ТОВАРИ (де вказано ціну на всі товари), неможливо. Тому у файлі УГОДИ проставляється сума угоди (хоча за кількістю товару та його кодом система могла б сама прорахувати загальну суму угоди).

Розглянемо інший приклад. Прикладна програма підраховує загальну суму комісійних, які нараховують агентам. Потрібно знову впорядкувати файл *Угоди*, але цього разу за кодом агента. І лише після цього обробляються обидва файли УГОДИ та ТОРГОВИЙ_АГЕНТ. Схема обробки аналогічна до попередньої.

Файли послідовного та довільного доступу

Надійне та довгострокове зберігання інформації можливе лише за наявності запам'ятовуючих пристроїв, що зберігають інформацію після вимкнення комп'ютера.

Спочатку більшість файлів із даними зберігалися на магнітних стрічках, тому доступ до записів був **послідовний**. Це означає, що кожен запис у файлі міг бути прочитаний та оброблений лише після прочитання всіх попередніх записів цього файлу. Таким чином, можливості файлових систем із суто по-

слідовним доступом були обмеженими. Вони використовувалися для створення звітів, рахунків та платіжних відомостей 1-2 рази на місяць.

Проте для виконання великих обсягів рутинної роботи визріла необхідність довільного доступу до файлів. Саме з появою файлів довільного доступу проблеми, пов'язані з обмеженнями, що їх накладають вимоги суто послідовного доступу до файлів, частково були зняті. **Довільний доступ** – спосіб звертання до файлу, що забезпечує прямий доступ до конкретного запису (без попереднього сортування чи послідовного читання всіх записів). Пристроями, що забезпечували такий доступ, були магнітні барабани, які давали можливість довільного доступу до даних, але були обмеженого розміру. Широкого застосування в 60-х роках минулого століття набули індексно-послідовні файли, які давали змогу обрати одне або декілька полів для точного визначення того, який саме запис потрібно використати. Поле (або поля) даних, що однозначно визначає запис у файлі, називається **ключем**.

Недоліки традиційних файлових систем

Незважаючи на появу файлів із довільним доступом, стало очевидним, що файлові системи будь-якого типу мають невиправні недоліки. Серед яких надмірність та неузгодженість даних, недостатній контроль даних, слабкі можливості керування даними, великі затрати праці програміста. Розглянемо кожен із цих недоліків детальніше.

1. Надмірність та неузгодженість даних. Головна складність полягала в тому, що деякі програми використовували свої власні файли даних. Наприклад, у банках ім'я клієнта зберігалося у файлах, що містять відомості про поточний рахунок, ощадний рахунок та позики. Наслідком такої надмірності даних були додаткові витрати на підтримку та зберігання даних. Надмірність даних також породжує ризик протиріч між одними й тими ж даними в різних файлах. Наприклад, клієнт змінив прізвище. Зміни одразу занесені у й файл поточних рахунків, через декілька днів у файл ощадних рахунків, а у файлі із позиками зміни були внесені з помилкою. Така неодностайність може вплинути на точність звітів.

2. Недостатній контроль даних. У файлових системах відсутній централізований контроль на рівні елементів даних. Часто один і той самий елемент даних має різні імена залежно від того, в які файли він входить. Це пов'язано з тим, що різні відділи однієї корпорації можуть користуватися термінологією, що не є узгодженою з іншими. Виникає проблема синонімів та омонімів.

Синоніми – слова, що тотожні за змістом, проте мають різне написання.

Омоніми – слова, що мають різний зміст та однакове написання.

Наприклад, різні відділи страхової установи, використовуючи неоднакові назви – *Власник_полісу* та *Клієнт*, – можуть мати на увазі одну й ту саму особу (приклад проблеми синонімів). Водночас під терміном *Рахунок* різні відділи банку можуть розуміти як рахунок банківського вкладу, так і рахунок банківської позики (проблема омонімів).

3. Слабкі можливості керування даними. Індексно-послідовні файли звертаються до конкретного запису за ключовим полем (ключем), наприклад, за кодом товару. Якщо ми знаємо код чайного сервізу (див. мал. 1 а.), то ми можемо видобути з файлу *Товар* відповідний запис. Проте цього вистачає лише доти, доки нам потрібен лише один запис. Якщо ж потрібно відшукати низку пов'язаних між собою записів, таку інформацію з файлової системи добути майже неможливо, оскільки у файлових системах не передбачено зв'язків між даними різних файлів. Приклади завдань, розв'язок яких є неможливим за умов файлових систем:

- назвати всі товари, які придбав клієнт Петренко;
- підрахувати загальну кількість таких продаж та їхню середню ціну;
- назвати виробника кожного з придбаних клієнтом Петренком товарів.

Цю проблему – спрощення зв'язків між даними з різних файлів – у подальшому вирішили СКБД, які й були для цього умисно розроблені.

4. Значні затрати праці програміста. За умов використання файлових систем існувала жорстка залежність між прикладними програмами та даними. Логічна і фізична структура файлу даних повинна була бути повністю ідентичною до тієї, яка описана у прикладній програмі. Навіть зміна фізичного розташування файлу даних (наприклад, новий запам'ятовуючий пристрій) вимагала модифікації прикладної програми. Крім того, зміни в одній прикладній програмі досить часто призводили до необхідності варіювання іншої програми, що була пов'язана з попередньою інформаційно. Все це зумовлювало збільшення вартості супроводжувального програмного забезпечення. Таким чином, затрати на працю програміста зі створення нових та підтримки вже вироблених програм були досить значними (інколи вони сягали 70% вартості попередньо розробленого програмного забезпечення).

Впровадження інформаційних систем на основі БД

Інформаційні системи (ІС) на основі БД, що незабаром з'явилися, стали основним засобом забезпечення точною та своєчасною інформацією. Крім того, ІС на основі БД не мали недоліків традиційних файлових систем.

1. Надмірність та неузгодженість даних. ІС, що використовують БД, дають змогу позбавитися надмірності даних, оскільки всі програми використовують один і той самий набір даних. Важлива інформація (наприклад, ім'я та адреса клієнта) записується в базі лише один раз, що значно заощаджує ресурси. Крім того, змінивши ім'я або адресу клієнта у базі один раз, можна бути впевненим, що всі програми використовують узгоджені дані. Варто зазначити, що певне дублювання даних є в кожній БД. Воно призначене для встановлення зв'язків між окремими файлами, а також прискорення пошуку під час роботи з даними. Проте таке дублювання є мінімальним і обов'язково відоме адміністратору БД. Таке дублювання ще називають "ненадлишковим дублюванням".

2. Недостатній контроль даних. Система керування базами даних здійснює централізований контроль даних та допомагає уникнути непорозумінь,

що породжені синонімами та омонімами. Характерним для інформаційної системи на основі СКБД є те, що дані та їх опис зберігаються разом. Централізований контроль даних здійснюється за допомогою словника, який є частиною такої системи, про що детальніше йтиметься в пункті "Склад інформаційної системи, що використовує БД".

3. Слабкі можливості керування даними. ІС на основі БД були розроблені з урахуванням того, що файли можуть бути пов'язані між собою за різними полями. Це зняло проблему складності керування наявними даними.

4. Робота програміста. ІС, що використовують БД, дозволили розподілити програми та дані, що зробило програми відносно незалежними від деталей визначення даних. Раніше у файлових системах опис даних зберігався безпосередньо у прикладній програмі. Це означає, що прикладна програма містила повний опис усіх полів файлу, незалежно від кількості полів, які вона використовувала у своїй роботі. В інформаційних системах, що використовують БД, прикладна програма потребує опису лише тих полів, які вона безпосередньо використовує. Таким чином було досягнуто значного здешевлення праці програміста.

Висновок: таким чином, інформаційні системи підтримували цілісну, централізовану структуру даних, доступ до яких мала вся організація (навчальний заклад, держустанова тощо). Дані контролювалися за допомогою словника даних, яким керувала група співробітників – адміністратори баз даних. За допомогою нових методик звертання до даних спростився процес зв'язку між елементами даних, що у свою чергу призвело до розширення можливостей роботи з даними. Як наслідок усіх цих змін – спрощення процесу створення програмного забезпечення та подальшої програмної підтримки БД.

Ієрархічна та мережна моделі даних

Сьогодні процес створення потужних інформаційних систем керування базами даних набув максимального розвитку. За декілька десятиліть з'явилися системи, засновані на трьох базових моделях даних: ієрархічній, мережній, реляційній. Модель даних – концептуальний спосіб структурування даних.

Перша ІС, що використовувала БД (початок 60-х років XX ст.), була заснована на ієрархічній моделі. **Модель даних, у якій зв'язки між даними мають вид ієрархії, називається ієрархічною.** Це означає, що і зв'язки між файлами даних мали ієрархічну структуру.

В ієрархічній моделі прийнята така термінологія:

нащадок – підлеглий запис в ієрархії;

предок – підпорядковуючий запис в ієрархії.

В ієрархічній моделі для кожного нащадка може бути лише один предок.

На малюнку 2 зображено частину вже розглядуваної БД деякої фірми із внесеними ієрархічними змінами: замість файлу УГОДИ (де кожна угода була у вигляді одного рядка) маємо файли РАХУНОК та РЯДОК_РАХУНКА. Кожен запис файлу РАХУНОК складається із декількох рядків, відображених у файлі РЯДОК_РАХУНКА. Кожному клієнту може відповідати декілька

рахунків, і кожен рахунок може містити декілька рядків. При цьому кожен рядок визначає продаж одного товару.

КЛІЄНТ

Код_ клієнта	*Прізвище_ клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_ сума_ боргу*	*Виплачено_ на_сьогодні*
100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
2000	Клінтон	Чикаго, 910	США	33 333	30 000

РАХУНОК

Номер_рахунку	*Дата*	*Код_клієнта*	*Код_агента*
100	18.08.05	100	2
200	28.08.05	105	2
310	8.09.05	105	3
440	10.09.05	100	4

РЯДОК_РАХУНКА

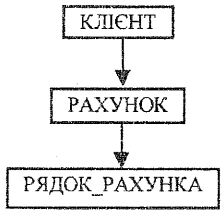
Номер рахунку	*Номер рядка*	*Код товару*	**Кількість**	*Загальна сума*
100	1	3	2	700
100	2	2	2	2 000
200	1	4	3	450
200	2	3	4	1 400
200	3	1	2	240
310	1	3	3	1 050
310	2	2	4	4 000
310	3	1	1	120
440	1	1	2	240
440	2	2	1	1 000

Мал. 2. Дані файлів комерційної фірми з внесеними ієрархічними змінами

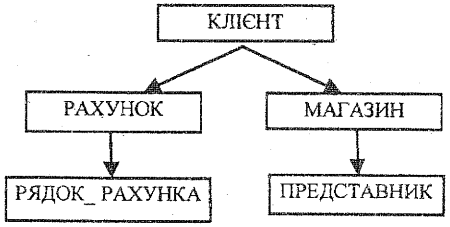
На малюнку 3 зображено, як виглядає ієрархія відношень між клієнтами, рахунками та рядками рахунків. Клієнту підпорядковані рахунки, яким, у свою чергу, підпорядковані рядки.

В ієрархічній базі даних файли зв'язуються між собою фізичними вказівниками. **Вказівник – фізична адреса, що визначає місце зберігання запису на диску.** Кожен запис про клієнта містить у собі вказівник, що визначає перший запис рахунку цього клієнта. У свою чергу, записи рахунків містять

вказівники на записи рядків рахунків. Таким чином, система відшуковує всі записи рахунків та записи рядків рахунків, які стосуються конкретного клієнта. У процесі роботи також може виникнути необхідність мати додаткову інформацію про клієнтів. Наприклад, якщо клієнтами є торгівельні компанії, то виникає потреба знати список усіх магазинів кожної компанії та представника кожного з магазинів. На мал. 4 показано розширену ієрархічну структуру. На обох малюнка відображено ті види зв'язків, які можуть бути реалізованими в ієрархічній моделі.



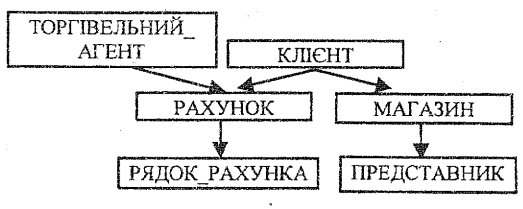
Мал. 3 Схема ієрархічної моделі зв'язків між файлами КЛІЄНТ, РАХУНОК, РЯДОК_РАХУНКА



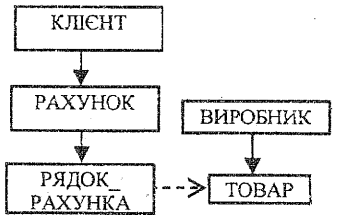
Мал.4 Розширена ієрархічна структура

Проте не всі види зв'язків можна реалізувати за допомогою такої моделі. Наприклад, може виникнути необхідність отримати всі рахунки, що були здійснені певним торговим агентом, для підрахунку загальної суми його комісійних. Нові зв'язки зображені на мал. 5. Така діаграма не є ієрархічною, оскільки нащадок (РАХУНОК) має два предки (ТОРГОВИЙ_АГЕНТ, КЛІЄНТ). Така модель даних є мережною. Очевидна необхідність обробляти такі зв'язки між даними призвела до появи в кінці 60-х мережних систем керування базами даних. Як і в ієрархічних, у мережних моделях для зв'язування файлів використовували фізичні вказівники.

Мережна модель даних реалізує зв'язки між даними тоді, коли кожен запис може бути підпорядкований записам більше ніж з одного файлу. Таким чином, мережна модель – це вдосконалена ієрархічна модель.



Мал. 5. Мережна модель даних



Мал.6. Схема логічного зв'язку, що не підтримується фізичними вказівниками

Реляційна модель даних

Використання фізичних вказівників стало прогресивним напрямком розвитку баз даних, оскільки дозволяло безпосередньо звертатися до даних, які були пов'язані між собою. Проте така модель мала суттєвий недолік: відношення між даними потрібно було визначати до запуску системи. Отримати дані на основі інших зв'язків, що не були наперед визначені, було майже неможливо. Причиною цього є те, що мережна модель базується на фізичних вказівниках, які пов'язують дані з різних файлів. У разі виникнення питання, на яке не можна відповісти, послуговуючись лише тими зв'язками, які відразу були визначені фізичними вказівниками, потрібно було писати окрему (інколи досить істотну) програму. Зрозуміло, що з розвитком інформаційних систем такі обмеження стали неприйнятними.

Ідеальним варіантом є такий: система відповідає на довільне питання, якщо відповідь міститься в даних. Тобто дані повинні бути пов'язані не вказівниками, а мати *логічний* зв'язок.

Для того, щоб краще зрозуміти поняття логічного зв'язку та недоліки систем, основаних на фізичних вказівниках, розглянемо малюнок 6, на якому зображено ієрархію відношень між файлами КЛІЄНТ, РАХУНОК та РЯДОК_РАХУНКІВ. Файли ВИРОБНИК та ТОВАР також ієрархічно пов'язані. Пунктирна лінія позначає *логічний* зв'язок між РЯДКОМ РАХУНКА та ТОВАРОМ, оскільки кожен рядок рахунка відповідає окремому товару. Якщо файли ТОВАР та РЯДОК_РАХУНКА не пов'язані між собою фізичними вказівниками, неможливо дати відповідь на запит – скласти список виробників, товари яких були придбані певним клієнтом. Для того, щоб таки отримати цю інформацію, доведеться використовувати старі методи роботи з файлами, що вимагатиме додаткового програмування.

Ідею логічних зв'язків між даними висунув у 1970 році доктор Едгар Тед Кодд[1], співробітник фірми ІВМ, опублікувавши революційну на той час за змістом статтю, у якій висунув дві прогресивні ідеї.

Перша ідея Кодда: користувач повинен комбінувати дані з різних файлів, якщо логічна інформація, необхідна для такого комбінування, наявна у вихідних даних. Фактично, це була ідея про розмежування фізичного та логічного рівнів збереження даних. Це відкривало нові можливості для розвитку інформаційних систем, оскільки запити до баз даних тепер не обмежувалися фізичними вказівниками. Розглядаючи дані з логічного, а не фізичного погляду, Кодд сформулював другу ідею.

Друга ідея Кодда: в БД весь файл із даними повинен оброблятися однією командою відразу. Такий підхід Кодда надзвичайно підвищив ефективність програмування в базах даних, адже у традиційних на той час схемах за один раз оброблявся лише один запис. Це відкривало нові можливості для

[1] Едгар Тед Кодд – доктор філософії з обчислювальної техніки (Мічиганський університет), народився у Портленді (Англія), помер 2003 році.

розвитку інформаційних систем, оскільки запити до баз даних тепер не обмежувалися фізичними вказівниками.

Для реалізації цих ідей Кодд запропонував просту модель: усі дані були зведені в пов'язані між собою таблиці, що складалися з рядків та стовпчиків. Такі таблиці з даними отримати назву **реляцій**[1], а модель на їх основі – **реляційною**. Для роботи з даними в таблицях Кодд запропонував використовувати дві мови – реляційну алгебру та реляційне числення (про них йтиметься трохи згодом). Обидві мови забезпечують роботу з даними на основі логічних відношень, а не фізичних вказівників. Застосування логічних зв'язків призвело до виникнення понять концептуального проектування та концептуального рівня.

Концептуальний рівень – структурний рівень БД, що визначає її логічну схему.

Концептуальне проектування – створення схеми БД на концептуальному рівні.

Публікації робіт Кодда на початку 70-х та логічних підхід до даних зробив можливим створення мов запитів, які були доступними для звичайних користувачів, що не були спеціалістами в комп'ютерах:

Structured Query Language (SQL) – мова структурних запитів;

Query Language – мова запитів;

Query-by-Example (QBE) – запит за зразком.

У 80-х з інтенсивним розвитком комп'ютерної техніки з'явилися реляційні бази даних (РБД) і для ПК. А в 1986 році мова SQL була прийнята як стандарт мови для РБД. Цей стандарт поновлювався в 1989 та 1992 роках.

На сьогодні РБД розглядаються як основа сучасних інформаційних систем. Файлові, ієрархічні та мережні бази даних також інколи використовують, проте наявна суттєва тенденція до переходу саме на РБД, крім того, відбувається їх вдосконалення.

Розвиток об'єктно орієнтованого програмування призвів до розробки об'єктно орієнтованих СКБД, за допомогою яких можна працювати зі складними об'єктами, що містять взаємозалежні дані, великі описи та мультимедіа. На сьогодні розроблені та користуються попитом такі продукти: O2 (Ardent Software, Inc. www.ardentsoftware.com), Cashe (InterSystems Corporation, www.intersys.com), ObjectStore (Object Design, Inc., www.odi.com), VERSANT (Versant Corporation, www.versant.com), GemStone/S (GemStone Systems, Inc., www.gemstone.com) та ін.

Порівняльна характеристика способів звертання до даних

Спосіб доступу до даних	Характеристика
Файли послідовного доступу	Записи обробляються лише в послідовному порядку (один за одним).

[1] Назва походить від англ. *Relation* – відношення.

Спосіб доступу до даних	Характеристика
Файли довільного доступу	Підтримка доступу до конкретного запису. Складність звертання до декількох записів, що пов'язані з одним.
Ієрархічна база даних	Підтримує доступ до декількох записів, що пов'язані з одним. Зв'язки між даними ієрархічні. Залежить від наперед визначених фізичних вказівників.
Мережна база даних	Підтримує як ієрархічні, так і неієрархічні зв'язки між даними. Залежить від наперед визначених фізичних вказівників.
Реляційна база даних	Підтримує всі логічні зв'язки між даними. Здійснює логічний доступ до даних, що не залежить від фізичної реалізації.

Сучасний напрямок – технологія Клієнт/Сервер

Поява в 1981 році IBM PC зробила настільний ПК звичайним явищем в багатьох установах та організаціях (йдеться, в першу чергу, про США та розвинені країни Європи). Зручно було поєднувати комп'ютери в мережу для спілкування електронною поштою та спільного використання ресурсів, таких, як принтер та диски. На початку сервери були створені саме для друку та доступу до файлів (*сервери друку* та *сервери дисків*). Наприклад, за запитом клієнта сервер відшуковував потрібний файл та надсилав його на клієнтський комп'ютер.

На сьогодні більшість серверів є серверами баз даних.

Сервер бази даних – програма, яка завантажується на сервері та обслуговує доступ клієнтів до баз даних.

Схема роботи наступна:

- клієнт завантажує прикладну програму;
- клієнт звертається до сервера із запитом ;
- сервер виконує запит та повертає результат клієнту.

Сучасна система Клієнт/Сервер – локальна мережа, що містить декілька локальних ПК (клієнти), які обслуговує один спеціальний комп'ютер (сервер).

Клієнти звертаються до сервера за послугами. Наприклад, сервер може надсилати клієнтам програми обробки тексту чи роботи з таблицями; виконувати запити по БД та надсилати результати запитів. Прикладна програма може також посилати на сервер вимогу поновити БД, після чого сервер вносить необхідні зміни. Така система є більш потужною та гнучкою, з нею легше працювати: кінцевий користувач працює з графічним інтерфейсом; ПК є на кожному робочому столі; легко та недорого збільшується потужність системи, шляхом встановлення додаткових комп'ютерів.

В основі продуктивності системи Клієнт/Сервер лежить принцип розподілу праці: кожен комп'ютер виконує свою роботу.

Технологія Клієнт/Сервер – поєднання розподіленої обробки даних і централізованого керування та доступ до даних.

Окрім того, система Клієнт/Сервер забезпечує вирішення питання проблеми колективного доступу до БД у локальній мережі.

До локальних СКБД належать: Microsoft Access, Microsoft Excel (підтримка простих баз даних), FoxPro; Paradox тощо. До клієнт-серверних БД належать Microsoft SQL Server, Inter Base Server, Oracle server.

Хронологія розвитку систем керування базами даних

Рік	
До 1960	Файли послідовного доступу
1960	Файли послідовного та довільного доступу
60-ті	Ієрархічні системи керування базами даних. Мережні системи керування базами даних
1970	Публікація реляційної моделі Кодд
1980	Реляційні бази даних.
1990	Технологія Клієнт/Сервер
1995	Об'єктно-орієнтовані БД

§ 1.2. СПІЛЬНЕ ВИКОРИСТАННЯ ДАНИХ

Суттєвою відмінністю СКБД від файлової системи є можливість спільного використання даних. БД в організації – це сукупність взаємопов'язаних керованих даних, що знаходяться у спільному користуванні.

Користувачі, які перейшли до такої моделі БД, повинні враховувати у своїй роботі, що доступ до даних є колективним. Спільне використання також вимагає змін у способі обробки та утриманні даних. Розглянемо три випадки спільного використання даних:

- різними відділами однієї установи;
- використання на різних рівнях керування;
- географічно віддалені користувачі.

1. Спільне використання даних різними відділами. Співробітники різних відділів однієї установи використовують загальну множину даних, кожен відділ для своїх власних прикладних програм. Об'єднання даних для спільного використання називається *інтеграцією даних*. При цьому зведення всіх даних до однієї бази має синергетичний[1] ефект. Це означає, що цінність зібраних разом даних вища, ніж сума цінностей даних окремих файлів.

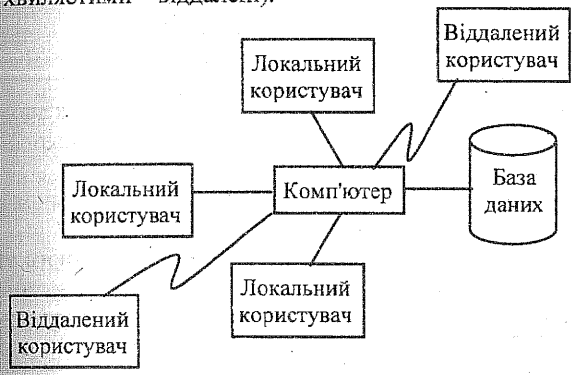
2. Спільне використання даних користувачами різних рівнів. Для спільного використання даних необхідно поділити користувачів за рівнем. Як правило, розрізняють три рівні: звичайні працівники, управлінці середньої ланки, керівники. Ці три рівні відповідають трьом типам автоматичних систем:

[1] Синергізм (від грец. συνεργός – діючий разом) – явище посилення дії одного чинника іншим.

1) електронна обробка даних (ЕОД). Застосовується на найнижчому робочому рівні для автоматизації канцелярської роботи (створення звітів, рахунків за даними бази тощо);
2) інформаційно-керуюча система (ІКС). Використовується для створення запитів та звітів про успішність роботи тих чи інших структур установи тощо;
3) система підтримки прийняття рішень (СППР). Надає стратегічну інформацію керівниками установи для прийняття рішень. На основі аналізу і виявлення довгострокових тенденцій економічного, соціального політичного розвитку робиться висновок про необхідність введення нових спеціальностей у ВНЗ, побудови нового підприємства, припинення випуску того чи іншого товару тощо (залежно від профілю установи).

3. Спільне використання даних регіональними відділеннями. Якщо установа має декілька регіональних відділень, які розташовані на значній території, то використання даних цих відділень може стати проблематичним. Поки дані приходять із певного регіонального відділення та вводяться в базу даних центрального офісу, вони можуть застаріти і прийняти правильне рішення на їх підставі буде важко. Тому з часом виникла необхідність у централізованій БД.

Централізована БД – БД, що фізично зосереджена в одному місці (на одному комп'ютері). З такою базою працювати набагато простіше. Структуру такої бази зображено на малюнку 7 (прямим лініями позначені локальні користувачі, хвилястими – віддалені).

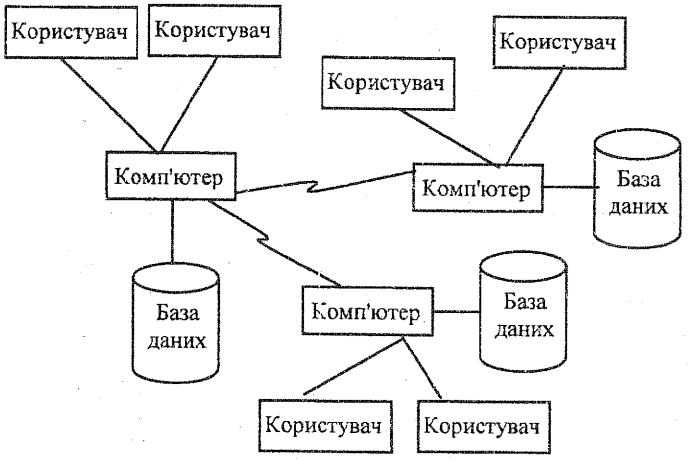


Мал. 7. Структура централізованої БД

Проте якщо організація розвивається і з кожним роком створює все більше філіалів та представництв у різних кінцях світу, то вони можуть мати потребу у своїх власних БД. Розвиток технології комунікацій призвів до появи розподілених систем баз даних.

Розподілена система БД – система БД, що складається з декількох систем локальних БД (кожна керується окремим комп'ютером), які поєднані комунікаційними лініями.

Структуру бази зображено на мал. 8 (хвилястими лініями позначено лінії зв'язку між локальними БД). Розподілені БД роблять можливим локалізацію даних: дані зберігаються в тих вузлах, де до них звертаються найчастіше, проте до них є доступ із будь-якого місця системи.



Мал. 8. Структура розподіленої БД

Недоліки розподілених систем БД:

– необхідність звертання до засобів комунікацій, адже ефективне функціонування розподілених БД вимагає наявності відповідної технології (інколи не завжди дешевої);

– складність керування базою в цілому (проблема, розв'язанням якої повинні займатися проектувальники БД).

Процес поєднання розрізнених даних в одну базу для спільного використання є складним процесом, що вимагає створення бізнес-план установи, визначення інформаційних потреб, проектування плану БД та її створення.

Розподілена СКБД (РСКБД) – програмне забезпечення, що керує розподіленою системою БД.

Важливою складовою діяльності РСКБД є коректна робота із транзакціями (що значущу функцію СКБД буде розглянуто в наступному параграфі).

Транзакція – послідовність операцій над БД, що розглядається СКБД як єдине ціле.

Транзакція, як правило, не контролюється лише одним програмним модулем. Вона може викликати декілька процесів у різних вузлах, що контролюються незалежними програмними модулями.

Кожен з окремих процесів, що взаємодіє всередині транзакції, називається **агентом.**

Локальна транзакція – транзакція, що складається з одного агента.

Глобальна транзакція – транзакція, що складається з декількох агентів.

§ 1.3. СКЛАД ІНФОРМАЦІЙНОЇ СИСТЕМИ, ЩО ВИКОРИСТОВУЄ БД

Повна інформаційна система, що використовує БД, складається з чотирьох компонентів: обладнання, програмного забезпечення, даних, людей. Розглянемо кожну зі складових більш детально.

1. Обладнання

Обладнання – набір фізичних пристроїв, на основі яких існує БД. Як правило, складається з одного або декількох комп'ютерів, дисководів, моніторів, принтерів, кабелю та інших додаткових та з'єднувальних пристроїв.

Успішна робота ІС, що використовують БД, залежить від досконалості обладнання. Для підтримки керування великою кількістю даних потрібен великий обсяг оперативної пам'яті та запам'ятовуючих пристроїв. Крім того, у системі з великою кількістю користувачів потрібен комп'ютер з високою швидкодією (для забезпечення отримання користувачами потрібної інформації у короткий термін). Останнім часом потужність ПК істотно зростає, а вартість знижується, що сприяє широкому застосуванню ІС на основі БД.

2. Дані

БД створюється на основі даних – основних елементів, з яких формується вся інформація. Дані, з яких складається БД, повинні бути логічно організованими: потрібно точно визначити елементи даних та відношення між ними, внести це в словник даних (усе це етапи розробки БД, про що трохи згодом). Лише після цього відповідно до структури в базу можна вносити дані. Усі дані поділяються на фонд даних та архів даних.

Фонд даних – активні дані, з якими постійно працюють прикладні програми та користувачі. Вони зберігаються на вінчестері та перебувають у безпосередньому керуванні СКБД.

Архіви – архівні копії файлів БД. Дані не є активними, зберігаються окремо, згідно із законодавчими актами декілька років (зазвичай на інших ЕОМ) . Використовуються також для відновлення БД у разі її руйнування.

3. Люди

Жодна ІС не може існувати без зовнішнього втручання людини.

Користувачі – люди, яким інформація з БД потрібна для виконання прямих службових обов'язків, що не пов'язані з ПК, програмуванням та СКБД. Це керівники установ, менеджери, завідувачі відділів та ін.

Обслуговуючий персонал – люди, що відповідають за роботу ІС та відповідного прикладного програмного забезпечення. Це адміністратори БД, розробники, аналітики.

4. Програмне забезпечення

Програмне забезпечення є надзвичайно важливою складовою ІС, що використовує БД. Існує два типи програмного забезпечення (ПЗ) інформаційної системи:

- ПЗ для підтримки роботи СКБД;
- ПЗ, що використовує засоби СКБД для виконання конкретних завдань (створення рахунків, аналіз даних тощо).

СКБД – системне програмне забезпечення, що забезпечує керування базою даних.

СКБД виконує вісім основних функцій.

1. Функція СКБД – централізований контроль даних.

Як уже зазначалося, у БД зберігаються не лише самі дані, а й їх опис.

Метадані (метаінформація) – дані, їх описання, інформація про предметну область (об'єкти керування, інформація яких моделюється за допомогою БД та використовується для розв'язування конкретних задач), користувачів системи, статистику роботи, паролі та інші відомості.

Для централізованого зберігання всіх метаданих використовують **словник/каталог даних (СД)**. Він має такі функції:

- СД відстежує визначення всіх елементів БД (включаючи поля, структуру даних на рівні записів, файли або реляційні таблиці); відношення, що існують між різними групами даних; установки формату виведення даних:
- СД підтримує індекси, що прискорюють звертання до даних;
- СД встановлює зв'язки між користувачами;
- СД здійснює централізоване керування даними;
- СД запобігає неузгодженості даних (вирішення проблеми синонімів та омонімів).

СД використовують користувачі (під час роботи із СКБД), програмісти (для написання прикладних програм), системні програмісти (в процесі розширення системи).

2. Функція СКБД – захист даних та підтримка їхньої цілісності.

СКБД захищає дані, забороняючи несанкціонований доступ до них. Повноваження користувачів можуть бути різними (перегляд даних, коригування даних, повна заборона доступу до даних). Такий доступ контролюється системою паролів та представленнями даних.

Представлення даних – опис обмеженої частини БД для конкретного користувача. Користувач, який звертається до БД, вводить наданий йому пароль. Система дає користувачу у відповідності до пароля ті права доступу, що записані в словнику даних.

Цілісність даних – точність та несуперечливість значень даних у базі даних. Цілісність забезпечується:

- обмеженням, що накладаються на значення елементів даних. Це є до певної міри слабкою стороною сучасних СКБД. Опис обмежень на елементи даних зберігаються в словнику даних;
- шляхом створення резервних копій, що забезпечує відтворення знищених даних. Резервні копії та відтворення даних підтримуються програмами, які автоматично фіксують внесені в базу даних зміни та забезпечують відтворення бази даних при збоях системи.

3. Функція СКБД – забезпечення доступу до даних кількох користувачів одночасно.

Одна з головних функцій СКБД – підтримка доступу до БД, добування та поновлення даних бази. Це стосується і віддалених користувачів, які звертаються до БД через систему телекомунікацій. Централізоване збереження інформації є причиною великої ймовірності того, що декільком користувачам одночасно знадобиться одна й та сама інформація. Важливо, щоб СКБД не допускала такої ситуації, коли один користувач звертається до даних, а інший вносить до них зміни. Для цього в СКБД використовуються складні блокувальні механізми, що захищають дані, які наразі коригуються одним із користувачів. У цей час система дає можливість одночасного перегляду даних іншим користувачам.

4. Функція СКБД – керування буферами оперативної пам'яті.

БД займають обсяг набагато більший, ніж обсяг наявної оперативної пам'яті. Тому під час звертання до елементів даних відбувається буферизація даних в оперативній пам'яті. Навіть якщо операційна система робить загальносистемну буферизацію (як у випадку ОС UNIX), цього недостатньо для цілей СКБД. Тому в розвинених СКБД підтримується власний набір буферів оперативної пам'яті з власними правилами та дисципліною заміни буферів.

5. Функція СКБД – керування транзакціями.

Під час зняття грошей із картки відбуваються зміни як на рахунку власника, так і на рахунку банку – здійснюється це в межах однієї транзакції, тому не порушується цілісність БД.

Перед виконанням транзакції їй присвоюють ідентифікатор початкової транзакції і виконують усі необхідні кроки. Інші користувачі не обмежені у виконанні операції пошуку в файлах, що задіяні в транзакції (вони отримують інформацію в незмінному вигляді, доки транзакція не закінчена). Операція редагування цих файлів під час транзакції заборонена.

Транзакція вважається успішною, у разі виконання всіх операцій, що входять до її складу. Якщо виникає помилка хоча б в одній з операцій, уся транзакція вважається помилковою, і результати інших операцій цієї транзакції скасовуються. У разі вдалого виконання транзакції в БД фіксуються відповідні зміни, що були проведені в зовнішній пам'яті. Тепер ці дані стають відомими всім користувачам. В іншому випадку зміни не відображаються в БД. Транзакція є вагомим елементом підтримки логічної цілісності даних. У разі

ж глобальних транзакцій та в умовах доступу до даних багатьох користувачів важливість транзакції неможливо переоцінити.

6. Функція СКБД – журналізація.

Після можливого збою СКБД обов'язково повинна поновити дані. Для забезпечення цього процесу ведеться журнал змін БД. Журнал – частина БД, яка доступна лише адміністратору БД. Часто підтримуються дві копії журналу, що розташовуються на різних фізичних дисках. Для журналізації використовують протокол WAL (Write Ahead Log – запис до реєстрації), за яким запис про зміну будь-якого об'єкта БД повинна потрапити в зовнішню пам'ять журналу до того, як змінений об'єкт потрапить у зовнішню пам'ять основної частини БД.

7. Функція СКБД – створення орієнтованих на користувача запитів та звітів.

Прості мови запитів надають користувачам можливість формулювати запити й отримувати звіти прямо з БД. У такому випадку програмістам не потрібно писати спеціальні прикладні програми з формулюваннями запитів. Мова запитів містить у собі і засоби оформлення звітів за цими запитами.

8. Функція СКБД – підтримка засобів, які орієнтовані на програміста.

Прикладне програмне забезпечення для розв'язання конкретних задач установи може бути написано однією зі стандартних мов програмування (Clipper, Кобол, Сі) або мовою програмування, що входить у комплект самої системи. Сучасні ІС, що використовують БД, укомплектовані власними мовами. Наприклад, СКБД Microsoft Access працює з мовою Visual Basic for Application (VBA), що підвищує ефективність самого процесу програмування.

Зв'язок між компонентами системи

Між чотирма складовими ІС існує взаємозв'язок.

Обслуговуючий персонал (3), проконсультувавшись із *користувачами (3)*, створює структуру бази даних, що відповідає певним вимогам.

Структура БД передається СКБД через *словник (каталог) даних (4)*.

Користувачі (3), підпорядковуючись певним правилам, вводять дані, на основі яких створюються архіви та фонд даних *(2)*.

Збереження та використання введених даних забезпечується *обладнанням (1)*.

Прикладні програми (4) розробляються *програмістами (3)*, а *користувачі (3)* застосовують їх на своїх ПК.

Таким чином, у ефективній ІС усі чотири компоненти – обладнання, ПЗ, дані та люди – взаємодіють, створюючи єдину систему.

§ 1.4. ЖИТТЄВИЙ ЦИКЛ БД

Життєвий цикл БД (ЖЦБД) – процес створення концептуальної (логічної) схеми БД, визначення даних для реалізації схеми, створення програм керування та обробки даних.

ЖЦБД включає збір інформації для потреб користувача, створення логічної структури бази даних (схеми), що відповідає цим потребам; обрання СКБД для підтримки БД; обрання програм для роботою з БД.

Концептуальне проектування БД – визначення елементів даних, відношень між ними та обмеження значень.

Обмеження значень – правило, що визначає допустимі значення кожного з полів елемента.

Фізичне проектування БД – визначення засобів збереження, методів одержання даних та індексів, що використовуються в БД.

ЖЦБД складається з таких етапів:

1) попереднє планування;
2) перевірка можливості виконання такого плану;
3) визначення вимог та завдань;
4) концептуальне проектування;
5) реалізація проекту;
6) оцінювання роботи та підтримка БД.

1. Попереднє планування БД – стратегічна спроба визначити інформаційні потреби на тривалий проміжок часу та майбутні вимоги до створюваної бази даних.

Працівники, які продукують такий проект, повинні мати великий досвід роботи з інформаційними системами, бути спеціалістами в галузі, для якої проектується БД. У процесі планування необхідно визначити:

- кількість прикладних програм, що використовуються на поточний момент; які функції ці програми виконують; наявність документів, інших БД, які виконують ті ж функції, що і майбутня база;
- які файли пов'язані з кожною із існуючих прикладних програм обробки даних;
- які джерела інформації використовуватимуться майбутньою базою даних;
- які нові документи, бази даних знаходяться в процесі розробки, їхня функціональність, джерела цих документів?

Інформація, отримана на цьому етапі, дає змогу передбачити (спланувати) економічні переваги створення та експлуатації БД.

2. Перевірка можливості виконання плану БД. На цьому етапі визначається технологічний, операційний та економічний аспект можливості втілення плану створення БД.

Технологічний аспект.

Потрібно дати відповідь на запитання: чи існує технологія, яка забезпечить реалізацію запланованої бази даних?

Для цього визначають:

- чи існує апаратне та програмне забезпечення, необхідне для функціонування БД;
- чи всі ресурси наявні для реалізації плану (якщо ні, то чи є можливість їх придбати);
- чи потрібне навчання персоналу для роботи з обладнанням та програмами?

Апаратне забезпечення – одна зі складових технологічного аспекту реалізації плану. Це перш за все персональні комп'ютери користувачів та сервери БД. Програмне забезпечення являє собою, насамперед, СКБД, прикладне програмне забезпечення адміністратора БД та інших користувачів.

Операційний аспект.

Потрібно дати відповідь на запитання: чи забезпечена установа персоналом, засобами та експертами, необхідними для успішного втілення плану? Необхідно здійснити аналіз кваліфікації та досвіду спеціалістів, які працюватимуть із майбутньою БД.

Економічний аспект.

Визначення економічної доцільності створення БД – досить складна процедура. Найпростіше визначити затрати на придбання обладнання та програмних засобів. Складніше визначити, як швидко можна отримати економічний ефект від упровадження БД, які непередбачувані ризики можуть виникнути у зв'язку з реалізацією системи (наприклад, під час експлуатація виявилося, що в ній немає потреби, або затрати на створення не виправдали себе). Крім того, легко недооцінити час, потрібний на інтеграцію незалежних прикладних систем. Зміни у програмному забезпеченні можуть також призвести до необхідності незапланованого збільшення потужностей обладнання з метою оптимальної швидкодії. Такі зміни обов'язково призводять до додаткових витрат.

Якщо в результаті перевірки можливості виконання плану виникли невідповідності з вимогами – повторно здійснюється попереднє планування БД. Якщо ж установлено, що передбачені аспекти реалізації плану задовольняють вимоги, то переходять до наступного етапу.

3. Визначення вимог та завдань. На цьому етапі формулюють інформаційні потреби користувачів БД, визначають необхідність створення централізованої БД чи розподільної, окреслюються вимоги відповідно до потреб користувачів різних рівнів. На основі попередніх досліджень установлюються вимоги до апаратного забезпечення.

4. Концептуальне проектування[1]**.** Це етап створення логічної схеми БД. Попередні етапи можна віднести до збору інформації, необхідної для побудови БД. Концептуальне проектування – один із найважливіших етапів, без

[1] Концептуальному проектування та реляційним схемам присвячена друга частина посібника.

якого неможливо здійснити реалізацію БД. Якщо робота установи передбачає обмеження доступу користувачів до БД, то розроблюється концептуальна модель для кожного рівня доступу, після чого встановлюються зв'язки між концептуальними моделями. Узагальнена модель стане основою для реалізації БД. При побудові концептуальної моделі одразу ж необхідно орієнтуватися до типу майбутньої бази даних (ієрархічна, мережна, реляційна тощо). Якщо реалізація буде проходити на основі файлової моделі, то і концептуальну модель будують у вигляді файлової схеми, зв'язку між файлами. Якщо це реляційний тип, то і концептуальну модель бажано будувати у вигляді реляційної схеми.

5. Реалізація проекту. На цьому етапі ЖЦБД здійснюється перетворення концептуальної моделі у фізично-функціонуючу базу даних. Для цього необхідно:

1. Обрати тип бази даних (якщо ще не був обраний на попередніх етапах) та придбати обрану СКБД.

2. Засобами СКБД здійснити перетворення концептуальної моделі в фізичну, реальну базу даних. При цьому кожен елемент концептуальної моделі перетворюється у відповідний елемент реальної бази даних за певними правилами[1].

3. Сформувати словник БД, який є сукупністю визначень структури даних БД. Крім того, словник БД повинен містити інформацію про повноваження доступу, правила захисту даних та контроль даних.

4. Наповнити БД шляхом завантаження даних із файлів, з інших БД або заповнення вручну.

5. Створити прикладні програми.

6. Здійснити навчання та надавати консультації користувачам.

6. Оцінювання роботи та підтримка БД. Після реалізації необхідно провести аналіз дієздатності системи в цілому, визначити, які інформаційні потреби не враховано. У разі можливості та необхідності провести вдосконалення елементів бази. На цьому етапі забезпечується підтримка системи шляхом внесення змін та додавання нових прикладних програм, якщо виникає така необхідність.

ЖЦБД не припиняється на останньому етапі. Внаслідок удосконалення комп'ютерної техніки, удосконалення способів обробки та збереження інформації розроблена БД через певний час застаріває. Це призводить до необхідності вдосконалювати або розроблювати сучасніші бази даних. Тому процес ЖЦБД повторюється знову.

Розділення логічного та фізичного представлення даних.

Як уже зазначалося, доступ до даних розвивався протягом декількох десятиліть від громіздких, фізично орієнтованих методів (файлові системи, фізичні вказівники) до сучасних СКБД. Одним із найбільших важливих аспектів

[1] Перетворення концептуальної моделі в реляційну детально розглянуто у другій частині посібника.

"реляційної революції Кодда" стала ідея відділення логічної структури даних та маніпуляції ними (рівень звичайного користувача) і фізичного представлення (на рівні комп'ютера).

Різницю між логічним та фізичним представленням даних було офіційно визнано в 1978 році, коли було запропоновано узагальнену структуру систем БД, що отримала назву трирівневої архітектури.

Трирівнева архітектура БД – стандартна структура бази даних, що складається з концептуального, зовнішнього та внутрішнього рівнів.

На *концептуальному рівні* виконується концептуальне проектування, результатом якого є концептуальна схема, єдиний логічний опис усіх елементів даних та відношень між ними. Таким чином, концептуальний рівень – структурний рівень БД, що визначає логічну схему БД.

Зовнішній рівень – структурний рівень БД, що визначає представлення даних із позиції кінцевого користувача та програміста. Кожна група користувачів отримує своє власне представлення даних у базі. Кожне таке представлення визначає опис даних та відношень між ними, орієнтовані на конкретного користувача. Сукупність цих користувацьких представлень і складає зовнішній рівень

Внутрішній рівень – структурний рівень БД, пов'язаний із фізичним розміщенням даних у пам'яті ЕОМ (розглядається з погляду системного програміста). На цьому рівні формується фізична модель БД, що включає: структуру зберігання даних у пам'яті ЕОМ; опис форматів записів, порядок їх фізичного або логічного впорядкування, розміщення (фізичні адреси даних), характеристики і шляхи доступу до даних. Жоден користувач не має справи з цим рівнем.

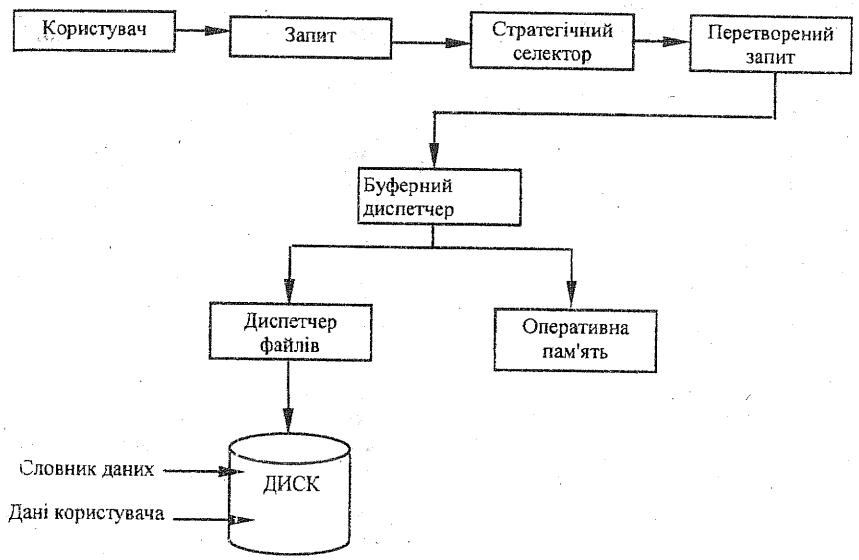
Для представлення даних користувачам на концептуальному або зовнішньому рівнях система повинна вміти перетворювати адреси і вказівники у відповідні логічні імена та відношення. Таке перетворення може відбуватися і в зворотному порядку – з логічного рівня на фізичний. При цьому наявна деяка системна затримка, проте саме це забезпечує незалежність логічного та фізичного представлення даних.

§ 1.5. ФІЗИЧНА ОРГАНІЗАЦІЯ БД

Фізичний доступ до БД

Фізична організація даних – тема в першу чергу для спеціалістів, які мають справу з обладнанням та розробкою системного програмного забезпечення. Простий користувач, як правило, не замислюється над деталями фізичної організації БД, проте ці деталі впливають на швидкодію інформаційної системи, тому є важливими для нього. При вдалому створенні бази дані будуть зберігатися так, щоб їх можна було добувати, оновлювати та маніпулювати ними в максимально короткий проміжок часу.

Схема фізичного доступу до БД зображена на малюнку 9.



Мал. 9 Схема фізичного доступ у до БД

Стратегічний селектор – програмне забезпечення, що перетворює запит користувача у форму, зрозумілу для ПК та ефективну для виконання на ньому.

Буферний диспетчер – програмне забезпечення, що контролює переміщення даних між оперативною пам'яттю та диском.

Диспетчер файлів – програмне забезпечення, що керує розміщенням даних на диску та структурами даних.

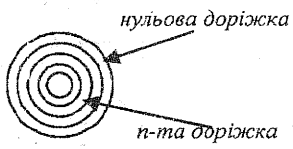
Словник даних – частина СКБД, що визначає структуру даних користувача, правила їх використання, представлення та ін.

Після введення запиту користувачем стратегічний селектор перетворює команду користувача у форму, найбільш ефективну для виконання ПК. Перетворений запит активізує буферний диспетчер, який відшуковує потрібні дані на жорсткому диску та завантажує їх у оперативну пам'ять за допомогою диспетчера файлів.

Фізичні пристрої збереження даних

Жорсткий диск складається з великої кількості нанизаних на вал (стержень) дисків, які покриті з обох боків магнітним матеріалом. Таким чином, усі диски мають дві поверхні. Крім того, кожен диск розподілено на концентричні доріжки, кількість яких може сягати сотень (малюнок 10). Кожна з доріжок у свою чергу поділена на блоки (сектори). Фізичний запис або блок –

найменша одиниця даних. Блок може містити один або декілька логічних записів. Кількість записів у блоці називається **коефіцієнтом блокування**.

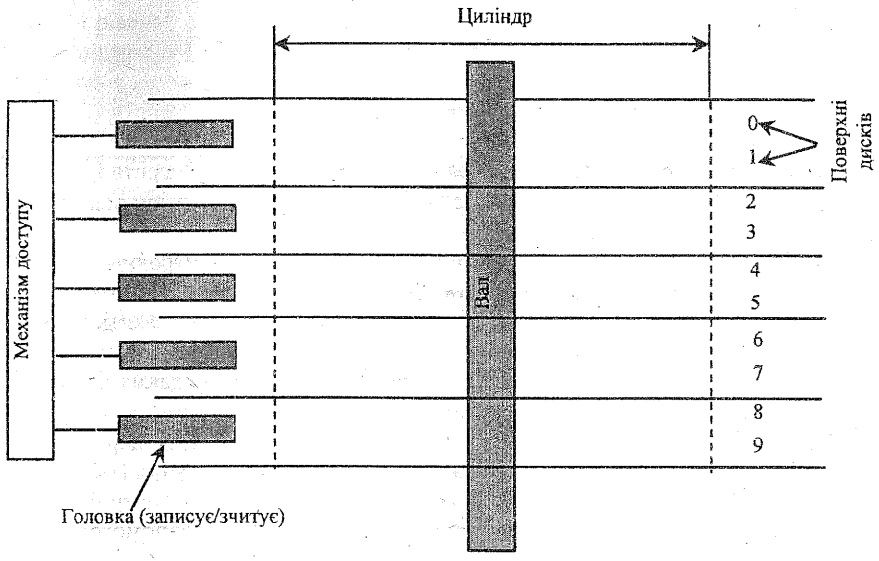


Мал. 10. Поверхня диска

Жорсткий диск розглядається як тривимірна система, тому, крім доріжок та блоків, розглядають ще й циліндри.

Циліндр – сума всіх доріжок, що вертикально збігаються одна з одною по всіх робочих поверхнях.

На малюнку 11 зображено, як відбувається доступ до даних. Декілька магнітних головок (які і зчитують, і записують) переміщуються групою так, щоб бути розташованими на доріжках, що мають один радіус. Отже, розташування цих головок може бути використано для зазначення номера циліндра. Таким чином, на всі доріжки одного циліндра можна записувати (або зчитувати) інформацію без руху головок. Крім того, диски обертаються, і головки мають доступ до всіх секторів того чи іншого диска.



Мал.11. Схема доступу до даних

Повна адреса запису на диску складається з номера циліндра, номера поверхні та номера блоку (сектора). Наприклад, інформація про студента Петренка в базі даних університету міститься на жорсткому диску і має таку фізичну адресу: номер циліндра – 7, номер поверхні – 8, номер блоку 5.

Для того, щоб добути запис, рухомі голівки встановлюються у положення циліндра 7. Потім активізується голівка поверхні з номером 8, яка в проце-

сі обертання дисків зчитує на цій поверхні сектор із номером 5. Коли віднайдено потрібний сектор, усі його логічні записи (якщо їх декілька) завантажуються в оперативну пам'ять, де обирається запис про студента Петренка.

Існують дисководи і з нерухомими голівками. Вони дорожчі, проте мають кращу швидкодію, оскільки в таких пристроях на кожному з наявних циліндрів встановлені фіксовані голівки, а отже, не витрачається час на рух голівок до потрібного циліндра.

Фактори, що впливають на швидкодію обміну даними

Час встановлення голівки (A) – час, необхідний для переміщення голівок на циліндр із певним номером. Цей час ще називають часом пошуку. Перехід від сусіднього циліндра та переміщення через весь диск (від внутрішньої доріжки до зовнішньої) займає різний час. Для підрахунку беруть середній час встановлення голівки, що потрібен для проходу половини всіх циліндрів. Типовий середній час встановлення складає від 8 до 20 мілісекунд (залежно від моделі та виробника дисків).

Час активізації голівки – час, необхідний для електронної активізації голівки, що встановлено на поверхню диска. Порівняно з іншими факторами, які впливають на швидкодію, цей час є настільки мізерним, що ним нехтують.

Затримка обертання – час, необхідний для повороту диска, у результаті якого шуканий блок (сектор) буде знаходитися біля голівки, тобто час, за який потрібний блок переміститься до голівки. Затримка обертання визначається двома факторами: швидкістю обертання диска та розташуванням шуканого блоку відносно голівки. Цей час може змінюватися від нуля (голівка знаходиться на потрібному блоці) до часу повного повороту диска – R. Середній час затримки при підрахунку швидкодії приймають рівним R/2.

Швидкість обміну даними (D) – швидкість зчитування даних із диску в оперативну пам'ять. Цей параметр залежить від швидкості обертання диску та щільності запису даних, вимірюється в тисячах байт за секунду.

Очікуваний час доступу T до диску і зчитування даних визначається за формулою:

$T=A + R/2 + L/D,$

де L – довжина блока в байтах (інші змінні було вже згадано).

Формати зберігання даних на диску

Кожна доріжка має мітку (індексну точку), що визначає початок цієї доріжки. Ця мітка одночасно визначає і кінець доріжки.

Після цієї мітки встановлено власну адресу, за якою можна визначити номер циліндра, номер голівки, що обслуговує цю доріжку, та стан доріжки (робоча чи ушкоджена). У разі дефекту доріжки для використання вказується альтернативна доріжка.

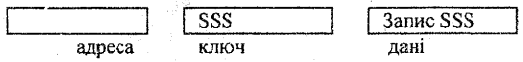
Крім індексної точки та власної адреси, існують окремі області доріжок із даними. Усі ці області розділені зазорами. Довжина зазора залежить від пристрою та від попереднього запису. Довжина зазора після індексної точки

відрізняється від довжини зазора після власної адреси. Довжина зазора після запису залежить від довжини запису. Мета таких відмінностей полягає в тому, щоб час, за який зазор проходить через голівку, відповідав виконуваній у цей час функції обладнання.

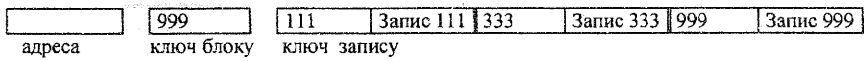
Фізичні записи (або блоки) зберігаються на доріжках в одному з чотирьох форматів (малюнок 12): фіксованої довжини (блоковані та деблоковані) та змінної довжини (блоковані та деблоковані).

Записи фіксованої довжини. Записи мають одну і ту саму довжину. У разі *деблокованого* запису (малюнок 12 а) в кожному фізичному записі (секторі) міститиметься лише один логічний запис (наприклад, запис про одного студента). Якщо запис *блокований* (малюнок 12 а), то кожний фізичний запис (сектор) міститиме декілька логічних записів (наприклад, три записи з відомості про студентів). У цьому випадку коефіцієнт блокування запису дорівнює трьом. У цьому випадку в область ключа вміщується найбільший ключ запису. Це потрібно для швидкого відшукання даних. Наприклад, у сусідніх блоках знаходяться записи 10,15,17 та 18, 20,22. Система шукає запис із номером 20. У ключі першого блоку система знайде значення 17 (оскільки тут зберігається найбільший ключ) і визначить, що запис не може бути в цьому блоці. Ключ наступного блоку 22, отже, шуканий запис буде знаходитися в цьому блоці. Інформація блоку зчитується в оперативну пам'ять, де й буде відшуковуватися потрібний запис (із номером 20).

Фіксована довжина, деблоковані:



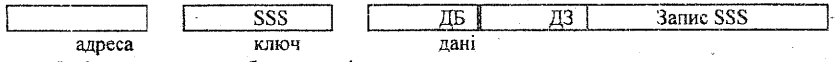
Фіксована довжина, блоковані:



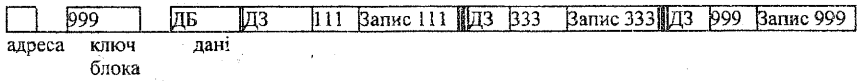
Мал. 12 а. Структура запису фіксованої довжини

Записи змінної довжини. За такого формату записи можуть мати різну довжину. Оскільки довжини записів неоднакові, то виникає необхідність у зазначенні кінця запису. Для цього існують дві області: ДБ – довжина блока та ДЗ – довжина запису

Змінна довжина, деблоковані:



Змінна довжина, блоковані:



Мал. 12 б. Структура записів змінної довжини

Організація доступу до даних та способи адресації.

Існують три основні способи фізичної організації файлів: послідовний, індексно-послівний та прямий. Існують й інші способи, проте вони є модифікаціями трьох названих.

Спосіб збереження даних тісно пов'язаний зі способом доступу до даних.

Послідовна організація файлів. У стандартній реалізації записи впорядковано за зростанням значення ключа. Доступ до записів лише послідовний. Для прочитання десятого запису потрібно прочитати попередні 9. У сучасних умовах використання його незначне.

Індексно-послідовна організація файлів. Зазвичай записи впорядковано за зростанням значення ключа. Такий спосіб дає змогу прикладним програмам звертатися до запису послідовно, а запитам користувача напряму. Кожен циліндр має індекс – максимальний ключ запису, що зберігаються на ньому. Перегляд індексу циліндра дає змогу визначити, на якому із циліндрів розташовано запис. Так само кожна з доріжок має індекс (ключ), перегляд яких дає змогу визначити блок, де знаходиться потрібний запис.

У такому випадку послідовний пошук не повністю виключається, проте його межі значною мірою зменшуються.

Пряма організація файлів. Такий доступ ще називається хешируваним. Можливий лише прямий доступ. Використання техніки хешування позбавляє від необхідності підтримувати та проглядати індекси. А відмова від індексів позбавляє від необхідності двічі звертатися до запам'ятовуючого пристрою (один раз для читання індексу, другий – для доступу до запису).

Технологія (алгоритм) хешування полягає у процедурі визначення адреси запису, спираючись на деяке поле цього запису (як правило, ключа). Такі алгоритми ще називають хеш-функціями.

Розглянемо приклад. На диск, який дає змогу зберігати у блоці до 2000 байт, потрібно записати 500 записів (наприклад, з відомості про стипендію), кожен із яких має довжину 100 байт. Якщо розподілити потрібний обсяг інформації (500*100=50000 байт) по всіх допустимих місцях, то для збереження такої інформації потрібно буде 25 блоків (50000:2000=25). Для того, щоб алгоритм не обчислював одну і ту саму адресу для різних записів (ситуація називається конфліктною або колізією), загальну кількість блоків збільшують приблизно на 20 відсотків. Це – обов'язкова перестраховка. У нашому випадку відведемо 30 блоків з метою зменшення кількості конфліктів.

Відношення дійсного обсягу файла до обсягу пам'яті, що відводиться для його запису, називається **коефіцієнтом навантаження.** У нашому випадку він дорівнює 83 % (25:30=83), тобто навантаження зменшується.

Рівномірний розподіл записів по блоках означає, що приблизно на кожну з 5 можливих адрес припаде 4 записи. Припустимо, що наша послідовність блоків для записів починатиметься з адреси 1003 та закінчуватиметься адресою 1036. Це припущення не є принциповим. Деякий запис має ключ 1562. Розглянемо тепер покроково один із можливих алгоритмів хешування (визначення адреси за ключовим полем).

1. Ділимо значення ключа на зарезервовану кількість блоків. 1562:30=52, 2 в остачі.

2. Остача від ділення – *відносна адреса блоку*. Відносна адреса блока запису = 2. Взяття остачі гарантує, що обчислена адреса завжди міститься у проміжку від 0 до 29 (оскільки остачу отримуємо від ділення на 30).

3. Додаємо результат кроку 1 до адреси початкового блока, отримуючи *абсолютну адресу блока*, де зберігатиметься запис.

У нашому випадку запис із ключем 1562 міститиметься за адресою 1005 (2+1003=1005). Залежно від того, блоковані чи деблоковані записи ми розглядаємо, у цьому блоці або розміщуються ще записи (аж поки блок не заповниться), або записування за цією адресою припиняється.

Проте для запису із ключем 1592 також матимемо остачу 2 за розглянутим алгоритмом. Тобто в нас виникла конфліктна ситуація. Один із способів вирішення колізії полягає в уточненні методу ділення, який називається *методом квадратів ділених*. При такому методі запам'ятовуються значення остачі R (у нашому випадку –2) та значення частки Q (у нашому випадку – 52). Використовуючи значення R та Q, розглядають послідовність можливих відносних адрес, що задані формулою:

$R+Q\cdot i^2+i$ (за модулем кількості блоків), де i – пробігає цілі значення від 0 доти, доки конфлікт не буде вичерпано.

Ми вже визначили для запису з ключем 1592 відносну адресу. Вона дорівнює 2. Проте ця відносна адреса вже зайнята, тому ми повторюємо обчислення при наступному значенні i доти, доки не буде знайдено наступну вільну відносну адресу. У нашому випадку при i=1

$(2+53\cdot 1^2+1)$ за модулем 30 =26. Таким чином, ми знайшли вільну відносну адресу. Додавши її до початкової адреси, відшукаємо абсолютну адресу для запису з ключем 1592.

Якщо необхідно на диску знайти запис із ключем 1592, алгоритм працює як звичайно, відшукавши спочатку запис із ключем 1562. Проте, переконавшись, що це не той запис, який потрібен, виконується наступна інтерпретація, доки не буде знайдено відповідний запис.

Розглянутий випадок є досить простим прикладом статичної хеш-функції. При збільшенні записів у БД збільшується кількість конфліктів, і, як наслідок, – збільшується час доступу до записів. Для вирішення цієї проблеми використовують складніші хеш-функцій – динамічні.

ЧАСТИНА 2
КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ

§ 2.1 ПРИНЦИПИ КОНЦЕПТУАЛЬНОГО ПРОЕКТУВАННЯ БД

Концептуальне проектування[1] БД є одним із найважливіших етапів ЖЦБД. Його метою є визначення складу й структури майбутньої бази та створення інформаційно-логічної моделі. Основними конструктивними елементами концептуальної моделі є об'єкти, їх атрибути та відношення між ними. Об'єкти найчастіше представлені іменниками, властивості – прикметниками та іменниками, а відношення – дієсловами.

Реальність і моделі

Процес концептуального (або інфологічного) проектування вимагає з'ясування інформаційних вимог користувачів та представлення їх у правильно сформульовані моделі. **Модель – представлення реальності, що відображає лише окремі значущі деталі.** Наведемо декілька ілюстрацій.

Приклад 1. Клієнт вносить деяку суму на свій рахунок у банку. При цьому для банківської установи важливі наступні деталі: номер рахунку, сума вкладу, процентні ставки, прізвище клієнта. Інші деталі є незначними, наприклад, кількість відвідувачів банку, наявність чи відсутність черги, погодні умови, колір волосся клієнта тощо. Реальність має безліч деталей, проте банк цікавлять лише ті, що мають безпосереднє відношення до вкладу клієнта.

Проте деякі деталі, що були несуттєвими для одних користувачів, можуть бути надзвичайно важливими для інших.

Приклад 2. Менеджера ресторану швидкого харчування надзвичайно цікавлять такі подробиці, як погода (оскільки в холодний день продається зовсім не те, що у спекотний), довжина черги (на основі цієї інформації визначатимуть необхідну кількість додаткових працівників). Несуттєвими є вік відвідувачів, їхні зовнішні ознаки.

Проте для салону краси або перукарні вік, колір волосся та інші зовнішні характеристики можуть виявитися вагомими.

Об'єкти

Об'єкти – предмети (або істоти); які користувачі вважають важливими в модельованій частині реальності.

[1] Концептуальне проектування іноді ще називають інфологічним.

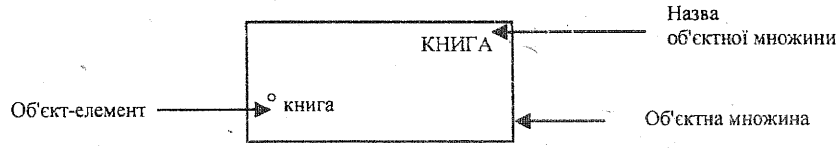
Наприклад, об'єктами можуть бути люди, автомобілі, книги, комп'ютери. Проте потрібно розмежовувати конкретну річ (книгу чи автомобіль) від множини речей (усіх книг або всіх автомобілів).

Об'єктна множина – множина речей одного типу.

У схемах об'єктні множини позначають прямокутником, а назву пишуть великими літерами (малюнок 13).

Множина може мати один елемент, декілька, безліч елементів або бути порожньою (не містити жодного елемента).

Об'єкт-елемент – конкретний елемент об'єктної множини. У схемах його позначають у вигляді кружечка, а назву пишуть малими літерами (малюнок 13).



Мал. 13. Об'єктна множина та об'єкт-елемент

При написанні сукупність елементів множини заключають у фігурні дужки.

Об'єктні множини можуть бути лексичними та абстрактними.

Лексична об'єктна множина – об'єктна множина, що складається з елементів, які можна роздрукувати.

Абстрактна об'єктна множина – об'єктна множина, що складається з елементів, які не можна роздрукувати.

Наприклад, ім'я може бути лексичною об'єктною множиною, оскільки елементами такої множини є рядки символів, які можна роздрукувати. Так само номер страховки або номер паспорта, дата тощо.

Проте люди є абстрактною об'єктною множиною. Навіть якщо людину можна ідентифікувати за номером паспорта, проте цей номер і людина не одне й те саме.

У комп'ютерній реалізації концептуальної моделі елементи лексичних об'єктних множин будуть представлені у вигляді рядків символів. Елементи абстрактних об'єктних множин будуть представлені внутрішніми номерами, що не мають ніякого смислу поза системою. Такий номер ще називають **сурогатним ключем**, оскільки він однозначно визначає абстрактний елемент-об'єкт реального світу.

Наприклад, Петро Петренко – елемент об'єктної множини ЛЮДИНА. У комп'ютерній реалізації цей об'єкт-елемент може позначатися деяким номером 1234567812. Усі дані на цю людину (повне ім'я, адреса, телефон, номер рахунка банку, номер медичного полісу та ін.) будуть записані у вигляді лек-

сичних даних, що пов'язані у БД сурогатним ключем. Користувач буде бачити лише лексичні дані і ніколи не зустрінеться з числом 1234567812. Проте система буде використовувати цей сурогатний ключ, що асоціюється з елементом Петренко.

Конкретизація та узагальнення

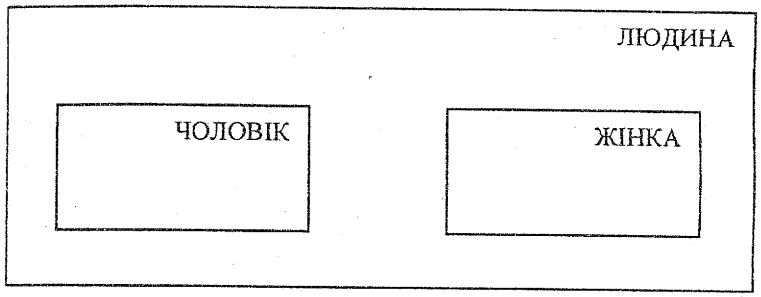
Деякі об'єктні множини містяться всередині інших об'єктних множин. Наприклад, множина МЕРСЕДЕС міститься всередині множини ХОРОШІ АВТОМОБІЛІ, яка, у свою чергу, міститься в множині АВТОМОБІЛІ. Множини ЧОЛОВІК та ЖІНКА містяться всередині множини ЛЮДИНА.

Конкретизація – об'єктна множина, що є підмножиною іншої об'єктної множини.

Узагальнення – об'єктна множина, що є надмножиною іншої об'єктної множини.

Схематичне представлення конкретизації та узагальнення зображено на мал. 14.

Інколи на складних схемах конкретизацію позначають не включенням прямокутника у прямокутник, а спеціальною лінією.



Мал. 14. Конкретизація та узагальнення

Відношення

Зв'язок між елементами двох об'єктних множин називається **відношенням**. Відношення між множинами схематично зображають ромбом, відношення між об'єкт-елементами – лініями. Розглянемо множини ОДРУЖЕНИЙ ЧОЛОВІК та ЗАМІЖНЯ ЖІНКА. Встановимо між цими множинами відношення ПЕРЕБУВАЄ У ШЛЮБІ, поставивши у відповідність кожному жонатому чоловіку його дружину (малюнок 15 а).

Відношення саме по собі є об'єктною множиною, яка складається з пар об'єктів-елементів, що взяті з двох множин, поєднаних відношенням. Наприклад:

ОДРУЖЕНИЙ ЧОЛОВІК = {Сергій, Олександр, Іван}

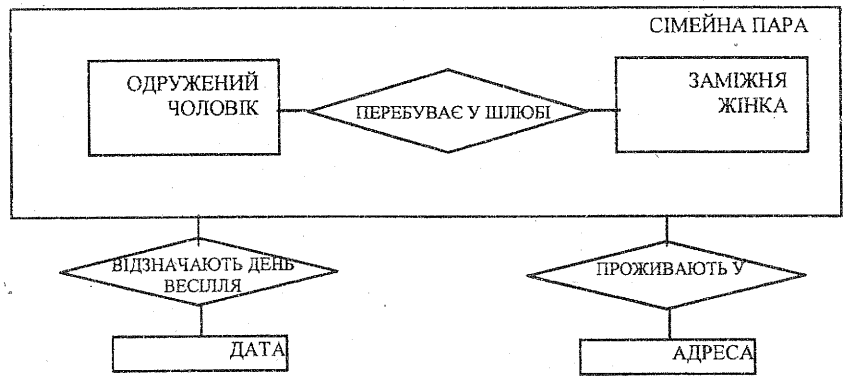
ЗАМІЖНЯ ЖІНКА = {Ірина, Марина, Світлана}.

Схематично ці відношення зображені на малюнку 15 б).

а) відношення ПЕРЕБУВАЄ У ШЛЮБІ

Сергій — Ірина
Олександр — Марина
Іван — Світлана

б) деякі приклади відношення ПЕРЕБУВАЄ У ШЛЮБІ



в) Складена об'єктна множина СІМЕЙНА ПАРА , що бере участь у відношеннях

Мал. 15. Схеми зодображення відношень

Складені об'єкти

Складена об'єктна множина (складений об'єкт) – це відношення, що розглядається як об'єктна множина.

Таким чином, множина СІМЕЙНА ПАРА, отримана з відношень між іншими множинами є складеною об'єктною множиною. Записують це так:

СІМЕЙНА ПАРА = {(Сергій, Ірина), (Олександр, Марина), (Іван, Світлана)}.

Складеним об'єктним множинам дають імена, їх також можна включати в інші відношення.

Такі відношення зображені на малюнку 15 в.

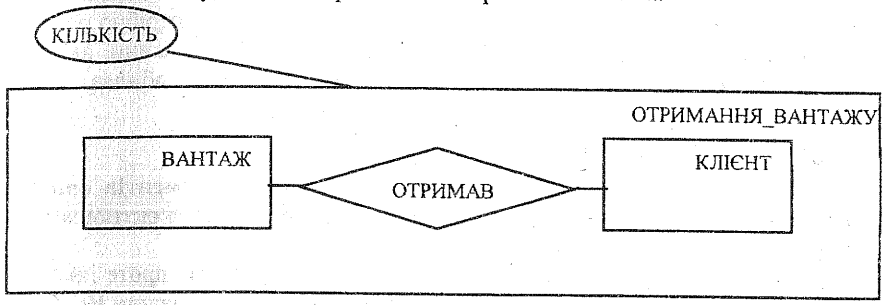
Відношення між двома об'єктними множинами називається *бінарним*.

Відношення між більше, ніж двома об'єктними множинами, називаються *відношенням високого порядку*.

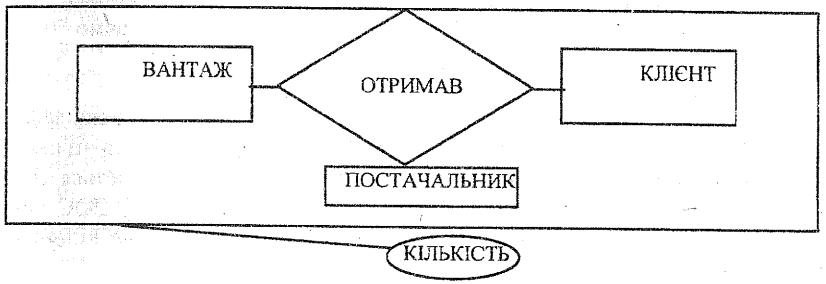
Взагалі, довільне відношення високого порядку можна розбити на послідовність вкладених бінарних відношень. Проте не всі вони мають суттєве значення, тому деталізують їх лише за потребою. У загальному ж випадку використовують відношення n-го порядку.

Розглянемо ще один приклад складеного відношення: об'єктну множину ВАНТАЖ та об'єктну множину КЛІЄНТ, між якими встановлено відношення ОТРИМАВ. Складена об'єктна множина (малюнок 16 а) матиме назву ОТРИМАННЯ_ВАНТАЖУ. Для визначання кількості певного ВАНТАЖУ, який отримав конкретний КЛІЄНТ (кількість літрів виноградного соку, яку отримав клієнт Сомов), установимо для об'єктної складеної множини ОТРИМАННЯ_ВАНТАЖУ атрибут КІЛЬКІСТЬ.

Звернемо увагу, що цей атрибут притаманний не об'єктній множині ВАНТАЖ і не об'єктній множині КЛІЄНТ, а складеній об'єктній множині ОТРИМАННЯ_ВАНТАЖУ, оскільки атрибут КІЛЬКІСТЬ залежить як від кількості вантажу, так і від прізвища конкретного клієнта.



Мал. 16. а. Складена об'єктна множина ОТРИМАННЯ_ВАНТАЖУ



Мал. 16. б. Складена об'єктна множина ОТРИМАННЯ_ВАНТАЖУ

Моделювання концептуальних та фізичних об'єктів

Розглянемо складнішу ситуацію: потрібно визначити кількість літрів води (соку, пива), яку отримав клієнт Петренко (Сидоренко, Іванов) від постачальника Жукова (Сидоренка, Сомова). Для цього необхідно встановити відношення між складеним об'єктом ОТРИМАННЯ_ВАНТАЖУ та об'єктною

множиною ПОСТАЧАЛЬНИК. Новому відношенню (відношенню високого порядку) надати атрибут КІЛЬКІСТЬ (малюнок 16 б).

Атрибут КІЛЬКІСТЬ тепер залежить від об'єктних множин КЛІЄНТ, ВАНТАЖ та ПОСТАЧАЛЬНИК одночасно.

Важливо відзначити, що в розглядуваному прикладі об'єктна множина ВАНТАЖ являє собою більше концептуальний, аніж фізичний об'єкт. Кожен об'єкт цієї множини задає лише тип вантажу (вода, консерви, соки), а не конкретну пляшку з водою чи соком. Приклад конкретного вантажу: "вода у пляшках, яку отримав клієнт Петренко від постачальника Сидоренка".

Концептуальний об'єкт – об'єкт, що визначає тип речей.

Концептуальна об'єктна множина – об'єктна множина, елементами якої є абстрактні поняття.

Фізичний об'єкт – об'єкт, що визначає реальний фізичний об'єкт.

Фізична об'єктна множина – об'єктна множина, елементами якої є фізичні предмети.

Існують випадки, коли при проектуванні БД обов'язково потрібно розрізняти концептуальні об'єктні множини та фізичні об'єктні множини, які їм відповідають (якщо виявиться необхідним зазначати у моделі обидва типи об'єктів).

Потужність відношень

Потужність відношення – максимальна кількість елементів однієї об'єктної множини, що пов'язані з одним елементом іншої об'єктної множини.

Як правило, інтерес представляє максимальна потужність, проте інколи корисною буває і мінімальна потужність. Якщо розглядати множини ЧОЛОВІК та ЖІНКА і відношення між ними ПЕРЕБУВАЄ У ШЛЮБІ, то мінімальна потужність дорівнює 0 (оскільки є неодружені чоловіки та жінки), а максимальна потужність дорівнює 1. Біля об'єктної множини зазначаємо "0, 1" – кожен чоловік може мати від 0 до 1 дружини.

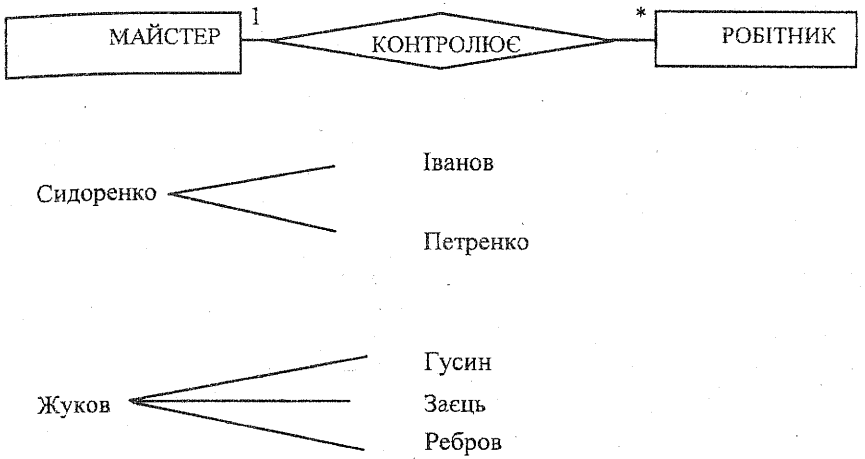
Деякі відношення не мають конкретної потужності.

Розглянемо приклад. Є дві множини працівників одного з цехів заводу: МАЙСТЕР та РОБІТНИК. Множина РОБІТНИК складається із працівників, які не контролюють роботу інших. Множина МАЙСТЕР складається із тих працівників, які контролюють робітників. Відношення КОНТРОЛЮЄ пов'язує кожного майстра з робітниками, яких він контролює. Схема відношень зображена на малюнку 17.

Наприклад, майстер контролює як мінімум одного робітника, а можливо, й більше. Таку потужність позначають "1, *".

Порівняємо два наведені приклади. Кожен елемент множини ОДРУЖЕНИЙ ЧОЛОВІК пов'язаний лише з одним елементом множини ЗАМІЖНЯ ЖІНКА. У відношенні КОНТРОЛЮЄ одному елементу множини МАЙСТЕР відповідає декілька елементів множини РОБІТНИК.

Відношення, яке має максимальну потужність в одному із напрямків, що дорівнює одиниці, називається **функціональним** у цьому напрямку. Відношення між майстром та робітниками є функціональним в одному напрямку з напрямків (від робітника до майстра). Тобто, знаючи прізвище робітника, ми можемо однозначно визначити його майстра. У зворотному напрямку відношення не є функціональним.



Мал. 17. Зображення відношення КОТРОЛЮЄ

Відношення один-до-одного – відношення, максимальна потужність якого дорівнює одиниці в обох напрямках.

Відношення один-до-багатьох – відношення, максимальна потужність якого дорівнює одиниці в одному напрямку і багатьом у зворотному напрямку.

Відношення багато-до-багатьох – відношення, максимальна потужність якого дорівнює багатьом в обох напрямках.

Приклад відношення багато-до-багатьох: студент відвідує лекції з багатьох предметів, лекції з предмета відвідує багато студентів.

Атрибути

Як правило, елементи об'єктних множин мають деякі ознаки, за якими ці об'єкти розрізняються. Наприклад, у кожної людини з об'єктної множини ЛЮДИНА є власне прізвище, ім'я, дата народження, стать, вага, колір волосся тощо. У кожного автомобіля з множини АВТОМОБІЛІ є власний номер, марка, колір, об'єм циліндра, дата випуску, величина пробігу та ін.

Під час побудови концептуальної моделі потрібно визначити, які з ознак об'єкта є суттєвими.

Атрибути – важливі характеристики елементів об'єктних множин, за якими ці об'єкти розрізняються. На схемі їх зображають у вигляді еліпсів.

Атрибути є функціональними відношеннями у напрямку від об'єкта до атрибута. Це означає, що значення атрибута однозначно встановлено для кожного елемента. У кожної особи лише одна дата народження (хоч у цей день народилося багато людей), а також лише одне прізвище (хоч існують однофамільці).

Якщо для певного елемента об'єктної множини значення котрогось із його атрибутів не визначено, то вважають, що цей атрибут має порожнє значення.

Атрибути об'єкта можуть змінюватися, а об'єкт, який вони описують, залишається тим самим. У певної особи може змінитися вага, зріст, місце проживання, проте це буде та сама людина.

Якщо об'єкт є конкретизацією іншого об'єкта, то конкретизований об'єкт наслідує всі атрибути і відношення узагальненого об'єкта.

Розглянемо приклад **наслідування атрибутів**. Об'єктна множина ОДРУЖЕНИЙ ЧОЛОВІК є конкретизацією множини ЧОЛОВІКИ. В одруженого чоловіка є прізвище, ім'я, дата народження, номер паспорта тощо. Об'єкт-елемент одружений чоловік наслідує ці атрибути від об'єкта-елемента чоловік. Проте в конкретизованого об'єкта є свої власні атрибути: в одруженого чоловіка ними можуть бути ім'я дружини, дата святкування весілля тощо.

Проілюструємо **наслідування відношень**. Нехай об'єкт-елемент множини ЧОЛОВІК пов'язаний із множиною КОМПАНІЯ відношенням ПРАЦЮЄ. ОДРУЖЕНИЙ ЧОЛОВІК (будучи конкретизацією множини ЧОЛОВІК) також пов'язаний з об'єктом КОМПАНІЯ тим самим відношенням ПРАЦЮЄ.

Наслідування атрибутів та відношень є надзвичайно важливим, оскільки воно дозволяє більш точно моделювати реальність, визначаючи підмножини об'єктних множин, які мають свої власні атрибути й відношення та зберігають усі атрибути й відношення надмножин.

Ключі

Під час побудови концептуальної моделі важливо знайти такі атрибути, значення яких є постійними, оскільки саме їх можна використати у ролі ключів. **Ключ – значення, яке однозначно визначає елемент об'єктної множини.**

У пункті "Об'єкти" цього параграфа згадувався сурогатний ключ. Реалізація концептуальної БД передбачає, що кожна людина з об'єктної множини ЛЮДИНА отримує свій сурогатний ключ для розрізнення її всередині бази. Проте, оскільки сурогатні ключі не можуть бути використані поза системою, користувачам БД потрібні й інші способи ідентифікації елементів.

Ідентифікатор – лексичний атрибут або набір лексичних атрибутів, значення яких завжди однозначно визначають елемент об'єктної множини.

Не варто обирати в ролі ідентифікатора номер паспорта (оскільки не всі люди мають паспорт, не всі пам'ятають його номер).

Інколи для створення ідентифікатора необхідне використання декількох атрибутів. У такому випадку ключем буде комбінація обраних атрибутів. Наприклад, будуючи БД для ветеринарної клініки можемо стикнутися з такою ситуацією: не всі тварини мають паспорт, за однією адресою може перебувати декілька тварин. За подібних обставин ідентифікатор буде складним.

Побудова концептуальної моделі

З'ясувавши на етапі попереднього планування інформаційні потреби організації на тривалий проміжок часу та майбутні вимоги до створюваної бази даних, можна перейти до побудови логічної схеми (концептуальної моделі), яка є основою інформаційної системи.

Для побудови схеми виконують такі дії:

- визначають об'єктні множини, які використовуватимуться в БД (у разі потреби зазначають узагальнення та конкртизації);
- описують усі атрибути кожної з об'єктних множин;
- обирають ключовий атрибут (або їх сукупність) для кожної з об'єктних множин;
- встановлюють відношення між об'єктними множинами, зазначаючи їх потужності.

Нерідко доводиться будувати логічні моделі БД на основі звітів, які отримані в результаті роботи. Наприклад, організація, яка планує створити БД, у результаті своєї діяльності щомісяця звітує про обіг коштів, обсяг виконаних робіт, використання тих чи інших ресурсів тощо. Проаналізувавши наявні звітні документи установи, можна на їх основі створювати концептуальну модель майбутньої БД. Проте такий шлях використовується для моделей, які застосовують для систем обробки даних.

§ 2.2. РЕЛЯЦІЙНА МОДЕЛЬ БД

З 1970 року відбулося поступове впровадження реляційної моделі. Як уже згадувалося, всі існуючі підходи до зв'язування записів із різних файлів використовували фізичні вказівники. Кодд довів, що такі БД істотно обмежують маніпуляцію даними. Крім того, вони залежать від фізичного оточення: у разі встановлення додаткового дисковода чи зміни адреси збереження даних потрібно додатково змінювати файли даних (змінювати вказівники). Якщо до формату запису у файлі додавалися нові поля, то фізичні адреси всіх записів файла змінювалися. Таким чином, можливості програмного забезпечення та користувачів були досить обмеженими та не дозволяли вільно маніпулювати даними.

Реляційна модель даних, яка грунтувалася на логічних зв'язках між даними, вирішила ці проблеми. Кінцевого користувача більше не цікавила фізична структура даних.

До проектування реляційної бази даних існує два підходи.

Перший підхід протягом значного проміжку часу був традиційним. Він передбачає, що на етапі проектування створюється не концептуальна модель даних, а безпосередньо розробляються таблиці даних (подібні до таблиць на малюнку 1). Проектування реляційної бази даних у такому випадку зводиться до нормалізації таблиць відповідно до стандартної процедури (процес нормалізації буде розглянуто в цьому параграфі трохи згодом).

2. Другий підхід базується на побудові концептуальної моделі даних, що створюється на етапі концептуального проектування. Далі ця модель за певними правилами перетворюється в реляційну. Дотримання цих правил гарантує отримання в кінцевому результаті унормованої реляційної моделі.

До широкого розповсюдження концептуальних моделей традиційно використовувався перший підхід. Він і зараз є корисним при створенні невеликої БД. У таких випадках виводять визначення реляційних таблиць безпосередньо з БД, а потім їх нормалізують. Другий підхід використовують при проектуванні складних БД, які використовуються в інформаційних системах.

Основні поняття реляційної моделі.

Реляційна модель даних організує та представляє дані у вигляді реляцій.

Реляції – двовимірні таблиці (з рядків та стовпчиків даних), що пов'язані між собою.

Прикладом реляції може бути будь-яка таблиця, зображена на малюнку 1.

Таблиця КЛІЄНТ представляє об'єктну множину КЛІЄНТ та її атрибути. Кожен стовпчик реляції – **реляційний атрибут**. Назва стовпчика – **ім'я атрибута**.

Відношення, що пов'язує об'єктну множину із самою собою, називається **рекурсивним відношенням**.

Кількість атрибутів реляції визначає **ступінь реляції**.

Кортеж – рядок реляції.

Певного, наперед визначеного порядку кортежів не існує. Ніякі два кортежі не мають однакового набору значень атрибутів.

Для запису реляції після імені зазначаються її атрибути, обмежені фігурними дужками. Для реляції КЛІЄНТ, що зображена на малюнку 1, загальне позначення буде таким:

КЛІЄНТ {Код_клієнта, Прізвище_клієнта, Адреса_клієнта, Країна, Загальна_сума_боргу, Виплачено_на_сьогодні}

Набір усіх можливих значень, яких може набувати атрибут, називається **областю атрибута**.

Дві області атрибутів збігаються, коли вони мають один і той самий смисл. Наприклад, атрибути *Адреса_клієнта* та *Країна* мають різні області, хоча в обох із них значенням кожного є рядок символів. Проте якщо у двох атрибутах, які мають різні назви (наприклад, *Професія* та *Спеціальність*), містяться назви різних спеціальностей, то області таких атрибутів збігаються.

Зрозуміло, що два атрибути з однаковою областю не обов'язково матимуть тотожні імена

Порожні значення

Існують випадки, коли визначений для множини атрибут не застосовується. Наприклад, для об'єктної множини СТУДЕНТИ атрибут *Вагітність* є незастосовним до осіб чоловічої статі. Також потрібно зауважити, що інколи при введенні даних котрийсь із атрибутів може бути тимчасово невідомим. В обох випадках ми нічого не вводимо і кортеж записується у БД із порожнім значенням таких атрибутів. Не варто плутати порожнє значення з пропуском чи нулем.

Порожнє значення – значення, що приписується атрибуту в кортежі, якщо атрибут незастосовний або його значення невідоме.

Ключі

Ми вже зверталися до поняття ключа при обговоренні концептуальної моделі (розглядали ключі, ідентифікатори та сурогатні ключі).

Звернімося знову до малюнка 1. Вважатимемо, що кожного клієнта в реляції представлено лише одним записом. Якщо якийсь атрибут однозначно визначає клієнта, ми можемо стверджувати, що значення цього атрибута однозначно визначає кортеж реляції. Припустимо, що атрибут *Код_клієнта* однозначно визначає клієнта. Тоді значення атрибута *Код_клієнта* однозначно визначає рядок таблиці КЛІЄНТ. Отже, атрибут *Код_клієнта* є ключем таблиці КЛІЄНТ.

Суперключ – довільний набір атрибутів, що однозначно визначає кожен кортеж реляції.

Ключ – мінімальний набір атрибутів, що однозначно визначає кожен кортеж реляції (мінімальний суперключ).

Проілюструємо ці два поняття на прикладі реляції КЛІЄНТ (малюнок 1).

Набір атрибутів (*Код_клієнта, Прізвище_клієнта*) однозначно визначає довільний кортеж реляції. Отже, такий набір є суперключем реляції КЛІЄНТ. Проте, цей набір не є мінімальним, тому він не є ключем. Окремо взятий атрибут *Код_клієнта* є ключем, оскільки кожен кортеж однозначно визначається значенням цього атрибута.

Складений ключ – ключ, що містить два і більше атрибутів.

У довільній реляції залежно від обставин, можна виявити більше, ніж один набір атрибутів, що можна обрати в ролі ключа.

Потенційний ключ – довільний набір атрибутів, що можуть бути обрані в ролі ключа.

Первинний ключ – ключ, обраний для переважного використання в ролі однозначного визначення кортежу.

Як правило, в ролі первинного ключа обирають такий потенційний ключ, яким найлегше користуватися у повсякденній роботі.

Зазвичай, первинний ключ називають просто ключем. З урахуванням поняття ключа загальне позначення реляції КЛІЄНТ буде таким:

КЛІЄНТ {Код_клієнта, Прізвище_клієнта, Адреса_клієнта, Країна, Загальна_сума_боргу, Виплачено_на_сьогодні}.

Розглянемо реляцію УГОДИ (малюнок 1).

Для угоди ключем може бути набір атрибутів *Дата* та *Код клієнта.*, оскільки один і той самий клієнт не може одночасно здійснити дві угоди[1].

Зовнішні ключі

Досить часто імена атрибутів використовуються у різних реляціях. Наприклад, атрибут *Код_товару* знаходиться в реляціях ТОВАР та УГОДИ. Атрибут *Код_клієнта* віднаходимо в реляціях КЛІЄНТ та УГОДИ. Такі атрибути ілюструють поняття зовнішнього ключа.

Зовнішній ключ – набір атрибутів однієї реляції, що є ключем іншої (або цієї ж самої) реляції. Використовується для логічного зв'язування даних однієї реляції з даними іншої.

Наприклад, *Код_товару* з реляції УГОДИ однозначно визначає товар з реляції ТОВАР. Якщо, розглядаючи якусь угоду (визначили її суму, дату укладення), потрібно дізнатися, який саме товар було реалізовано, то, переглянувши реляцію ТОВАР, за кодом товару можна однозначно визначити назву товару. Отже, атрибут *Код_товару* реляції УГОДИ є зовнішнім ключем реляції ТОВАР.

Рекурсивний зовнішній ключ – зовнішній ключ, що посилається на власну реляційну таблицю.

Пригадаємо розглянутий раніше приклад: майстер контролює робітника (малюнок 18). Розглянемо реляцію з можливої БД для підприємства, на якому працюють ці робітники (майстер також є робітником фабрики).

ПРАЦІВНИК			
Код_ працівника	*Прізвище_ працівника*	*Посада*	*Хто_ контролює*
10	Петренко	Робітник	11
11	Сидоренко	Майстер	15
12	Іванов	Робітник	11
13	Заєць	Робітник	16
14	Гусин	Робітник	16
15	Клінтон	Начальник цеху	
16	Жуков	Майстер	15
17	Ребров	Робітник	16
18	Чайковська	Секретар	15

Мал. 18. Приклад реляції ПРАЦІВНИК

У цьому випадку для реляції ПРАЦІВНИК атрибути *Код_працівника* і *Хто_ контролює* мають однакову область, проте різні імена.

Атрибут *Хто_контролює* є зовнішнім ключем реляції ПРАЦІВНИК, що посилається на ключ власної таблиці. Для кожного працівника атрибут

[1] При зазначенні дати у БД фіксується й час.

Хто_контролює однозначно визначає контролера, який, у свою чергу, є також працівником фабрики.

Інформація про зовнішні ключі є надзвичайно важливою. Тому зазначення зовнішніх ключів обов'язкове одразу після запису реляції:

ПРАЦІВНИК {Код_працівника, Прізвище_працівника, Посада, Хто_контролює}

Зовнішній ключ: *Хто контролює* посилається на ПРАЦІВНИК *(Код_працівника)*

Реляційна схема бази даних – список, у якому записано імена реляцій з зазначеними атрибутами для кожної (ключі підкреслено) та встановлено зовнішні ключі.

Така схема є попереднім результатом створення ЖЦБД.

Обмежувальні умови, що підтримують цілісність даних

Обмежувальні умови забезпечують логічну основу для підтримки правильних значень даних у БД, запобігаючи введення помилок під час оновлення чи обробки даних.

Обмежувальна умова – правило, що обмежує значення даних у БД.

Такі правила є надзвичайно цінними, оскільки одна з найважливіших функцій БД – забезпечувати точну інформацію для прийняття рішень.

У реляційній моделі Кодда передбачено декілька обмежувальних умов, що використовуються для перевірки даних у БД: категорійна цілісність, цілісність на рівні посилань, функціональні залежності.

Категорійна цілісність.

Кожен кортеж реляції представляє в базі даних елементи конкретних реальних об'єктів. Наприклад, кортеж реляції КЛІЄНТ представляє конкретну людину-покупця. Ключ реляційної таблиці однозначно визначає кожен запис (рядок, кортеж). Якщо користувач схоче отримати дані конкретного запису, він повинен знати значення ключа. Отже, при заповненні реляційних таблиць даними спочатку потрібно вводити значення ключового атрибута. Таким чином, значення ключа (або його частини) не може бути порожнім. У цьому й полягає правило категорійної цілісності.

Правило категорійної цілісності: ніякий ключовий атрибут кортежа не може бути порожнім.

Цілісність на рівні посилань.

При побудові реляцій для зв'язування кортежів використовуються зовнішні ключі. Дуже важливо, щоб значення атрибуту, який є зовнішнім ключем, у кожному рядку відповідало якомусь із значень атрибута, на який відбувається посилання.

Якщо в реляції ТОВАР є коди товарів 1, 2, 3, 4, то лише такі значення можуть бути в атрибута, що є зовнішнім ключем (*Код_товару* реляції УГОДИ).

Якщо в реляції ПРАЦІВНИК (малюнок 18) є коди працівників 10-18, то лише такі значення можуть бути в атрибута, що є зовнішнім ключем (*Хто_контролює* цієї ж реляції).

Звернемо увагу, що зовнішній ключ може бути й порожнім (начальника Клінтона ніхто не контролює, тому значення атрибута в цьому кортежі порожнє).

Правило цілісності на рівні посилань – значення кожного зовнішнього ключа повинно бути або порожнім, або дорівнювати одному зі значень ключа іншої таблиці.

Функціональні залежності.

Функціональні залежності дозволяють накладати додаткові обмеження на реляційну схему. У таблиці КЛІЄНТ (малюнок 1) атрибут *Код_клієнта* однозначно визначає атрибут *Прізвище_клієнта*. Атрибут *Прізвище_клієнта* однозначно визначає адресу клієнта та країну. Хоч Україна віднайдена двічі, проте для клієнта з кодом 100, де б він не був зафіксований в інших таблицях, країною завжди буде Україна.

У таблиці ПРАЦІВНИК (малюнок 18) (атрибут *Код_працівника* однозначно визначає атрибути *Прізвище_працівника* та *Посаду*.

Функціональна залежність – значення атрибута в кортежі однозначно визначає значення іншого атрибута кортежу.

Записується функціональна залежність так:

ФЗ: *Код клієнта* → *Прізвище клієнта*

ФЗ: *Прізвище клієнта* → *Адреса клієнта*

Знак → читається "функціонально визначає".

Запис A→B означає: якщо два кортежі реляції мають однакове значення атрибута A, то вони мають одне й те саме значення атрибута B. Таке формулювання застосовне й тоді, коли A та B є не поодинокими стовпчиками, а групою стовпчиків.

Детермінант – атрибут у лівій частині виразу (до стрілки), що визначає значення інших атрибутів кортежа.

Складений детермінант – детермінант, що містить декілька атрибутів

Ключ таблиці є детермінантом, оскільки його значення однозначно визначає значення кожного атрибута таблиці.

Повна функціональна залежність – атрибут B знаходиться у повній функціональній залежності від складеного атрибута A, якщо він функціонально залежить від A і не залежить функціонально від будь-якої підмножини атрибута A.

§ 2.3. ПРОЦЕС НОРМАЛІЗАЦІЇ

Кінцевою метою нормалізації реляцій є створення проекту БД, у якому кожен факт з'являється лише в одному місці бази. У такому випадку не лише

економиться пам'ять, а й виключена можливість суперечливості збережених даних.

Нормалізація – процес приведення реляцій до стандартного виду.

Уже зазначалося, що до проектування реляційної бази даних існує два підходи.

Перший підхід передбачає, що на етапі концептуального проектування створюється не концептуальна модель даних, а безпосередньо реляційна схема даних із конкретних реляцій, які в подальшому потрібно нормалізувати.

З метою виявлення проблем, які можуть виникнути через нескрупульозне проектування БД, розглянемо приклади реляцій, які є далеко не бездоганними з погляду нормалізації.

Звернемося до завдання 5 практичної роботи № 3. Нехай існує таблиця з даними про проходження студентами різних видів практик (малюнок 19). Зрозуміло, що це – частина великої реляції, і кількість студентів у ній значно більша, та інформація про них повніша (факультет, спеціальність, адреса тощо), ніж та, що зображена на малюнку. Обмежимося розглядом частини такої реляції.

ПРАКТИКА					
Код_студента	*Прізвище_студента*	*Вид_практики*	*Хто_контролює_(код)*	*Дата_початку*	*Дата_кінця*
10	Іванов	агробіостанція		12.06.05	19.06.05
10	Іванов	педагогічна	11	20.06.05	27.06.05
10	Іванов	гуртожиток	12	1.07.05	7.07.05
11	Петренко	педагогічна		20.06.05	27.06.05
11	Петренко	корпуси	14	10.07.05	17.07.05
12	Сидоренко	педагогічна	11	1.07.05	7.07.05
12	Сидоренко	гуртожиток		20.06.05	27.06.05
13	Кравченко	агробіостанція	10	12.06.05	19.06.05
14	Сомов	агробіостанція	10	12.06.05	19.06.05
14	Сомов	педагогічна	11	20.06.05	27.06.05
14	Сомов	корпуси		10.07.05	17.07.05

Мал. 19. Частина реляції з відомостями про проходження студентами практики

Проаналізуємо наведену таблицю, визначивши наявні недоліки.

1. У кортежах відбувається повторення інформації (прізвища студентів, назви практик, дати). Ця надлишковість даних призводить не лише до втрати вільного місця, але й може викликати порушення цілісності БД.

Наприклад, в одному із кортежів при введенні прізвища Іванов буде допущено помилку (Іванова). Тоді між кортежами, що містять інформацію про одну й ту саму особу, виникне невідповідність. При введенні дат практик, які повторюються, також можна помилитися. Отже, між кортежами знову виникне невідповідність. Така невідповідність називається аномалією оновлення.

Аномалія оновлення – неузгодженість даних, що викликана їхньою надлишковістю та частим оновленням.

2. У разі, якщо студентів захворів і не відбував практику (наприклад, звільнений), дані про проходження ним практики видаляють. Разом із цим будуть утрачені й дані про самого студента.

Аномалія видалення – непередбачувана втрата даних через видалення інших даних.

3. Розглянемо таку ситуацію: з іншого ВНЗ перевівся студент, відомості якого ще не внесено у БД. Проте він повинен відпрацювати практику замість студента, який був призначений керівником групи, але захворів. Заповнити поле *Хто_контролює* для студентів, які внесені до таблиці і яких має контролювати цей новоприбулий, неможливо (оскільки коду цього студента в таблиці ще просто не існує).

Аномалія введення – неможливість ввести дані в таблицю, що викликано відсутністю інших даних.

Зрозуміло, що аномалії оновлення, видалення та введення є дуже небажаними. Для вирішення цих проблем таблицю поділяють на декілька. Результат поділу зображено на малюнку 21.

Розбиття таблиць – процес поділу таблиць на декілька з метою усунення аномалій та підтримки цілісності даних.

Для поділу таблиць існують **нормальні форми** – правила структурування таблиць.

Перша нормальна форма

Введемо поняття атомарного значення атрибута.

Атомарне значення – значення атрибута, що не є множиною значень.

Розглянемо таблицю (малюнок 20), що не відповідає першій нормальній формі. У процесі створення реляційної схеми на основі даних про практику (без попередньої побудови концептуальної моделі) така таблиця теж могла мати місце.

ПРАКТИКА

Код_ студента	*Прізвище_ студента*	*Вид_практики*	*Кого_ контролює*	*Дата_ початку*	*Дата_ кінця*
10	Іванов	агробіостанція	{13,14}	12.06.05	19.06.05
10	Іванов	педагогічна		20.06.05	27.06.05
10	Іванов	гуртожиток		1.07.05	7.07.05
11	Петренко	педагогічна	{10, 12, 14}	20.06.05	27.06.05
11	Петренко	корпуси		10.07.05	17.07.05
12	Сидоренко	педагогічна		20.06.05	27.06.05
12	Сидоренко	гуртожиток	10	20.06.05	27.06.05
13	Кравченко	агробіостанція		12.06.05	19.06.05
14	Сомов	агробіостанція		12.06.05	19.06.05
14	Сомов	педагогічна		20.06.05	27.06.05
14	Сомов	корпуси	11	10.07.05	17.07.05

Мал. 20. Частина реляції з відомостями про проходження студентами практики, що не відповідає 1НФ.

У деяких кортежах значення атрибута *Кого_контролює* є множиною значень (контролює декілька осіб). Збільшення кількості студентів у БД призведе і до збільшення множинних значень атрибута (адже на малюнку відображено лише частину реляції).

Перша нормальна форма (1НФ) – значення всіх атрибутів повинні бути атомарними.

Таким чином, таблиця, зображена на малюнку 20, не перебуває в 1НФ, оскільки значення атрибута *Кого_контролює* для деяких кортежів не є атомарними.

За принципами реляційної моделі Кодда, усі реляції повинні знаходитися в 1НФ. Чи перебуває в 1НФ таблиця, що зображена на малюнку 19? Безумовно, так.

Друга нормальна форма.

Друга та третя нормальні форми стосуються відношень між ключовими та неключовими атрибутами. Зазначимо також, що кожна наступна нормальна форма передбачає виконання попередньої.

Друга нормальна форма (2НФ) – неключові атрибути не є функціонально залежними від частини ключа.

Одразу потрібно зазначити: якщо реляція у ролі ключа має лише один атрибут, то вона перебуває у 2НФ.

Розглянемо таблицю, зображену на малюнку 19. Вона задовольняє 1НФ.

Чи перебуває вона у 2НФ? Ключ цієї таблиці складається з двох атрибутів: *Код_студента* та *Вид_практики*. Поодинці кожен із цих атрибутів не є ключем, а лише одночасно вони однозначно визначають довільний кортеж реляції. Проте неключовий атрибут *Прізвище* функціонально залежить лише від атрибута *Код_студента*, який є частиною ключа. Так само кожен із неключових атрибутів *Дата_початку* та *Дата_кінця* функціонально залежать від частини ключа, а саме від атрибуту *Вид_практики*.

Отже, розглядувана таблиця не відповідає 2НФ. Залишивши таблицю без змін, можемо стикнутися з аномаліями оновлення, видалення, введення.

Для приведення цієї таблиці до 2НФ її потрібно розбити на декілька таких, що задовольняли б 2НФ. Запишемо реляційну схему цих створених таблиць:

СТУДЕНТИ {Код_студента, Прізвище_студента}
ВИД ПРАКТИКИ {Код_практики, Вид_практики, Дата_початку, Дата_кінця}
ПРАКТИКА {Код_студента, Код_практики, Хто_контролює}
Зовнішній ключ: *Код_студента* посилається на ПРАКТИКА
Код_практики посилається на ВИД ПРАКТИКИ

Таблиці, побудовані на основі такої реляційної схеми, зображено на малюнку 21.

У кожній із цих реляцій жоден неключовий атрибут не є функціонально залежним від частини ключа. У перших двох таблицях уже тільки тому, що ключ складається з одного атрибута.

У третій таблиці також дотримана 2НФ. Проте тут існує надмірність даних, тому в разі потреби можна продовжувати розбиття.

СТУДЕНТИ

Код_ студента	*Прізвище_ студента*
10	Іванов
11	Петренко
12	Сидоренко
13	Кравченко
14	Сомов

ВИД_ПРАКТИКИ

Код_ практики	*Вид_ практики*	*Дата_ початку*	*Дата_ кінця*
1	агробіостанція	12.06.05	19.06.05
2	педагогічна	20.06.05	27.06.05
3	гуртожиток	1.07.05	7.07.05
4	корпуси	10.07.05	17.07.05

ПРАКТИКА

Код_студента	*Код_практики*	*Хто_контролює*
10	1	
10	2	11
10	3	12
11	2	
11	4	14
12	2	11
12	3	
13	1	10
14	1	10
14	2	11
14	4	

Мал. 21. Проекції таблиць, що отримані в результаті розбиття.

Таблиці, зображені на малюнку 21, називаються проекціями початкової таблиці.

Проекція таблиці – таблиця, що складається з декількох обраних атрибутів іншої таблиці.

Зрозуміло, що кількість кортежів проекції зменшується порівняно з початковою реляцією (таким чином позбуваємося зайвих даних).

Виокремимо декілька простих кроків у процесі розбиття (декомпозиції) таблиці на декілька таких, що відповідають 2НФ:

1. Створюється нова таблиця, атрибутами якої будуть атрибути початкової таблиці, які входять у ФЗ, що не задовольняє 2НФ. Детермінант цієї функціональної залежності обирають за ключ нової.

2. Атрибути, що знаходилися у правій частині ФЗ, виключають із початкової таблиці.

3. Якщо у таблиці більше ніж одна ФЗ, що порушує 2НФ, то перші два кроки повторюють для кожної з них.

4. Якщо один і той же детермінант наявний у декількох ФЗ, то всі атрибути, які залежать від нього функціонально, вміщують у ролі неключових атрибутів у таблицю, ключем якої буде детермінант (на малюнку 21 – таблиця ВИД_ПРАКТИКИ, що отримана з таблиці, зображеної на малюнку 19).

Третя нормальна форма

Третя нормальна форма – довільний детермінант є ключем.

Це означає, що для будь-якої ФЗ А→В у реляції А є ключем.

Зазначимо також, що тоді, коли реляція відповідає 3НФ, то вона автоматично відповідає і 2НФ. Проте зворотне твердження не є правильним.

Проілюструємо це на прикладі таблиць на малюнку 21. Ці таблиці були отримані з початкової (малюнок 19) шляхом розбиття, і кожна з них відповідає 2НФ.

Крім того, реляції СТУДЕНТИ та ПРАКТИКА задовольняють і 3НФ.

А от реляція ВИД_ПРАКТИКИ не відповідає 3НФ. Оскільки у ФЗ *Вид_практики→ Дата початку* детермінант *Вид_практики* не є ключем. Аналогічна ситуація і з ФЗ *Дата_початку →Дата_кінця*.

Транзитивна залежність – залежність, у якій неключовий атрибут функціонально залежить від одного або декількох неключових атрибутів. Інакше кажучи, це випадок, коли у функціональному відношенні не бере участі ключ.

На цьому прикладі переконалися: якщо реляція перебуває у 3НФ, то вона автоматично перебуває і в 2НФ (реляції СТУДЕНТИ та ПРАКТИКА). Зворотне твердження неправильне (реляція ВИД_ПРАКТИКИ).

Найпростіший метод перетворення таблиці, що порушує 3НФ, у набір таблиць, що задовольняють 3НФ – розбиття. Для цього виконаємо наступні кроки:

1. Видалити із реляції ВИД_ПРАКТИКИ всі атрибути, які містяться у правій частині функціональних відношень, що порушують 3НФ. Це два відношення: *Вид практики→Дата початку* та *Вид практики→Дата кінця*.

Початкова таблиця матиме позначення

ВИД_ПРАКТИКИ {Код практики, Вид практики}

2. Створимо нову реляцію ПОЧАТОК_ПРАКТИКИ на основі видалених атрибутів *Вид_практики* та *Дата_початку*, причому одразу замінимо атрибут *Вид_практики* на ключовий атрибут *Код_практики* таблиці ВИД_ПРАКТИКИ .

ПОЧАТОК_ПРАКТИКИ {Код_практики, Дата_початку}

Зовнішній ключ: *Код_практики* посилається на ВИД_ПРАКТИКИ.

3. Аналогічно створимо і реляцію КІНЕЦЬ_ПРАКТИКИ:

КІНЕЦЬ_ПРАКТИКИ {Код_практики, Дата_кінця}

Зовнішній ключ: *Код_практики* посилається на ВИД_ПРАКТИКИ.

Наведений приклад демонструє, що зайве розбиття зовсім не поліпшує проекту БД. Тому теоретики РБД Кодд і Бойс обґрунтували й запропонували визначення для 3НФ, яке враховує, що в таблиці може бути кілька можливих ключів.

Таблиця перебуває в нормальній формі Бойса-Кодда (НФБК), якщо будь-яка функціональна залежність між її полями зводиться до повної функціональної залежності від *можливого* ключа.

НФБК є більш строгою версією 3НФ. Якщо таблиця перебуває у НФБК, то вона перебуває і у 3НФ. Зворотне не вірне.

Четверта нормальна форма

1НФ забороняє реляціям мати неатомарні (багатозначні) атрибути . Проте існує досить багато ситуацій, що вимагають саме багатозначних атрибутів. Один із таких прикладів уже було розглянуто (малюнок 20).

Можна навести ще декілька прикладів: один доцент викладає декілька предметів, викладач задіяний у декількох екзаменаційних комісіях, одна людина має декілька ощадних рахунків в одному банку.

У таких випадках потрібно моделювати таку БД, у якій відсутні багатозначні атрибути. Розглянемо такий приклад. Викладач N приймає екзамен з геометрії та з алгебри на різних курсах.

Прізвище	Предмет	Курс
N	{Алгебра, Геометрія}	{I, II, III}

У такому випадку не очевидно, що атрибути Предмет та Курс не залежать один від одного. Може виникнути питання: чи не залежить якимось чином предмет, який викладається, від курсу? Для того, щоб позбавитися уявних зв'язків між атрибутами, ставиться вимога: кожне значення атрибута повинно поєднуватися з кожним значенням іншого атрибута принаймні в одному рядку. Проілюструємо це:

1.

Прізвище	Предмет	Курс
N	Алгебра	I
N	Алгебра	II
N	Геометрія	II
N	Геометрія	III

Багатозначна залежність (БЗ) – для кожного значення атрибута А існує певна кількість відповідних атрибутів В.

БЗ, яка забезпечує незалежність атрибутів шляхом обов'язкового повторення значень, називається багатозначною залежністю.

Оскільки багатозначні залежності потребують великої кількості повторень, важливим етапом нормалізації полягає в позбавленні від багатозначних залежностей.

Таблиця має четверту нормальну форму (4НФ), якщо вона має 3НФ і не містить багатозначних залежностей. Оскільки проблема багатозначних залежностей виникає у зв'язку із багатозначними атрибутам, то вирішити її можна, помістивши кожен багатозначний атрибут в окрему таблицю разом із ключем, від якого цей атрибут залежить. Наприклад:

Предмет

Прізвище	Предмет
N	Алгебра
N	Геометрія

Курс

Прізвище	Курс
N	I
N	II
N	III

Атрибут Прізвище – це ключ деякої іншої таблиці, що ідентифікує викладача.

Правила Кодда для РБД

Правило інформації. Уся інформація БД зберігається в таблицях.

Правило гарантованого доступу. Доступ до інформації в таблицях гарантований при зазначенні імені таблиці, атрибута у таблиці та значення ключа.

Правило підтримки недійсних значень. У ролі відсутніх значень БД може використовувати пусті значення. Ніякі значення ключових атрибутів не можуть бути пустими.

Правило оновлення, додавання, видалення. Реалізуються на рівні записів таблиці.

Правило незалежності фізичних даних. У разі зміни способу зберігання даних та доступу до них на рівні фізичної організації логічна структура БД не змінюється.

Правило цілісності даних. РБД повинна підтримувати можливість визначення цілісності даних.

§ 2.4. ПЕРЕТВОРЕННЯ КОНЦЕПТУАЛЬНОЇ МОДЕЛІ В РЕЛЯЦІЙНУ

Концептуальні моделі забезпечують найбільш точне й наочне представлення складних аспектів прикладної проблеми (ситуації з життя). Проте на сьогодні не існує систем, що реалізують концептуальні моделі, які задані графічно. Тому концептуальну модель необхідно спочатку перевести в таку модель, що може бути реалізованою. Такою моделлю є реляційна.

Концептуальна модель даних складається з об'єктів, відношень, атрибутів, конкретизацій, узагальнень, складених об'єктів. Розглянемо методи перетворення кожної з цих конструкцій у реляції. Важливим результатом цього процесу є те, що отримані реляції відповідатимуть усім вимогам нормалізації.

Перетворення об'єктних множин та атрибутів

Розглянемо найпростішу концептуальну модель. Нелексична абстрактна множина СТУДЕНТ має два лексичних атрибути *Прізвище* та *Дата_народження*.

Атрибути реляції повинні бути лексичними, тому атрибути концептуальної моделі можуть бути атрибутами реляції. Отже, реляційна схема буде наступною:

СТУДЕНТ {Прізвище, Дата_народження}

Якщо припустити, що прізвища студентів не повторюються, то у ролі ключа можна обрати атрибут *Прізвище*. Проте це малоймовірно, тому в таких

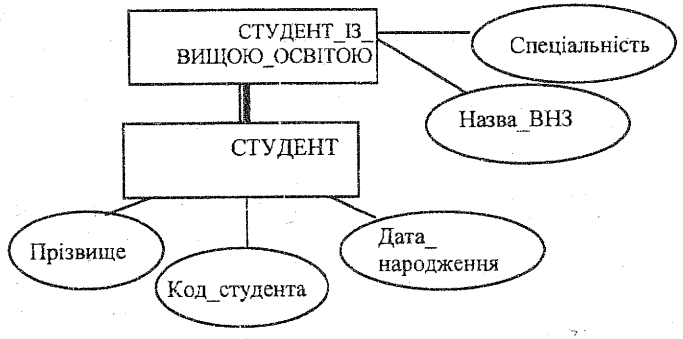
випадках модель доповнюють додатковим ключовим атрибутом *Код_студента*. Отже, кінцева реляційна схем буде такою:

СТУДЕНТ {Код студента, Прізвище, Дата_народження}.

Мал. 22. Об'єктна множина СТУДЕНТ та її атрибути

Перетворення конкретизацій та узагальнень об'єктних множин

Розглядаючи попередній приклад, можна допустити, що частина студентів здобуває другу вищу освіту.



Мал. 23. Конкретизація й узагальнення та їх спільні й індивідуальні атрибути.

Множина СТУДЕНТ – узагальнення, множина СТУДЕНТ_ІЗ_ВИЩОЮ_ОСВІТОЮ – конкретизація.

Зрозуміло, що для множини СТУДЕНТ реляція залишається тією самою:

СТУДЕНТ {Код студента, Прізвище, Дата народження}

Оскільки множина СТУДЕНТ_ІЗ_ВИЩОЮ_ОСВІТОЮ є підмножиною множини СТУДЕНТ, то вона наслідує всі атрибути множини СТУДЕНТ. Крім того, у неї є свої власні атрибути. Тому реляційна схема конкретизації буде такою:

СТУДЕНТ_ІЗ_ВИЩОЮ_ОСВІТОЮ {Код_студента, Прізвище, Дата_народження, Назва_ВНЗ, Спеціальність}

Зовнішній ключ: *Код студента* посилається на СТУДЕНТ.

Важливо зазначити, що ключ конкретизації є також і її зовнішнім ключем, що вказує на таблицю узагальнення, адже кожен елемент конкретизації повинен знаходитися і в узагальненні з тим самим номером. Таким чином, *Прізвище* та *Дата_народження* в реляції СТУДЕНТ_ІЗ_ВИЩОЮ_ОСВІТОЮ містить інформацію, що повторюється. Для того, щоб позбавитися даних, які повторюються, ми видалимо з таблиці конкретизації неключові атрибути, що повторюються. Таким чином, кінцева реляційна схема матиме вигляд:

СТУДЕНТ {Код студента, Прізвище, Дата народження}

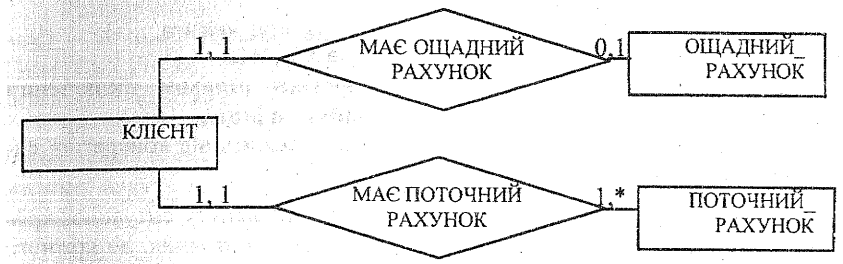
СТУДЕНТ ІЗ ВИЩОЮ ОСВІТОЮ {Код студента, Назва ВНЗ, Спеціальність}

Зовнішній ключ: *Код студента* посилається на СТУДЕНТ.

Перетворення відношень

Відношення перетворюються одним із трьох способів, залежно від їхньої потужності.

Відношення один-до-одного. Розглянемо такий приклад: кожен клієнт банківської установи має лише один ощадний рахунок; до кожного ощадного рахунка має доступ лише один клієнт (малюнок 24). Саме такі потужності – мінімальні і максимальні – позначені на схемі.



Мал. 24. Зображення відношень один-до-одного та один-до-багатьох

Ключем для множини КЛІЄНТ оберемо *Код_клієнта*. Ключем для множини ОЩАДНИЙ_РАХУНОК оберемо *Код_ощадного_рахунку*. Інші атрибути (наприклад, прізвище клієнта та ін.) для спрощення опускаємо, оскільки розглядаємо перетворення лише самого відношення. Таким чином, реляційна схема матиме такий вигляд:

КЛІЄНТ {Код_клієнта, … }

ОЩАДНИЙ_РАХУНОК {Код_ощадного_рахунка, … }.

Для того, щоб відобразити зв'язок між реляціями, потрібно включити *Код_ощадного_рахунку* в реляцію КЛІЄНТ, а *Код_клієнта* – у реляцію ОЩАДНИЙ_РАХУНОК. Звернемо увагу, що кожен із цих атрибутів буде зовнішнім ключем, що вказує на іншу таблицю.

КЛІЄНТ {Код клієнта, Код_ощадного_рахунка, ...}
Зовнішній ключ: *Код_ощадного_рахунка* посилається на ОЩАДНИЙ_РАХУНОК
ОЩАДНИЙ_РАХУНОК {Код_ощадного_рахунка, Код_клієнта, ... }
Зовнішній ключ: *Код_клієнта* посилається на КЛІЄНТ.

У такому випадку дані дублюються, оскільки наявні дві однакові комбінації (*Код_клієнта, Код_ощадного_рахунку)*. Одну з них можна опустити. Для визначення того, яку саме, звернемося то величин мінімальних потужностей. Якщо клієнт не має ощадного рахунка, то значення атрибута *Код_ощадного_рахунка* буде порожнім. З іншого боку, у реляції ОЩАДНИЙ_РАХУНОК значення атрибута *Код_клієнта* буде визначено для кожного ощадного рахунка. Отже, мінімальна потужність з боку реляції ОЩАДНИЙ_РАХУНКА дорівнює нулю, а мінімальна потужність з боку КЛІЄНТА – одиниця. У такому випадку найкращим вирішенням буде таке: виключити *Код_ощадного_рахунка* з реляції КЛІЄНТ (оскільки не кожен із них має ощадний рахунок). Таким чином, атрибут, який був ключовим в об'єктній множині з меншою мінімальною потужністю, виключили з множини із більшою мінімальною потужністю. Реляційна схема матиме такий вигляд:

КЛІЄНТ {Код_клієнта, ...}
ОЩАДНИЙ_РАХУНОК {Код_ощадного_рахунка, Код_клієнта, ... }
Зовнішній ключ: *Код_клієнта* посилається на КЛІЄНТ.

Відношення один-до-одного перетворюється шляхом включення ключа однієї із об'єктних множин у ролі атрибута в іншу.

Вибір того, ключ якої з множин включати, залежить від конкретної ситуації. Інколи обидва варіанти є прийнятними.

Відношення один-до-багатьох. Розширимо розглянуту ситуацію: крім ощадного рахунка клієнт має від одного до декількох поточних, до кожного поточного рахунка має доступ лише один клієнт. Потужності зображено на малюнку 24.

У довільному відношенні один-до-багатьох у таблицю, що описує множину, потужність відношення зі сторони якої є "багато", додатково включають атрибут, що є ключем іншої таблиці. Цей атрибут буде зовнішнім ключем таблиці.

Отримаємо наступну структуру реляції:

КЛІЄНТ {Код_клієнта, ... }
ПОТОЧНИЙ_РАХУНОК {Код_поточного_рахунка, Код_клієнта, ... }
Зовнішній ключ: *Код_клієнта* посилається на КЛІЄНТ

Відношення багато-до-багатьох. Таке відношення моделює випадок, коли до поточного рахунка буде мати доступ не один клієнт, а декілька (наприклад, чоловік та дружина). У загальному ситуація виглядає так: чоловік та дружина є клієнтами банку, кожен із них має свій власний ощадний рахунок. Крім того, вони мають один чи декілька поточних рахунків, доступ до яких мають обидва. Тобто потужності в обох напрямках будуть 1,*.

У випадку відношення багато-до-багатьох для перетворення створюють таблицю перетину.

Таблиця перетину – таблиця, у якій наявні елементи двох інших таблиць, що знаходяться у відношенні багато-до-багатьох.

Такою таблицею у нашому випадку буде таблиця МАЄ_ПОТОЧНИЙ_РАХУНОК (за назвою відношення). Оскільки ні клієнт однозначно не визначає номер поточного рахунка (рахунків може бути багато), ні номер поточного рахунка однозначно не визначає клієнта (до одного поточного рахунка має доступ декілька клієнтів), то ключем таблиці перетину будуть обидва ключі з двох реляцій КЛІЄНТ та ПОТОЧНИЙ_РАХУНОК.

КЛІЄНТ {Код_клієнта, ... }
ПОТОЧНИЙ_РАХУНОК {Код_поточного_рахунка, ...}
МАЄ_ПОТОЧНИЙ_РАХУНОК { Код_клієнта, Код_поточного рахунка}
Зовнішні ключі: *Код_клієнта* посилається на КЛІЄНТ
Код_поточного_рахунку посилається на ПОТОЧНИЙ_РАХУНОК.

У реляцію МАЄ_ПОТОЧНИЙ_РАХУНОК включають лише ключі реляцій КЛІЄНТ та ПОТОЧНИЙ_РАХУНОК. Якщо ці реляції мають інші атрибути (прізвище та адреса клієнта, дата відкриття рахунка тощо), їх не потрібно включати до таблиці перетину.

Реляція МАЄ_ПОТОЧНИЙ_РАХУНОК є таблицею перетину, оскільки вона відображає, як саме пов'язані елементи реляцій КЛІЄНТ та ПОТОЧНИЙ_РАХУНОК. Подібна таблиця перетину (яка відображає відношення) може мати додаткові неключові атрибути, що притаманні лише їй.

Довільне відношенні багато-до-багатьох перетворюється в реляційну схему шляхом створення нової таблиці (таблиця перетину), ключами якої будуть ключі двох об'єктних множин, що беруть участь у відношенні.

Перетворення складених об'єктних множин

Звернемося до вже розглядуваного нами прикладу поставок вантажу. Його концептуальну схему зображено на малюнок 16 а. Потрібно відстежувати відправки різних видів вантажу до різних клієнтів.

Перетворимо це відношення в реляційну схему за правилом переведення відношення багато-до-багатьох:

ВАНТАЖ {Код_вантажу, ... }
КЛІЄНТ {Код_клієнта, ...}
ОТРИМАВ { Код_вантажу, Код_клієнта, Кількість, ... }
Зовнішні ключі: *Код_вантажу* посилається на ВАНТАЖ
Код_клієнта посилається на КЛІЄНТ.

Таблиці ВАНТАЖ та КЛІЄНТ у зазначеній реляційній схемі мають по декілька атрибутів. Якщо це не так, то таблиці ВАНТАЖ та КЛІЄНТ можна не включати у схему (видалити їх). Тоді реляційна схема складатиметься з однієї таблиці ОТРИМАВ (вона не матиме зовнішніх ключів):

ОТРИМАВ {Вантаж, Клієнта, Кількість}.

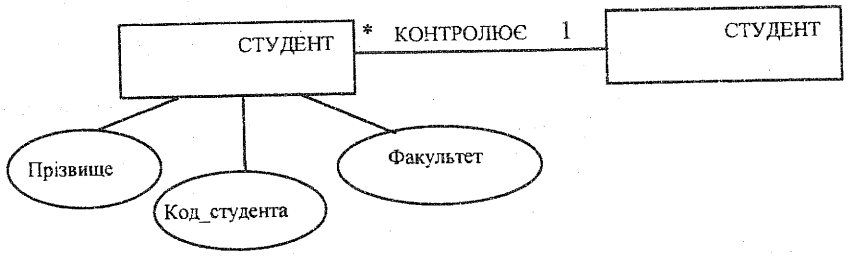
Розглянемо тепер випадок, коли відношення бере участь в іншому відношенні (малюнок 16 б). Необхідно визначити: скільки літрів води (соку, пива тощо) отримав клієнт Іванов (Петров, Сидоров) від постачальника Петренка (Сидоренка, Сомова).

Для цього потрібно пов'язати складений об'єкт ОТРИМАННЯ_ВАНТАЖУ із множиною ПОСТАЧАЛЬНИК. І вже цьому новому відношенню ОТРИМАВ ВІД (що теж розглядається як складений об'єкт) надати атрибут КІЛЬКІСТЬ. Реляційна схема такої концептуальної моделі (створена за тими самими правилами, що й попередня) виглядатиме так:

ВАНТАЖ {Код_вантажу, ... }
КЛІЄНТ {Код_клієнта, ...}
ПОСТАЧАЛЬНИК {Код_постачальника, ...}
ОТРИМАВ_ВІД { Код_вантажу, Код_клієнта, Код_постачальника, Кількість}
Зовнішні ключі: *Код_вантажу* посилається на ВАНТАЖ
Код_клієнта посилається на КЛІЄНТ
Код_постачальника посилається на ПОСТАЧАЛЬНИК.

Перетворення рекурсивних відношень

Звернемося до розглядуваної ситуації про проходження студентами різних практик: один студент контролює певну групу. Концептуальна схема такої моделі зображена на малюнку 25.



Мал. 25. Перетворення рекурсивних відношень

Зрозуміло, що об'єктна множина СТУДЕНТ, яка наявна у схемі двічі, є однією й тією самою. Ці дві копії мають одні й ті ж атрибути, хоч зображені вони лише для однієї з множин. На схемі вони використані одночасно для того, щоб відобразити рекурсивне відношення. Потужність відношення один-до багатьох. Один студент контролює багатьох. За визначеним раніше правилом перетворення відношення один-до-багатьох (у множину із потужністю багато додаємо ключ з іншої множини) трансформуємо це відношення в реляційну схему:

СТУДЕНТ {Код_студента, Прізвище, Факультет, Код_студента}

Такий розв'язок не є коректним, оскільки в таблиці наявні два атрибути з однаковими назвами. Вирішення проблеми полягає в тому, щоб замінити ім'я неключового атрибута, що повторюється, на ім'я, яке відповідає відношенню, яке цей атрибут представляє. Отже, кінцева реляційна схема з урахуванням зовнішнього ключа виглядатиме так:

СТУДЕНТ {Код_студента, Прізвище, Факультет, Контролює}
Зовнішній ключ: *Контролює* посилається на СТУДЕНТ

Висновок

Після перетворення всіх елементів концептуальної моделі потрібно ще раз переглянути отриману реляційну схему, позбавивши її від повторюваних даних (якщо такі будуть). Будь-які надлишкові дані або таблиці (дані яких повністю міститься в інших реляціях) видаляємо зі схеми.

Отримана реляційна схема є повністю нормалізованою. Причина цього полягає в тому, що атрибути відношень один-до-одного та один-до-багатьох є функціональними відношеннями. Розглянутий процес перетворення кожної з цих конструкцій в атрибути реляційної таблиці гарантує, що вони будуть залежати лише від ключових атрибутів. Отже, кожна отримана таблиця матиме 3НФ. Багатозначні (не атомарні) атрибути можуть бути лише у відношеннях багато-до-багатьох. А такі відношення перетворюються в окремі реляційні таблиці, що мають складені ключі – із ключових атрибутів окремих об'єктних множин.

ЧАСТИНА 3
РЕЛЯЦІЙНА АЛГЕБРА ТА РЕЛЯЦІЙНЕ ЧИСЛЕННЯ

Традиційні комп'ютерні мови програмування є процедурними (Fortran, Pascal, Basic та ін.). Процедурна мова забезпечує покрокове розв'язування задачі. Використовуючи синтаксис, набір команд, функцій, операторів тощо процедурної мови, записується текст програми, тобто всі етапи користувач повинен детально описати. Простіше кажучи, застосовуючи процедурну мову, даємо вказівки ПК, як потрібно вирішити те чи інше завдання.

Непроцедурна мова – мова, за допомогою якої формулюють, що потрібно зробити (отримати в кінцевому результаті).

Реляційна алгебра – процедурна мова обробки реляційних таблиць.

Реляційне числення – непроцедурна мова створення запитів.

Кодд довів, що реляційна алгебра та реляційне числення логічно еквівалентні. Це означає, що довільний запит, який можна сформулювати за допомогою реляційного числення, також можна сформулювати, користуючись операціями реляційної алгебри, і навпаки. Якщо мова запитів має ті самі можливості, що й реляційна алгебра, то така мова називається **реляційно повною**. Такою мовою можна означити довільний запит, що формулюється в реляційній алгебрі. Якщо мова матиме меншу логічну потужність, ніж реляційна алгебра чи реляційне числення, то будуть існувати запити, які не можна сформулювати за допомогою цієї мови.

§ 3.1. РЕЛЯЦІЙНА АЛГЕБРА

Операції реляційної алгебри застосовують до реляційних таблиць. Такі операції використовують одну або дві з наявних таблиць для створення третьої. Нова таблиця також може бути використана в інших операціях. Створення нових таблиць на основі існуючих дає великі можливості для обробки даних та отримання нової інформації.

Реляційна алгебра складається із дев'яти операцій: **об'єднання, перетин, різниця, добуток, вибірка, створення проекцій, поєднання, ділення, присвоєння.**

Перші чотири майже повністю збігаються з операціями теорії множин. Це – теоретико-множинні операції. Реляції є множинами, тому абсолютно логічно застосовувати до них операції над множинами.

Наступні чотири – спеціальні реляційні операції – застосовуються лише до реляційних таблиць. Операція присвоєння – стандартна для багатьох ком-

п'ютерних мов, використовується для надання змінній певного значення, має позначення :=.

Операції теорії множин (об'єднання, перетин, різниця) застосовують лише до об'єднально-сумісних таблиць

Об'єднально-сумісні таблиці – реляційні таблиці, що мають еквівалентні стовпчики з погляду їхньої кількості та області атрибутів.

Об'єднання

Нехай маємо чотири реляції з відомостями про студентів чотирьох факультетів ВНЗ, кожна з яких містить наступну інформацію: прізвище, ім'я, дату народження, адресу, факультет. Отже, ці таблиці є об'єднально-сумісними. Для того, щоб отримати зведену таблицю з даними про всіх студентів, потрібно об'єднати наявні чотири.

Об'єднання (позначення ∪, UNION) – операція реляційної алгебри, що створює теоретико-множинне об'єднання двох об'єднально-сумісних реляційних таблиць.

TabF1, TabF2, TabF3, TabF4 – таблиці з даними про студентів чотирьох різних факультетів. Для отримання загальної реляції виконаємо такі дії:

T1:= TabF1 ∪ TabF2

T2:= T1 ∪ TabF3

TabRez:= T2 ∪ TabF4.

Якщо один кортеж повторюється в декількох реляціях, результуюча таблиця міститиме його лише один раз.

Вимога об'єднальної сумісності початкових таблиць потрібна для того, щоб результатом об'єднання початкових таблиць була знову ж таки подібна реляція. Наприклад, якщо об'єднати таблиці КЛІЄНТ та УГОДИ (малюнок 1), то отримаємо деяку множину, яка, проте, не буде реляційною таблицею.

Зрозуміло, що ця вимога є необхідною і для операцій перетину та різниці.

Перетин

Перетин (позначення ∩, ^, INTERSECT) – операція реляційної алгебри, що створює теоретико-множинний перетин двох об'єднально-сумісних реляційних таблиць.

Операція перетину двох реляцій дає змогу визначити лише ті рядки, що наявні в обох початкових реляціях.

Наприклад, маємо дві реляційні таблиці: перша з даними про проходження студентами педагогічної практики (PED), друга – про проходження практики на агробіостанції (AGRO). Для визначення студентів, які пройшли обидві практики, використовують оператор перетину.

END:= PED ∩ AGRO.

Реляційна таблиця END міститиме дані про студентів, які наявні і в реляції PED, і в реляції одночасно.

Різниця

Різниця (позначення - , MINUS) – операція реляційної алгебри, що створює теоретико-множинну різницю двох об'єднально-сумісних реляційних таблиць.

Операція дає змогу визначити ті рядки, що наявні в першій реляції, але відсутні в другій.

Наприклад, маємо реляцію END із відомостями про студентів, які відпрацювали обидві практики, та реляцію PED із відомостями про студентів, які пройшли лише педагогічну. Використовуючи оператор різниці, можна легко отримати дані про студентів, які відпрацювали практику на агробіостанції:

AGRO:= END - PED.

Різниця двох реляційних таблиць визначається як реляція, що складається з усіх рядків, які входять до першої таблиці, але не входять до другої.

Добуток

Добуток (позначення *, TIMES) – операція реляційної алгебри, що створює декартовий добуток двох реляційних таблиць.

Для створення добутку двох реляцій А та В потрібно виконати такі дії:

1) поєднати атрибути двох таблиць А та В;

2) приєднати до кожного рядка таблиці А кожен рядок таблиці В.

Проілюструємо це на прикладі двох таблиць А та В.

A	
A1	A2
23	*12*
17	*67*

B	
B1	B2
100	*200*
300	*400*
500	*600*

C:=A*B			
A1	A2	B1	B2
23	*12*	*100*	*200*
23	*12*	*300*	*400*
23	*12*	*500*	*600*
17	*67*	*100*	*200*
17	*67*	*300*	*400*
17	*67*	*500*	*600*

Звернемо увагу, що кількість стовпчиків результуючої реляції дорівняє сумі стовпчиків початкових реляцій, а кількість рядків результуючої реляції – добутку рядків початкових реляцій.

Операція не є особливо важливою з погляду формулювання запиту, оскільки не дає ніякої нової інформації з вихідних даних. Для реальних запитів вона не використовується – дуже важко уявити собі запит на її основі. Проте добуток застосовується як складова частина операції поєднання, яка є надзвичайно важливою і часто вживаною.

Вибірка

Вибірка (позначення SELECT) – операція реляційної алгебри, яка виконує відбір рядків із таблиці, що задовільняють деяку умову.

Розглянемо таблицю КЛІЄНТ (малюнок 1).

Запит: отримати прізвища всіх клієнтів з України.

Сформуємо реляцію КЛІЄНТ_УКРАЇНА з новою інформацією та присвоїмо їй певне значення:

КЛІЄНТ_УКРАЇНА := SELECT (КЛІЄНТ: Країна = 'Україна')

Після команди SELECT у круглих дужках розміщуємо:

1) назву таблиці, з якої виконується вибірка;

2) двокрапку;

3) умову відбору (вказуємо назву стовпчика та значення, якому цей атрибут повинен дорівнювати. Беремо його в одинарні лапках, якщо використовується текст).

У цьому випадку отримаємо результуючу таблицю:

КЛІЄНТ_УКРАЇНА					
Код_клієнта	*Прізвище_клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_сума_боргу*	*Виплачено_на_сьогодні*
100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000

Умови відбору загалом ті ж самі, що й для оператора розгалуження IF більшості мов програмування.

В умові можуть бути використані 5 операторів порівняння: =, <, >, <=, >=. Для кожного з них існує оператор заперечення, наприклад, not =, not <.

Проте потрібно стежити за тим, щоб імена стовпчиків, що записані в умові відбору, обов'язково були наявними в таблиці, з якої виконується вибірка. Наведемо декілька прикладів умов, що можуть бути використані для таблиці КЛІЄНТ:

1) Код_клієнта = 115

2) Прізвище_клієнта = 'Клінтон'

3) Виплачено_на_сьогодні > 10000

4) Країна not= 'Росія'

За допомогою цих умов можна зробити такі запити на вибірку:

1) Надати інформація про клієнта, який має код 115.

2) Надати інформацію про клієнта (клієнтів) з прізвищем Клінтон.

3) Хто з клієнтів виплатив на сьогодні суму, більшу 10000 грн?

4) Надати інформацію про усіх клієнтів, крім тих, що проживають у Росії.

Різниця

Різниця (позначення - , MINUS) – операція реляційної алгебри, що створює теоретико-множинну різницю двох об'єднально-сумісних реляційних таблиць.

Операція дає змогу визначити ті рядки, що наявні в першій реляції, але відсутні в другій.

Наприклад, маємо реляцію END із відомостями про студентів, які відпрацювали обидві практики, та реляцію PED із відомостями про студентів, які пройшли лише педагогічну. Використовуючи оператор різниці, можна легко отримати дані про студентів, які відпрацювали практику на агробіостанції:

AGRO:= END - PED.

Різниця двох реляційних таблиць визначається як реляція, що складається з усіх рядків, які входять до першої таблиці, але не входять до другої.

Добуток

Добуток (позначення *, TIMES) – операція реляційної алгебри, що створює декартовий добуток двох реляційних таблиць.

Для створення добутку двох реляцій А та В потрібно виконати такі дії:

1) поєднати атрибути двох таблиць А та В;

2) приєднати до кожного рядка таблиці А кожен рядок таблиці В.

Проілюструємо це на прикладі двох таблиць А та В.

A	
A1	A2
23	*12*
17	*67*

B	
B1	B2
100	*200*
300	*400*
500	*600*

C:=A*B			
A1	A2	B1	B2
23	*12*	*100*	*200*
23	*12*	*300*	*400*
23	*12*	*500*	*600*
17	*67*	*100*	*200*
17	*67*	*300*	*400*
17	*67*	*500*	*600*

Звернемо увагу, що кількість стовпчиків результуючої реляції дорівняє сумі стовпчиків початкових реляцій, а кількість рядків результуючої реляції – добутку рядків початкових реляцій.

Операція не є особливо важливою з погляду формулювання запиту, оскільки не дає ніякої нової інформації з вихідних даних. Для реальних запитів вона не використовується – дуже важко уявити собі запит на її основі. Проте добуток застосовується як складова частина операції поєднання, яка є надзвичайно важливою і часто вживаною.

Вибірка

Вибірка (позначення SELECT) – операція реляційної алгебри, яка виконує відбір рядків із таблиці, що задовільняють деяку умову.

Розглянемо таблицю КЛІЄНТ (малюнок 1).

Запит: отримати прізвища всіх клієнтів з України.

Сформуємо реляцію КЛІЄНТ_УКРАЇНА з новою інформацією та присвоїмо їй певне значення:

КЛІЄНТ_УКРАЇНА := SELECT (КЛІЄНТ: Країна = 'Україна')

Після команди SELECT у круглих дужках розміщуємо:

1) назву таблиці, з якої виконується вибірка;

2) двокрапку;

3) умову відбору (вказуємо назву стовпчика та значення, якому цей атрибут повинен дорівнювати. Беремо його в одинарні лапках, якщо використовується текст).

У цьому випадку отримаємо результуючу таблицю:

КЛІЄНТ_УКРАЇНА					
Код_ клієнта	*Прізвище_ клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_ сума_боргу*	*Виплачено_ на_сьогодні*
100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000

Умови відбору загалом ті ж самі, що й для оператора розгалуження IF більшості мов програмування.

В умові можуть бути використані 5 операторів порівняння: =, <, >, <=, >=. Для кожного з них існує оператор заперечення, наприклад, not =, not <.

Проте потрібно стежити за тим, щоб імена стовпчиків, що записані в умові відбору, обов'язково були наявними в таблиці, з якої виконується вибірка. Наведемо декілька прикладів умов, що можуть бути використані для таблиці КЛІЄНТ:

1) Код_клієнта = 115

2) Прізвище_клієнта = 'Клінтон'

3) Виплачено_на_сьогодні > 10000

4) Країна not= 'Росія'

За допомогою цих умов можна зробити такі запити на вибірку:

1) Надати інформація про клієнта, який має код 115.

2) Надати інформацію про клієнта (клієнтів) з прізвищем Клінтон.

3) Хто з клієнтів виплатив на сьогодні суму, більшу 10000 грн?

4) Надати інформацію про усіх клієнтів, крім тих, що проживають у Росії.

Створення проекцій

При виконанні операції вибірки відповідь на кожен запит містив весь рядок даних, хоч це не завжди вимагається. Наприклад, у результаті виконання запиту "Визначити загальну суму боргу клієнта Сидоренка" потрібно отримати лише один стовпчик (атрибут *Загальна_сума_боргу*) .

Створення проекцій – операція реляційної алгебри, що створює нову таблицю шляхом виключення стовпчиків з існуючої таблиці.

Отримана в результаті створення проекції таблиця називається проекцією початкової таблиці. Спеціального позначення для цієї операції не існує. Потрібно просто вказати вихідну таблицю, а після неї у квадратних дужках зазначити потрібні стовпчики.

Звернемося до вже розглядуваного запиту: отримати прізвища всіх клієнтів з України. В результаті вибірки ми отримали таблицю КЛІЄНТ_УКРАЇНА з повною інформацією про потрібних клієнтів (хоч у запиті вимагалися лише прізвища). Створення проекції буде таким:

КЛІЄНТ_УКРАЇНА [Прізвище_клієнта]

У результаті ми отримаємо таку проекцію:

Прізвище_клієнта
Петренко
Сидоренко

Можна обрати більше одного атрибута, наприклад:

КЛІЄНТ_УКРАЇНА [Код_клієна, Прізвище_клієнта]

У результаті ми отримаємо таку проекцію:

Код_клієнта	*Прізвище_клієнта*
100	Петренко
105	Сидоренко

Важливою характеристикою операції створення проекцій є те, що вона автоматично виключає повтори рядків з результуючої таблиці, якщо вони наявні.

Наприклад, якщо потрібно визначити, з яких країн є клієнти, то запишемо таке створення проекції: КЛІЄНТ [Країна]. У результаті отримаємо таку проекцію:

Країна
Україна
Росія
США

Хоч Україна відзначена у двох кортежах.

Це відбувається і тоді, коли результуюча таблиця містить декілька стовпчиків. Якщо два кортежі мають однакові значення в кожному із стовпчиків проекції, то кортеж увійде в результуючу таблицю лише один раз.

Вкладення операцій

У реляційній алгебрі допускається **вкладення операцій – послідовне виконання двох різних операцій без присвоєння імені проміжній результуючій таблиці.**

Розглянемо запит: Назвати прізвище клієнта, який має код 115?

SELECT (КЛІЄНТ: Код_клієнта = '115') [Прізвище_клієнта].

Вкладення операцій можна виконувати довільним чином. У разі необхідності використовують дужки для визначення порядку операцій.

Поєднання

Поєднання (позначення JOIN) – операція реляційної алгебри, що пов'язує таблиці.

Операція використовується для поєднання даних між таблицями (зв'язування таблиць). Ця операція має три версії: природне поєднання, тета-поєднання, зовнішнє поєднання. Найбільш важливим і частовживаним є природне поєднання.

Природне поєднання. Розглянемо таблицю УГОДИ (малюнок 1). У ній зазначені коди клієнтів, торгових агентів та товарів, які беруть участь у кожній угоді. Ця інформація дає змогу створювати логічні зв'язки між таблицями КЛІЄНТ, ТОРГОВИЙ_АГЕНТ, ТОВАР.

Запит: назвати прізвища клієнтів, які здійснювали угоди через торгового агента з кодом 2.

Спочатку виконаємо вибірку всіх угод, що були здійснені за допомогою цього агента. Результуючу таблицю назвемо АГЕНТ_2:

АГЕНТ_2:= SELECT (УГОДИ: Код_агента = 2)

У результаті отримаємо таблицю АГЕНТ_2 (А):

А

АГЕНТ_2				
Дата	*Код_клієнта*	*Код_агента*	*Код_товару*	*Сума_угоди*
3.08.05	100	2	1	240
8.08.05	105	2	2	100
18.09.05	2000	2	4	150
3.12.05	2000	2	3	1050

Оскільки для виконання запиту потрібні дані з двох таблиць (КЛІЄНТ та АГЕНТ_2), виконаємо їх поєднання: JOIN (КЛІЄНТ, АГЕНТ_2).

Поєднання таблиць виконується в три етапи.

1. Знайдемо декартовий добуток двох таблиць, що поєднуються:

КЛІЄНТ_АГЕНТ_2:= КЛІЄНТ*АГЕНТ_2.

У результаті отримаємо таблицю (В) з 11 стовпчиків (за правилом множення таблиць, кількість стовпчиків результуючої таблиці дорівнює сумі стовпчиків вихідних таблиць) та 16 рядочків (за правилом множення таблиць, кількість рядочків результуючої таблиці дорівнює добутку кількості рядочків вихідних таблиць).

B

КЛІЄНТ_АГЕНТ_2

Дата	*Код_клієнта*	*Код_агента*	*Код_товару*	*Сума_угоди*	*Код_клієнта*	*Прізвище_клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_сума_боргу*	*Виплачено_на_сьогодні*
3.08.05	100	2	1	240	100	Петренко	Житомир, вул. В. Бердичівська, 1	Україна	10 267	8 000
3.08.05	100	2	1	240	105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
3.08.05	100	2	1	240	115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
3.08.05	100	2	1	240	2000	Клінтон	Чикаго, 910	США	33 333	30 000
8.08.05	105	2	2	100	100	Петренко	Житомир, вул. В. Бердичівська, 1	Україна	10 267	8 000
8.08.05	105	2	2	100	105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
8.08.05	105	2	2	100	115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
8.08.05	105	2	2	100	2000	Клінтон	Чикаго, 910	США	33 333	30 000
18.09.05	2000	2	4	150	100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
18.09.05	2000	2	4	150	105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
18.09.05	2000	2	4	150	115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
18.09.05	2000	2	4	150	2000	Клінтон	Чикаго, 910	США	33 333	30 000
3.12.05	2000	2	3	1050	100	Петренко	Житомир, вул. В.Бердичівська, 1	Україна	10 267	8 000
3.12.05	2000	2	3	1050	105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
3.12.05	2000	2	3	1050	115	Суханов	Москва, вул. В. Черкізівська, 33, кв 5.	Росія	75 500	70 300
3.12.05	2000	2	3	1050	2000	Клінтон	Чикаго, 910	США	33 333	30 000

2. Виконаємо вибірку лише тих рядків, у яких *Код_клієнта* з таблиці КЛІЄНТ дорівнює *Код_клієнта* з таблиці АГЕНТ_2. Отримаємо таблицю C:

C

КЛІЄНТ_АГЕНТ_2

Дата	*Код_клієнта*	*Код_агента*	*Код товару*	*Сума_угоди*	*Код_клієнта*	*Прізвище_клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_сума_боргу*	*Виплачено_на_сьогодні*
3.08.05	100	2	1	240	100	Петренко	Житомир, вул. В. Бердичівська,1	Україна	10 267	8 000
8.08.05	105	2	2	100	105	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
18.09.05	2000	2	4	150	2000	Клінтон	Чикаго, 910	США	33 333	30 000
3.12.05	2000	2	3	1050	2000	Клінтон	Чикаго, 910	США	33 333	30 000

3. У таблиці наявні два стовпчики *Код_клієнта* з однаковою інформацію. Один із них виключаємо. Отримаємо результуючу таблицю D:

D

КЛІЄНТ_АГЕНТ_2

Дата	*Код_клієнта*	*Код_агента*	*Код товару*	*Сума_угоди*	*Прізвище_клієнта*	*Адреса_клієнта*	*Країна*	*Загальна_сума_боргу*	*Виплачено_на_сьогодні*
3.08.05	100	2	1	240	Петренко	Житомир, вул. В. Бердичівська, 1	Україна	10 267	8 000
8.08.05	105	2	2	100	Сидоренко	Житомир, вул. Хлібна,15, кв. 78	Україна	1 000	1 000
18.09.05	2000	2	4	150	Клінтон	Чикаго, 910	США	33 333	30 000
3.12.05	2000	2	3	1050	Клінтон	Чикаго, 910	США	33 333	30 000

Таким чином, операція поєднання є результатом послідовного виконання декартового добутку, вибірки та створення проекції.

Розглянемо процес поєднання таблиць на рівні табличного пошуку. Для кожного рядка реляційної таблиці АГЕНТ_2 шукаємо рядки в таблиці КЛІЄНТ, що мають те ж саме значення стовпчика *Код_клієнта*. Оскільки

Код_клієнта є ключем таблиці КЛІЄНТ, то такий рядок у цій таблиці буде лише один. Отже, якщо в таблиці АГЕНТ_2 було 4 рядки, то в результуючій таблиці також буде 4 рядки. При цьому кожен із цих рядків буде розширено інформацією про клієнта.

У таблиці D міститься набагато більше інформації, ніж було в запиті. Тому створимо проекцію таблиці:

КЛІЄНТ_АГЕНТ_2 [Прізвище клієнта].

Природне поєднання – операція поєднання, що пов'язує таблиці, коли однакові стовпчики мають рівні значення.

У синтаксисі природного поєднання не вказується, за яким стовпчиком відбувається поєднання, оскільки воно здійснюється за всіма однаковими атрибутами.

Якщо дві таблиці мають більше ніж один спільний стовпчик, то поєднання виконується, враховуючи рівність значень у всіх стовпчиках. Якщо таблиця А має k стовпчиків, а таблиця В – m, то поєднання таблиць матиме (k+m-n) стовпчиків, де n – кількість однакових стовпчиків таблиць А і В.

Тета-поєднання

Тета-поєднання – операція поєднання, що пов'язує таблиці, у яких значення певних стовпчиків знаходяться в деякому відношенні.

Загальний вигляд тета-поєднання: JOIN (A,B: X <оператор відношення> Y),

де А та В – таблиці, що поєднуються; X та Y – стовпчики цих таблиць, що поєднані одним із шести операторів відношення.

При такому типі поєднання у зв'язку беруть участь не лише первинні та зовнішні ключі, але й інші стовпчики.

Розглянемо виконання наступного запиту: визначити всіх агентів, чиї менеджери отримують комісійні, більше 12%.

Дані про агентів та менеджерів містяться в реляції ТОРГОВИЙ_АГЕНТ. У стовпчику *Код_менеджера* зазначено код торгового агента, який є менеджером.

E

ТОРГОВИЙ_АГЕНТ				
Код_агента	*Прізвище_ агента*	*Код_ менеджера*	*Офіс*	*Комісійний_%*
1	Чейз		Лондон	15
2	Сомов	3	Москва	11
3	Дяченко	4	Київ	10
4	Акінава	1	Токіо	13
5	Райс	1	Глазго	10

Для виконання запиту використати операцію вибірки недоцільно, оскільки дані про комісійні відсотки стосуються торгового агента, а не його менеджера. Для визначення розміру комісійних менеджера потрібно приєднати запис про менеджера з таблиці ТОРГОВИЙ_АГЕНТ до запису про торгового

агента, тобто приєднати таблицю саму до себе. У такому випадку розмір комісійних менеджера стоятиме в тому ж рядку, що й ім'я торгового агента.

У наведеному прикладі природне поєднання застосувати не можна, оскільки поєднання не базується на рівності значень у однакових стовпчиках.

Крім того існує ще одна складність: приєднувати таблицю потрібно буде до самої себе, і, за правилами природного поєднання, у результуючій таблиці збігатимуться всі стовпчики, тобто поєднання втрачає смисл.

Вирішення цих проблем полягає у створенні додаткової копії вже існуючої таблиці та використанні тета-поєднання, яке дозволить задавати умову.

Проілюструємо виконання запиту.

1. Створимо дві копії таблиці ТОРГОВИЙ_АГЕНТ:

A1:= ТОРГОВИЙ_АГЕНТ

A2:= ТОРГОВИЙ_АГЕНТ

2. Виконаємо тета- поєднання:

МЕНЕДЖЕР:=JOIN (A1, A2 : A1.Код_менеджера = A2.Код_агента)

Пояснення. Поєднуються дві таблиці A1 та A2. Після двокрапки зазначено умову поєднання: два рядки поєднуються лише тоді, коли *Код_менеджера* першого рядка дорівнює *Код_агента* другого рядка. Тобто приєднуємо до кожного рядка агента рядок, що містить інформацію про його менеджера. У результаті такого об'єднання отримаємо таблицю:

I

МЕНЕДЖЕР									
Код_агента A1	*Прізвище_агента A1*	*Код_менеджера A1*	*Офіс A1*	*Комісійний_% A1*	*Код_агента A2*	*Прізвище_агента A2*	*Код_менеджера A2*	*Офіс A2*	*Комісійний_% A2*
2	Сомов	3	Москва	11	3	Дяченко	4	Київ	10
3	Дяченко	4	Київ	10	4	Акінава	1	Токіо	13
4	Акінава	1	Токіо	13	1	Чейз		Лондон	15
5	Райс	1	Глазго	10	1	Чейз		Лондон	15

Важливим є той факт, що кількість рядків результуючої таблиці на одну менша, ніж у таблиці ТОРГОВИЙ_АГЕНТ. Це пов'язано з тим, що в агента Чейза немає менеджера.

У таблиці МЕНЕДЖЕР визначимо тих агентів, менеджери яких отримують більше 12 відсотків комісійних, використавши просту операцію вибірки:

МЕНЕДЖЕР_12:=SELECT

(МЕНЕДЖЕР:[Комісійний_%A2] > 12 %) [Прізви-ще_агентаA1].

У результаті отримаємо таблицю:

МЕНЕДЖЕР_12
Прізви-ще_агентаА1
Дяченко
Акінава
Райс

Тета-поєднання, що базується на рівності визначених стовпчиків, називається еквіпоєднанням.

Саме таке поєднання було розглянуто в попередньому прикладі.

Тета-поєднанн дає змогу поєднувати й дві різні таблиці, а не лише таблицю з самою собою (як було розглянуто вище). Зрозуміло, що в такому випадку не потрібно створювати копії таблиць

Тета-поєднання, на відміну від природного поєднання не видаляє один або декілька спільних стовпчиків. Тобто якщо таблиця А має k стовпчиків, а таблиця В – m, то тета-поєднання таблиць матиме k+m стовпчиків.

Зовнішнє поєднання

Зовнішнє поєднання розширює природне поєднання, включаючи всі рядки з обох таблиць. Потреба у використанні такого поєднання є дуже незначною. Для запису операції використовують вираз OUTERJOIN (A, B).

Ділення

Операція ділення є оберненою до операції множення. На практиці використовується для виконання запитів, що містять в умовах слова "всі" або "кожен".

Ділення – операція реляційної алгебри, що створює нову таблицю шляхом вибору рядків однієї таблиці, які відповідають кожному рядку іншої таблиці.

Наведемо приклад використання операції.

Запит: назвати торгових агентів, які продали кожен товар.

Для виконання запиту скористаємося даними таблиць ТОВАР, ТОРГОВИЙ_АГЕНТ, УГОДИ (малюнок 1).

У таблицях наявна інформація про трьох торгових агентів та чотири найменування товару. Торговий агент задовольняє умову запиту, якщо він продав кожен із цих товарів хоча б 1 раз.

Ключовим словом запиту є *кожен,* оскільки вимагається, щоб для кожного агента ми перевіряли рядки таблиці УГОДИ доти, доки не переконаємося, що він продав кожен товар.

Розглянемо виконання запиту поетапно.

1. В усіх таблицях товар ідентифікується за його кодом, тому створимо проекцію: КОДТ := ТОВАР [Код_товару].

Результат: КОДТ

Код_товару
1
2
3
4

2. Створимо проекцію таблиці УГОДИ, де код торгового агента та код товару стоять поряд: ТОВ_АГ := УГОДИ [Код_товару, Код_агента]:

ТОВ_АГ

Код_товару	*Код_агента*
1	2
2	2
4	3
3	4
4	2
3	3
3	4
3	2

Якби в таблиці УГОДИ були рядки, що повторюються, вони були б виключені з проекції (в цьому випадку повторів не було).

3. Визначимо, кого із агентів представлено в таблиці ТОВ_АГ в поєднанні з кожним товаром. Це можна автоматично виконати за допомогою операції ділення

А:= ТОВ_АГ / КОДТ

У результаті отримаємо:

Код_агента
2

Визначимо деякі характерні особливості операції ділення:

1) для ділення таблиці А на В потрібно, щоб стовпчики таблиці В були підмножиною стовпчиків таблиці А;

2) у результаті ділення отримаємо таблицю із стовпчиками таблиці А, що не є стовпчиками таблиці В;

3) рядок таблиці А вміщується в результуючу таблицю лише за умови, що він поєднується з кожним рядком таблиці В.

У реляційній алгебрі не існує операції, яка б визначала максимальне значення. Операцію вибірки застосовувати недоцільно, оскільки ця операція за один раз обробляє лише один рядок (визначає відповідність його атрибутів певній умові). Можна використати тета-поєднання таблиці з самою собою, оскільки в такому випадку маємо змогу працювати з двома рядками одночасно.

Розглянемо такий запит: визначити максимальний розмір комісійних відсотків торгового агента. Дані будемо брати з уже розглядуваної таблиці:

Е

ТОРГОВИЙ_АГЕНТ				
Код_агента	*Прізвище_ агента*	*Код_ менеджера*	*Офіс*	*Комісійний_%*
1	Чейз		Лондон	15
2	Сомов	3	Москва	11
3	Дяченко	4	Київ	10
4	Акінава	1	Токіо	13
5	Райс	1	Глазго	10

Створимо дві нові таблиці на основі таблиці ТОРГОВИЙ_АГЕНТ:
N:= ТОРГОВИЙ_АГЕНТ [*Комісійний_%*]
M:=N

N

Комісійний_%
15
11
10
13
10

M

Комісійний_%
15
11
10
13
10

Z:= JOIN (N, M : N.Комісійний_% > M. Комісійний_%)
У результаті поєднання отримаємо таблицю:

Z

N. Комісійний_%	*M. Комісійний_%*
15	11
15	10
15	13
15	10
11	10
13	11
13	10

Проаналізуємо отриману таблицю. Характерним для неї є те, що в лівому стовпчику знаходяться всі комісійні без мінімального, а у правому – всі комісійні без максимального. Причина цього явища в тому, що саме так була сформульована умова відбору (N.Комісійний_% > M. Комісійний_%)

L:= Z [*M.Комісійний%*] (обрали всі значення без максимального).

L

M. Комісійний_%
11
10
13
10
10
11
10

Для отримання відповіді на запит використаємо операцію віднімання.
K:= N-L .

Операцію віднімання можна здійснювати, оскільки таблиці N та L є об'єдально-сумісними. У таблиці N містяться всі значення комісійних відсотків, а в таблиці L – усі значення, крім максимального. Таким чином, у результаті отримаємо відповідь – 15 %.

§ 3.2. РЕЛЯЦІЙНЕ ЧИСЛЕННЯ

У реляційному численні використовується інший підхід до виконання запитів. Проте обидві мови еквівалентні: довільний запит, який можна виконати засобами реляційної алгебри, можна виконати й за допомогою реляційного числення.

Реляційне числення за своєю суттю близьке до числення предикатів – одного з розділів математичної логіки. Поняття реляційного числення, як спеціального застосування числення предикатів, уперше було запропоновано Коддом у 1972 році. Згодом Кодд представив мову ALPHA, що базувалася безпосередньо на реляційному численні. Проте реалізованою мова не була.

Кодд стверджував, що реляційна алгебра потужніша за реляційне числення. Він створив алгоритм, за допомогою якого довільний вираз реляційного числення можна перетворити в еквівалентний вираз реляційної алгебри. Цей алгоритм отримав назву **алгоритмом редукції Кодда**.

Конструкції реляційного числення

Основним поняттям реляційного числення є змінна кортежу (позначаються малими латинськими літерами). У кожен момент роботи з реляціями певна змінна кортежу представляє конкретний кортеж.

Розв'язком запиту в реляційному численні є реляційна таблиця, що задається цільовим списком та визначальним виразом. Цільовий список визначає атрибути результуючої таблиці. На основі визначального виразу відбираються значення з бази даних, що будуть включені до кінцевої таблиці.

Цільовий список – список у виразі реляційного числення, що визначає атрибути таблиці розв'язку.

Визначальний вираз – умова у виразі реляційного числення, що обмежує входження елемента в таблицю розв'язку.

Запит: хто з виробників працює в Україні?

У реляційному численні цей запит реалізується за допомогою такого виразу:

{s.*Назва_виробника* : s in ВИРОБНИК and s.*Країна* = 'Україна'}

Пояснимо цей запис. Фігурні дужки означають, що відповіддю на запит може бути множина даних (не один виробник в Україні). Двокрапка розподіляє цільовий список та визначальний вираз.

Цільовий список

s.*Назва_виробника*

Визначальний вираз:

s in ВИРОБНИК and s.*Країна* = 'Україна'

s – довільний рядок реляції – змінна кортежу, що розглядається в певний момент часу;

s.*Назва_виробника* – значення атрибута *Назва_виробника* в рядку s.

s in ВИРОБНИК – визначає таблицю, з якої вибирається рядок s.

s.*Країна* = 'Україна' – визначення умови: значення атрибута *Країна* в рядку s дорівнює 'Україна'

Система переглядає почергово рядки таблиці ВИРОБНИК. Поточному рядочку присвоюється ім'я s та перевіряється істинність чи хибність визначального виразу (перевірка умови) для цього рядочка. У разі істинності дані з цього рядочка (у наведеному прикладі – лише *Назва_виробника*) вміщуються в таблицю розв'язку. Процес повторюється для кожного рядочка.

У результаті отримаємо:

Назва_виробника
Фабрика "Україна"

Як правило, цільовий список складається з одного атрибута, проте він може складатися і з декількох атрибутів, які потрібно розділити комою. Наприклад, якби у розглянутому випадку записати вираз

{s.*Назва_виробника*, s.*Адреса* : s in ВИРОБНИК and s.*Країна* = 'Україна'}, то отримали б результуючу таблицю:

Назва_виробника	Адреса
Фабрика "Україна"	Харків

У наведеному прикладі проілюстровано, як застосовують операції реляційної алгебри вибірка та створення проекції в реляційному численні.

Об'єднання, перетин, різницю та декартовий добуток також нескладно записати в конструкціях реляційного числення. Оскільки реляційне числення не передбачає покрокового виконання, операція присвоєння не використовується.

Для визначення операцій поєднання та ділення в реляційному численні введено поняття квантора. Квантор позначає кількість чогось. Квантор існування використовують для визначення операції поєднання, а квантор загальності – для ділення.

Квантор існування

Квантор існування (EXISTS) – вираз реляційного числення, що перевіряє існування принаймні одного кортежу реляції, що задовольняє певну умову.

Оскільки квантор існування забезпечує виконання операції поєднання реляційної алгебри, то для виконання запиту використовують дані з двох таблиць.

Запит: визначити прізвища клієнтів, який купили товар з кодом 1 (малюнок 1).

Запишемо цільовий список: s.Прізвище_клієнта.

Зазначимо таблицю, з якої обиратимемо дані: s in КЛІЄНТ.

{s.Прізвище_клієнта : s in КЛІЄНТ}

Змінну кортежу для таблиці КЛІЄНТ позначили літерою s.

Потрібно записати визначальний вираз. Для цього дамо відповідь на таке питання: які рядки повинні бути включеними до таблиці розв'язку?

Якщо *Код_клієнта* наявний у рядочку таблиці УГОДИ і має у цьому рядочку значення *Код_товару*, що дорівнює 1, то такий клієнт, точніше, за вимогою лише його прізвище з таблиці КЛІЄНТ уміщається до таблиці розв'язку.

Сформулюємо умову: існує хоча б один рядок таблиці УГОДИ, що містить код клієнта і значення атрибута *Код_товару* дорівнює 1. Засобами реляційного числення умова записується наступним чином:

exists z in УГОДИ (z.Прізвище_клієнта = s.Прізвище_клієнта and z. *Код_товару* = 1).

Читається цей запис так: існує рядок z у таблиці УГОДИ, для якого виконується умова (записана у круглих дужках). Важливо: цей вираз визначає рядок s із таблиці КЛІЄНТИ. Якщо рядок із таблиці КЛІЄНТИ задовольняє умову в дужках, то він (точніше, його частина) включається до таблиці розв'язку, в іншому випадку – ні.

Запишемо повний варіант виконання запиту в реляційному численні:

{s.Прізвище_клієнта : s in КЛІЄНТ and EXISTS z in УГОДИ

(z.Код_клієнта = s.Код_клієнта and z. *Код_товару* = 1)}

Цей вираз виконується в такому порядку:

- обирається перший рядок із таблиці КЛІЄНТ (позначений s);
- визначається значення атрибута *Код_клієнта* для цього рядка (у нашому випадку – 100);
- перевіряється, чи існує в таблиці УГОДИ такий рядок, що *Код_клієнта*=100, а *Код_товару*=1;
- s задовольнив записану умову, тому s.Прізвище_клієнта помішаємо в таблицю розв'язку;
- переходимо до наступного рядочка, позначивши його знову s, та аналогічно повторюємо пошук.

У результаті отримаємо таблицю розв'язків:

Прізвище_клієнта
Петренко

Наведений приклад демонструє виконання операції поєднання реляційної алгебри. Розглянемо ще один, складніший випадок, що потребує подвійного поєднання.

Запит: визначити, хто із клієнтів купував Чайний сервіз.

{r. Прізвище_клієнта : r in КЛІЄНТ and there exists s in УГОДИ and exists n in ТОВАР

(s.Код_клієнта = r. Код_клієнта and

s.Код_товару = n. Код_товару and

n.Назва_товару = ' Чайний сервіз').

Квантор загальності

Квантор загальності (FORALL) – вираз реляційного числення, який означає, що деяка умова виконується для кожного рядка певного типу.

Квантор використовується з тією ж метою, що і операція ділення в реляційній алгебрі (створення нової таблиці шляхом вибору рядків однієї таблиці, що відповідають кожному рядку іншої таблиці).

Квантор загальності FORALL введено для зручності, оскільки довільний вираз із використанням цього квантора можна записати використовуючи лише квантор існування EXISTS.

Запит: назвати прізвища торгових агентів, які продали кожен із товарів.

Використаємо таблиці ТОВАР, ТОРГОВИЙ_АГЕНТ, УГОДИ (малюнок 1).

Визначальним для запиту є слово *кожен,* оскільки вимагається, щоб для кожного агента ми перевіряли рядки таблиці УГОДИ доти, доки не переконаємося, що він продав кожен товар. Запис виразу мовою реляційного числення матиме такий вигляд:

```
{r. Прізвище_агента : r in ТОРГОВИЙ_АГЕНТ and
   FORALL z in ТОВАР
   EXISTS  s in УГОДИ
   and r.ТОРГОВИЙ_АГЕНТ=s. ТОРГОВИЙ_АГЕНТ}
```


ЧАСТИНА 4
МОВА ЗАПИТІВ SQL

Після створення реляційної моделі БД та реляційних мов (алгебри та числення) відбулося їхнє активне впровадження. З'явилися нові мови запитів:

SQL – Structured Query Language – мова структурованих запитів;

QBE – Query-by-Example – запит по зразку

QUEL – Query Language – мова запитів.

Перші дві мови було створено в 70-х роках XX столітті корпорацією IBM. Вони виконували досить подібні функції, проте SQL була текстовою мовою, а QBE – графічною. QUEL була розроблена у той самий період Каліфорнійським університетом Берклі.

На сьогодні SQL – одна з найпоширеніших мов запитів. Наприкінці 70-х вона стала доступною для загального вжитку, а у 1981 році корпорацією IBM було випущено комерційну СКБД, що підтримувала SQL.

Поступово широке застосування ПК у різних сферах призвело до появи реляційних баз і для мікрокомп'ютерів. А у 1986 році SQL була прийнята у якості стандарта мов для РБД. Цей стандарт було неістотно перероблено у 1989 році та досить суттєво – у 1992 (версія SQL-92). Й досі мова SQL залишається загальником, оскільки використовується в численних комерційних системах. Крім версій мови для універсальних ЕОМ, існують версії й для системи клієнт/сервер та персональних ЕОМ.

Хоч SQL позиціонується як мова запитів, проте вона включає в себе формування реляцій, оновлення БД, визначення представлень даних та пріоритетів доступу тощо.

Типи даних SQL

Для визначення областей, з яких обиратимуться дані, існує поняття типу даних. У SQL-92 визначені такі типи даних: точні числові, приблизні числові, символьні рядки, двійкові рядки, дата-час, interval.

Точні числові.

- Integer – довге ціле (4 байти: від – 2 147 483 648 до 2 147 483 648).
- Small Integer – коротке ціле (2 байти від – 32 768 до 32 768).
- Numeric (p, s) – число.
- Decimal (p, s) – десяткове число, де p – загальна кількість знаків, s – число знаків після коми.

Приблизні числові.

- Real – дійсне число.
- Double precision – дійсне число подвійної точності.
- Float – з плаваючою комою.

Ці типи даних використовуються для наукових та інженерних обчислень.

Символьні рядки.

Character (n) – символьний рядок.

Character varying (n) – символьний рядок змінної довжини.

Поля першого типу завжди зберігають n символів (заповнюючи у разі необхідності справа пропусками). Поля другого типу зберігають стільки символів, скільки введено.

Двійкові рядки.

Bit (n) – двійковий рядок

Використовують для прапорців та інших двійкових масок.

Дата-час.

DateTime – дата-час (довільний рік від 100 до 9999)

Interval.

Year-month – рік і місяць

Day-time – день і час

Проміжок – це різниця між двома датами (рік-місяць) чи двома проміжками часу (день-час)

Визначення схем

Звичайно під словником даних розуміють усі метадані, що визначають базу даних. У SQL цю функцію виконує інформаційна схема.

Інформаційна схема – схема у каталозі, що містить метадані.

У SQL для визначення схеми БД потрібно записати команду Create Schema <Ім'я > із необов'язковим параметром Authorization:

```
Create Schema FIRMA
Authorization N
```

Кожна така команда повідомляє СКБД про створення нової схеми даних. Команда Authorization визначає, хто є розпорядником (адміністратором) схеми. Схема ідентифікуватиме його за допомогою пароля. Саме розпорядник надає іншим користувачам право доступу до БД, її оновлення тощо.

Визначення області.

Визначення області – це зазначення типу даних, що будуть використані, обмежувальних умов для цих даних, а також визначення значень, що надаватимуться за замовчуванням. Визначається у схемі і використовується для окреслення окремих стовпчиків.

Обмежувальна умова – правило, що обмежує значення величини у БД.

Значення за замовчуванням – значення, що автоматично присвоюється системою, якщо користувач не вказав інше.

Наприклад, для визначення коду клієнта у БД потрібно визначити:

тип даних – Numeric;

обмежувальна умову – поле не може бути порожнім (бо ключове);

значення за замовчуванням – нуль.

У SQL область визначається командою Create Domain <Ім'я >. Для розглянутого вище атрибута із кодом клієнта запишемо таке визначення області:


```
Create Domain Cod_Klienta Numeric (5) Default 0
    Check (Value is not Null)
```

Цей запис означає, що область з іменем Cod_Klienta матиме такі властивості:

тип даних – п'ятизначне ціле число (Numeric (5));

значення за замовчування (Default 0);

обмежувальна умова визначається словом Check, а у дужках безпосередньо записана сама умова – Value is not Null (значення не нуль).

Визначення таблиць

Таблиці визначають у три етапи:

1) таблиці надається ім'я;

2) формується кожен стовпчик із можливими обмежувальними умовами та зазначенням ключових атрибутів;

3) визначається обмежувальні умови для всієї таблиці та зазначення зовнішніх ключів.

Розглянемо визначення таблиць КЛІЄНТ та УГОДИ, зображених на малюнку 1.

```
Create Table КЛІЄНТ (
    Код_клієнта          Cod                 Primary Key,
    Прізвище_клієнта     Character (15),
    Адреса_клієнта       Character (35),
    Країна               Character (15),
    Загальна_сума_боргу  Numeric (10, 2),
    Виплачено_на_сьогодні Numeric (10, 2) )
```

Зауважимо, що для атрибута *Код_клієнта* використано попередньо створену область Cod, та визначено цей атрибут як ключовий (Primary Key).

```
Create Table УГОДИ (
    Дата                 Data,
    Код_клієнта          Cod,
    Код_агента           Cod,
    Код_товару           Cod,
    Сума_угоди           Numeric (10, 2),
    Primary Key (Дата, Код_клієнта),
    Foreign Key  Код_клієнта References КЛІЄНТ
                         On Delete Cascade,
    Foreign Key Код_агента References ТОРГОВИЙ_АГЕНТ
                         On Delete Cascade,
    Foreign Key  Код_товару References ТОВАР
                         On Delete Cascade )
```

Зробимо деякі пояснення до записаного. По-перше, для таблиці визначено складений ключ (Primary Key) із переліком атрибутів у дужках, що одночасно є ключами. По-друге, у таблиці наявні три зовнішні ключі (Foreign Key), після яких зазначено поля та таблиці, на які вони посилаються. По-третє, для збереження цілісності БД після кожного зовнішнього ключа запи-

сано команду On Delete Cascade, яка забезпечує наступне: у разі видалення рядка, на який посилається зовнішній ключ, у відповідній таблиці (одній із трьох КЛІЄНТ, ТОГОВИЙ_АГЕНТ, ТОВАР), будуть видалені всі рядки поточної таблиці (УГОДИ), що посилаються на нього[1].

Нами були розглянуті обмеження, що накладалися на області (команда Check при визначенні області) та таблиці (команди On Delete Cascade, Delete Set Null). Розглянемо тепер приклад накладання обмежень на атрибут.

Пригадаємо таблицю із даними про проходження студентами різних видів практики (малюнок 21). У таблиці визначено чотири типи практики (педагогічна, агробіостанція, гуртожитки, корпуси). Отже, у полі *Вид_практики* могли міститися лише такі назви. Для такого обмеження на введення даних користуються командою Check. Таким чином, зазначення атрибута у рамках визначеної таблиці матиме наступний вигляд:

```
Тип_практики    Character (14)    Default 'Педагогічна'
Check (Тип_практики in ( 'Педагогічна', 'Агробіостанція', 'Гуртожитки',
                         'Корпуси' ) )
```

При введенні в інші таблиці коду практики (а їх є всього 4) знову можна скористатися обмежувальною умовою:

```
Код_практики    Numeric (1)       Default 1
Check (Код_практики >0 and  Код_практики <5)
```

Розглянемо випадок, коли при введенні у поле відомостей про стать потрібно вводити лише одну літеру – *ж* або *ч*. Для такого атрибута запис виглядатиме так:

```
Стать   bit (1)
Check (Стать in ( 'ж', 'ч' ) )
```

Зазначимо ще декілька важливих команд роботи зі схемою.

Alter Table – зміна таблиці – використовується для додавання, вилучення, або зміни існуючого атрибута;

Drop Table – видалення таблиці зі схеми – видаляє не лише кортежі таблиці, а й усі визначення цієї таблиці зі схеми;

Drop Chema <Назва схеми> Cascade – видалення схеми з усіма її таблицями, даними та іншими об'єктами.

Прості запити

Простий запит – запит, що звертається лише до однієї таблиці БД.

Найуживанішими командами мови SQL є: SELECT, FROM, WHERE.

SELECT – визначає стовпчики, які повинні ввійти в результуючу таблицю. Зрозуміло, що ці стовпчики повинні бути наявними в деякій реляційній таблиці. Якщо потрібно обрати декілька стовпчиків, то їх записують після команди SELECT через кому. Якщо потрібно вивести рядок повністю (усі атрибути), біля команди записують символ *: SELECT*.

[1] У випадку рекурсивного зовнішнього ключа використовується запис On Delete Set Null.

WHERE – визначає умову, за якою обираються рядки з реляційної таблиці. Для порівняння стовпчиків використовуються шість операторів порівняння (=, <>, <, >, <=, >=). Умов може бути декілька, у такому випадку вони поєднуються бульовими операціями: and, or, not. Для групування умов можуть бути використані дужки.

FROM – визначає таблицю (або таблиці), до яких звертається запит (з якого обиратимуть рядки та стовпчики). Усі стовпчики, що визначаються командами SELECT та WHERE, повинні бути наявними в одній із таблиць, записаних після команди FROM. Також можна звернутися до зовнішньої БД. У цьому випадку після команди FROM з переліченими назвами таблиць потрібно записати оператор IN та назву БД (наприклад, IN FIRMA).

Проілюструємо простий запит для таблиці КЛІЄНТ (малюнок 1).

Запит 1: хто з клієнтів є представником Росії?

```
SELECT    Прізвище_клієнта
FROM      КЛІЄНТ
WHERE     Країна ='Росія'
```

Для обробки запиту система звертається до команд у такому порядку: FROM, WHERE, SELECT. Рядки реляції, що вказана командою FROM, переміщуються в робочу область для обробки. До кожного кортежу послідовно застосовується умова, описана командою WHERE. Кортежі, які не задовольняють умову, виключаються з розгляду. Рядки, що залишилися, обробляються командою SELECT і включаються у результуючу таблицю.

Запит 2: хто з торгових агентів має комісійний відсоток більше 10 та проживає у Токіо?

```
SELECT    Прізвище_агента
FROM      ТОРГОВИЙ_АГЕНТ
WHERE     ([Комісійний_%] > 10) and (Офіс = 'Токіо').
```

Прокоментуємо записане. По-перше, у мові SQL для запису назв полів та таблиць не допускається безпосереднє використання специфічних символів, таких як #, %, &, *, {, }, [,]. У наведеному випадку існувало поле, що містить у своїй назві знак %, тому назва такого поля обмежена квадратними дужками. По-друге, після службового слова WHERE записана складна умова, що містить у собі дві прості (у круглих дужках), поєднані оператором and.

Запит 3: хто з торгових агентів має комісійний відсоток у межах від 10 до 12?

Для формулювання умови такого запиту можна записати:

```
WHERE     ([Комісійний_%] >= 10) and ([Комісійний_%] <= 12).
```

Проте, можна використати й команду **BETWEEN**:

```
WHERE     [Комісійний_%] BETWEEN 10 and 12.
```

Символи шаблонів

Символи шаблона – символи, що заміщують невизначені рядки символів.

У мові SQL використовують такі шаблони:

* – довільна кількість (починаючи з нуля) довільних символів;

? – один довільний символ.

Під час використання символів шаблона застосовують команду **LIKE**, яка й задає умову відбору.

Запит 4: Надати інформацію про торгових агентів, чиї прізвища починаються на "Акі"

```
SELECT  *
FROM    ТОРГОВИЙ_АГЕНТ
WHERE   Прізвище_агента LIKE 'Акі*'
```

Зазначимо, що в наведеному прикладі шаблон міститься безпосередньо у запиті, записаному на SQL. Проте команду LIKE можна використовувати й для введення шаблону безпосередньо від користувача під час використання запиту.

Запит 5: Надати інформацію про торгових агентів, чиї прізвища починаються з певних літер:

```
SELECT  *
FROM    ТОРГОВИЙ_АГЕНТ
WHERE   Прізвище_агента LIKE [Введіть початкові літери прізвища].
```

Запит почне працювати із виведення на екран повідомлення *Введіть початкові літери прізвища*. Після введення певної комбінації відбудеться виведення інформації за запитом із врахуванням правила відбору.

Сортування даних

Для сортування обраних за запитом записів використовують команду **ORDER BY.** Після неї зазначають поле, за яким потрібно відсортувати дані. У такому випадку сортування буде проведено за зростанням дати, числа, алфавіту тощо. Для сортування за спаданням потрібно після назви поля записати команду **DESC**.

Запит 6: хто з торгових агентів має комісійний відсоток більше 10 та проживає у Токіо? Прізвища агентів зазначити в алфавітному порядку.

```
SELECT      Прізвище_агента
FROM        ТОРГОВИЙ_АГЕНТ
WHERE       ([Комісійний_%] > 10) and (Офіс = 'Токіо' )
ORDER BY    Прізвище_агента
```

Для виведення даних у зворотному порядку останній рядок повинен мати вигляд:

```
ORDER BY    Прізвище_агента DESC
```

Сортування можна проводити за будь-яким полем, включаючи й поле, що має тип дата.

Використання у запитах полів, що мають тип Дата-Час

Використовуючи значення дати в запиті, потрібно дотримуватися таких правил:

- ✓ використовувати знак # на початку та в кінці значення дати;

- ✓ між значеннями дня, місяця та року ставиться розподілювач /, а не крапка (хоч зображення числа в таблиці матиме вигляд 14.10.2005);
- ✓ дата записується в такому порядку: місяць, число, рік

Запит 7: надати інформацію про угоди, що були здійснені 14.10.2005.

```
SELECT    *
FROM      УГОДИ
WHERE     Дата= #10/14/2005#
```

Для роботи з полями, що мають тип Дата-час, існує багато функцій. Розглянемо деякі з них:

Year (Дата) – повертає рік (повністю 1999 або 2003, хоч у форматі дати буде .99 або .03).

Month (Дата) – повертає місяць (номер)

Day (Дата) – повертає число (день)

Daty () – повертає поточну дату.

Запит 8: надати інформацію про угоди, що були здійснені протягом 2003 року.

```
SELECT  *
FROM    УГОДИ
WHERE   Year (Дата)=2003
```

Запит 9: хто з клієнтів сьогодні уклав угоди?

```
SELECT  Код_клієнта
FROM    УГОДИ
WHERE   Date()=Дата
```

Якщо в останньому запиті опустити круглі дужки, то дата вважатиметься невизначеною, і, перш ніж виконати запит, система намагатиметься уточнити дату в користувача (буде запропоновано ввести конкретну дату).

Для роботи з датами можна використовувати вже згадувану команду BETWEEN.

Запит 10: надати інформацію про угоди, що були укладені між 12.03.03 та 12.03.04.

```
SELECT  *
FROM    УГОДИ
WHERE   Дата BETWEEN #03/12/03# AND #03/12/04#.
```

Запит 11: надати інформацію про угоди, що були укладені за останні два тижні.

```
SELECT  *
FROM    УГОДИ
WHERE   Дата BETWEEN DATE() AND DATE() - 14.
```

Багатотабличні запити

Для об'єднання таблиць (таблиці повинні бути об'єднально-сумісними) використовують команду **UNION**. Звернемося до прикладу, що був розглянутий при ілюстрації операції об'єднання в реляційній алгебрі.

TabF1, TabF2 – таблиці з даними про студенті двох різних факультетів.

Для отримання реляції з прізвищами всіх студентів потрібно записати такий вираз:

```
SELECT  Прізвище From TabF1
UNION
SELECT  Прізвище From TabF2;
```

Якщо деякий рядок повторюється в обох таблицях, у результуючу реляцію він буде включений лише один раз. Для виведення всіх рядків (і тих що повторюються) використовують команду **UNION ALL**.

Поєднання даних із декількох таблиць

Зв'язок даних із декількох таблиць у реляційній алгебрі реалізується за допомогою операції поєднання. У SQL зв'язування таблиць виконується аналогічно. Для природного та тета-поєднання використовують операцію **INNER JOIN**, яка поєднує записи двох таблиць, якщо поля зв'язку містять однакові дані. У загальному операція має вигляд:

FROM *Таб1* INNER JOIN *Таб2* ON *Таб1.ПолеN оператор Таб2.ПолеM*

В умові може бути використано довільний логічний оператор (=, <, >, <=, >= <>). Потрібно пам'ятати, що поля, які пов'язуються, повинні мати однакову область значень.

Запит 12: які товари були придбані за угодами від 08.07.05?

```
SELECT  Назва_товару
FROM    УГОДИ INNER JOIN ТОВАР ON
        ТОВАР.Код_товару=УГОДИ.Код_товару
WHERE   Дата = #07/08/05#
```

Запит обробляється системою так:

1) обробляється команда FROM. Оскільки вона містить перелік двох таблиць з операцією INNER JOIN, то спочатку буде виконано їхнє природне поєднання;

2) виконується команда WHERE: з таблиці, яка утворилася в результаті поєднання, обираються ті рядки, в яких поле *Дата* задовольняє умову WHERE (дата = 8.07.05);

3) обробляється команда SELECT: з утвореної таблиці обираються дані лише із зазначеного поля *Назва_товару*.

У наведеному прикладі в реляції УГОДИ для кожного кортежу поле *Код_товару* обов'язково містить певне значення (якийсь товар за угодою продали, інакше угода не потрібна).

Проте існують й інші випадки. Наприклад, є дві реляції з даними про працівників університету та кафедри: СПІВРОБІТНИКИ та КАФЕДРИ. Для відбору всіх співробітників за кожною з кафедр (виводитиметься 2 поля – прізвище та кафедра) використовують операцію INNER JOIN. Проте співробітники, які не належать до жодної з кафедр (працівники бухгалтерії. бібліотеки та ін.), не будуть включені у запит, що використовує операцію INNER JOIN (тобто результуюча таблиця матиме менше рядків, ніж реляція СПІВРОБІТНИКИ).

Для відображення всіх співробітників використовують одну із операцій **LEFT JOIN, RIGHT JOIN**, які створюють зовнішнє поєднання. Загальний вигляд цих операцій той самий, що і у INNER JOIN.

Кількість кортежів результуючої таблиці за використання цих операцій завжди буде збігатися з кількістю кортежів реляції СПІВРОБІТНИКИ. У цьому випадку співробітник, який не працює на жодній кафедрі, теж буде включений у таблицю, а поле *Кафедра* буде порожнім. Причому в результуючій таблиці запиту саме з таких робітників буде розпочато перелік.

Операція LEFT JOIN створює ліве зовнішнє поєднання, яке включає всі записи_першої таблиці. Операція RIGHT JOIN створює праве зовнішнє поєднання, яке включає всі записи другої таблиці. У запитах ці операції можуть бути включені в операцію INNER JOIN. Проте INNER JOIN не може бути включена в операції LEFT JOIN та RIGHT JOIN.

Поєднання таблиці із самою собою

Запит 13: назвати всіх агентів, чиї менеджери отримують комісійний відсоток більше 12.

Усі дані для містяться в таблиці ТОРГОВИЙ_АГЕНТ (у таблиці є стовпчик *Код_менеджера,* який є торговим агентом).

ТОРГОВИЙ_АГЕНТ

Код_агента	*Прізвище_агента*	*Код_менеджера*	*Офіс*	*Комісійний_%*
1	Чейз		Лондон	15
2	Сомов	3	Москва	11
3	Дяченко	4	Київ	10
4	Акінава	1	Токіо	13
5	Райс	1	Глазго	10

```
SELECT  ТОРГОВИЙ_АГЕНТ.Прізвище_агента
FROM    ТОРГОВИЙ_АГЕНТ INNER JOIN ТОРГОВИЙ_АГЕНТ AS
        ТОРГОВИЙ_АГЕНТ _1 ON
        ТОРГОВИЙ_АГЕНТ.Код_агента=ТОРГОВИЙ_АГЕНТ_1.Код_менеджера
WHERE   [ТОРГОВИЙ_АГЕНТ.Комісійний_%]>[ТОРГОВИЙ_АГЕНТ_1.Комісійний_%]
```

Зауваження до використання операції поєднання

1. Якщо назва стовпчика наявна у більше ніж одній таблиці, то потрібно обов'язково вказати ім'я таблиці, з якої цей стовпчик береться (назва стовпчика і таблиці розділяється крапкою). Якщо ж ім'я стовпчика відзначено лише в одній таблиці, то її назву вказувати не потрібно.

2. Не можна поєднувати поля типу OLE та MEMO.

3. Допускається поєднання двох довільних числових полів подібних типів, наприклад, поле лічильника та поля типу "довге ціле". Проте не можна об'єднувати типи полів одинарної та подвійної точності (Single та Double)

4. Поєднувати можна не лише дані з таблиць але і з запитів (за тими самими правилами).

Підзапити

Підзапит – запит, який міститься всередині іншого запиту. Такі запити інколи називають вкладеними.

Зовнішній запит – запит, який містить підзапит.

Запит 14: які клієнти купили товару на суму, більшу 500 грн.?

```
SELECT   Прізвище_клієнта
FROM     КЛІЄНТ
WHERE    Код_клієнта IN
         (SELECT Код_клієнта
          FROM УГОДИ
          WHERE Сума_угоди > 500)
```

Підзапитом є вираз у круглих дужках. Його виконання призводить до створення підмножини угод на суму, більшу 500 грн. Так ми отримуємо наступну таблицю.

УГОДИ				
Дата	*Код_клієнта*	*Код_агента*	*Код_товару*	*Сума_угоди*
2.11.05	2000	4	3	700
3.12.05	2000	2	3	1050

А вже для цієї таблиці (підмножини) виконується зовнішній запит.

Зовнішній запит обробляє кожен рядок таблиці КЛІЄНТ відповідно до умови WHERE: якщо значення *Коду_клієнта* рядка міститься у підмножині підзапиту, то *Прізвище_клієнта* виводиться у результуючій таблиці.

Пояснення до записаного запиту 1) Команда SELECT підзапиту містить лише *Код_клієнта*, саме те поле, яке описане у команді WHERE зовнішнього запиту. 2) Спочатку виконується підзапит, він є самостійним і незалежним від зовнішнього запиту. 3) Вкладених підзапитів може бути декілька. Виконання починається із "найглибшого" підзапиту.

Підзапит, що не залежить від жодного зовнішнього запиту, називається **незалежним (некорельованим)**.

Незалежний підзапит виконується в першу чергу, а реляція, що утворилася в результаті такого підзапиту, використовується для виконання зовнішнього запиту (запит 14).

Корельований підзапит – підзапит, результат якого залежить від рядка, що розглядається головним (зовнішнім) запитом.

Такий запит виконується наступним чином: 1) обробляється поточний рядок таблиці зовнішнього запиту; 2) залежно від значення полів цього рядка виконується підзапит; 3) для визначення, чи потрібно вносити рядок до результуючої таблиці, перевіряється умова WHERE зовнішнього запиту.

Корельований підзапит можна використати для виконання запиту 13. У разі його застосування буде опущено операцію поєднання INNER JOIN.

Ключові слова IN, ANY, ALL, EXISTS.

У запиті 14 було використане ключове слово **IN**, яке перевіряє приналежність деякого елемента певній множині. У разі потреби переконатися, що елемент не міститься у визначеній таблиці, потрібно записати **NOT IN**.

У підзапитах також використовують ключові слова **ANY** та **ALL**, які записують після операторів порівняння (=, <>, <, >, <=, >=).

ANY застосовують для перевірки істинності умови хоча б для одного значення множини, ALL – для перевірки істинності умови для всіх значень множини.

Внутрішні функції

Часто в запитах доводиться визначати мінімальні та максимальні значення певного поля (наприклад, найбільший чи найменший комісійний відсоток), середні значення (наприклад, заробітної плати певної категорії працівників), загальну кількість (наприклад, угод, що перевищують певну суму), загальну суму (наприклад, вартість придбаного в 2003 році обладнання).

При створенні таких запитів використовують внутрішні функції:

MAX () – визначення максимального значення.

MIN () – визначення мінімального значення.

Ці функції оперують одним атрибутом таблиці (він вказується у дужках), вибираючи з нього мінімальне або максимальне значення. У формулюванні запиту не передбачено використання команди WHERE.

Запит 15: визначити максимальну та мінімальну суму угоди з таблиці УГОДИ.

```
SELECT  MAX(Сума_угоди) AS [MAX], MIN(Сума_угоди) AS [MIN]
FROM    УГОДИ
```

Записи AS [MAX], AS [MIN], означають, що нові поля із максимальним та мінімальним значенням називатимуться відповідно MAX та MIN. Загалом можна не вводити назви нових полів, проте тоді ці поля матимуть назву Expr1000 та Expr1001.

SUM – визначення загальної суми.

Запит 16: визначити загальну суму угод, які здійснив агент із кодом 2.

```
SELECT  SUM (Сума_угоди) AS [Загальна_сума_угод]
FROM    УГОДИ
WHERE   Код_агента =2;
```

COUNT – визначення загальної кількості.

Запит 17: визначити загальну кількість угод, як здійснив агент із кодом 2.

```
SELECT  COUNT (Сума_угоди) AS [Кількість_угод]
FROM    УГОДИ
WHERE   Код_агента =2;
```

AVG – визначення середнього значення.

Запит 18: визначити середню суму, на яку здійснив угоди певний агент.

```
SELECT    AVG (Сума_угоди) AS [Середня_сума_угод]
FROM      УГОДИ
WHERE     Код_агента LIKE [Введіть код агента];
```

Використання групових операцій у запитах

Під час експлуатації БД часто потрібна статистична інформація про певні групи даних (групи клієнтів, агентів, товарів тощо). Розглянемо приклад: існує БД із даними про всіх студентів університету. Проте логічно розглядати певні групи – студенти певного факультету, студенти, які пройшли службу в

армії, студенти-п'ятикурсники та ін. Групи у БД виділяють за певним атрибутом. Відповідно до наведеного прикладу про студентів це будуть такі атрибути: *факультет, служба_в_армії, курс.*

Для впорядкування даних за певною ознакою використовують операцію **GROUP BY**, вказавши безпосередньо після неї атрибут, за яким проводиться групування.

Запит 19: визначити, скільки разів продали кожен із товарів (малюнок 1).

```
SELECT    Товар.Назва_товару, Count(Угоди.Код_уг) AS [Кількість_угод]
FROM      Товар INNER JOIN Угоди ON Товар.Код_товару = Угоди.Код_товару
GROUP BY Товар. Назва_товару
```

Порядок виконання цього запиту такий:

після команди FROM відбувається поєднання таблиць ТОВАР та Угоди за кодом товару;

в отриманій таблиці всі рядки розбиваються на групи за таким правилом: рядки включаються в групу лише за умови, що у них збігаються значення атрибутів *Назва_товару*, тобто вони групуються за назвою товару (рядок GROUP BY);

для кожної групи товарів виконується вибірка та підрахунок кількості проведених угод (рядок SELECT).

Операція GROUP BY може бути застосована також із командою WHERE.

Запит 20: визначити, скільки разів продали кожен із товарів клієнту з кодом 100.

```
SELECT    Товар.Назва_товару, Count(Угоди.Код_уг) AS [Кількість_угод]
FROM      Товар INNER JOIN Угоди ON Товар.Код_товару =
          Угоди.Код_товару
WHERE     Угоди.Код_клієнта=100
GROUP BY Товар. Назва_товару
```

У розглянутому випадку умова накладалася на рядок.

Проте умова може накладатися і на групу. В такому випадку використовується операція **HAVING**. Ця операція має те саме значення, що й команда WHERE для рядків таблиці, і може бути застосованою лише за наявності у запиті операції GROUP BY.

Запит 21: визначити товари, які продали на загальну суму > 1000.

```
SELECT      Товар.Назва_товару, SUM (Сума_угоди)
            AS [Загальна_сума_угод]
FROM        Товар INNER JOIN Угоди ON Товар.Код_товару =
            Угоди.Код_товару
GROUP BY    Товар. Назва_товару
HAVING SUM  (Сума_угоди) > 1000.
```


ПРАКТИЧНІ РОБОТИ

Практична робота № 1
Основні поняття теорії БД та ІС, що використовують БД

Мета: закріпити на практиці використання основних понять теорії БД та ІС на основі БД. Набути вміння та навички складання ієрархічних і мережних структур даних, навчитися визначати їх атрибути та ключові поля.

Контрольні питання

Пояснити суть наступних термінів:

- система обробки даних;
- довільний доступ;
- інформаційно-керуюча система;
- база даних;
- ІС, що використовують БД (зазначити їх переваги);
- ієрархічна модель даних;
- мережна модель даних;
- вказівник;
- система клієнт/сервер;
- представлення даних;
- реляційна модель БД;
- життєвий цикл БД;
- спільне використання даних;
- трирівнева архітектура БД.

Завдання

1. Порівняти файлові системи з послідовним та довільним доступом. Назвати недоліки традиційних файлових систем.
2. Дати характеристику кожній з існуючих моделей даних.
3. Охарактеризувати та порівняти централізовані та розподілені БД.
4. Описати кожен із компонентів сучасної ІС, що використовує БД, та зв'язки між ними.
5. Описати функцій СКБД:
   - централізований контроль даних;
   - захист даних та підтримка їхньої цілісності;
   - забезпечення доступу до даних декількох користувачів одночасно;
   - керування буферами ОП;
   - керування транзакціями;
   - журналізація;
   - засоби створення прикладних програм.

6. Пояснити суть розділення логічного та фізичного представлення даних.
7. Назвати етапи ЖЦБД та охарактерихувати їх.
8. Визначити ключ кожного з файлів на малюнку 1. З'ясувати недоліки структури файлу УГОДИ.
9. Які з тверджень є даними, а які інформацією:
   - студент Шевченко навчається на 2 курсі;
   - студентка Петрова займається найкраще в групі;
   - студентка Петрова народилася 2 серпня 1987 року;
   - студент Шевченко народився в День Незалежності України;
   - товар з номером 123 вигідно продається;
   - торговий агент Сомов отримує 11% комісійних.
10. Створити для банкової установи ієрархічну структуру БД, що складається з таких файлів: ВНЕСОК, ОЩАДНИЙ РАХУНОК, КЛІЄНТ, КРЕДИТ, ЗНЯТТЯ, ДЕПОЗИТ.
11. Створити для транспортної компанії мережну модель БД, використовуючи такі файли: ВАНТАЖ, ТРАНСПОРТ, ПОСТАЧАЛЬНИК, УПАКОВКА, КЛІЄНТ.
12. У попередніх двох задачах визначити поля, що можуть бути в кожному з файлів. Встановити ключі.
13. Пояснити, яким чином одночасний неконтрольований доступ до БД може викликати проблеми в таких ситуаціях:
   - при резервуванні місць у системі продажу авіаквитків;
   - при оновленні кількості товарів на складі;
   - при веденні реєстрації студентського контингенту у ВНЗ.
14. Обговорити недоліки та переваги спільного використання даних у таких випадках:
   - різними відділами;
   - регіональними відділеннями.
15. Пояснити відмінність між концептуальним та зовнішнім рівнями у трирівневій архітектурі.
16. Класифікувати типи інформації відповідно до того, на якому рівні користувачів вони можуть мати найбільш суттєве значення (а можливо, не мати значення взагалі):
   - населення N-го району збільшувалося удвічі за кожне минуле десятиліття;
   - студентка Петренко змінила прізвище на Довженко;
   - студент Шевченко не вніс плату за навчання за останні 2 місяці;
   - у слюсаря автосервісу Жукова народилася друга дитина;
   - продаж автомобілів марки Мерседес лідирує впродовж останніх двох років в Україні;
   - витрати на ремонт обладнання набагато перевищили затверджений кошторис;

   - клієнт банку Сомов одружився 12.12.06.;
   - студентка Іванова не склала три іспити протягом літньої сесії;
   - найбільшим попитом у супермаркеті користується соняшникова олія виробника N;
   - клієнт банку Сидоренко змінив місце проживання.

Практична робота № 2
Концептуальне проектування БД

Мета: набути практичних умінь та навичок побудови концептуальних моделей для ситуацій реального життя (створення об'єктних множин, зазначення їх атрибутів, установлення відношень між ними, визначення потужностей об'єктних множин).

Контрольні питання

Пояснити суть наведених термінів:
- реальність і модель;
- об'єкт-елемент;
- об'єктна множина;
- лексична множина;
- абстрактна множина;
- сурогатний ключ;
- конкретизація та узагальнення;
- відношення;
- потужність відношень;
- функціональне відношення;
- атрибут;
- ключ;
- ідентифікатор.

Завдання

1. На основі об'єктних множин ПОТОЧНІ_РАХУНКИ, ОЩАДНІ_РАХУНКИ, КЛІЄНТИ побудувати таку концептуальну схему, щоб можна було отримати відповіді на запитання:
   - яка загальна кількість поточних рахунків?
   - яка загальна кількість ощадних рахунків?
   - яка загальна кількість клієнтів?
   - у кого з клієнтів є як поточні, так і ощадні рахунки?
2. Встановити відношення між об'єктними множинами та зазначити їх потужності в обох напрямках для таких випадків:
   - у кожного клієнта є лише один ощадний і один поточний рахунок;
   - у кожного клієнта є декілька як поточних, так і ощадних рахунків;
   - до одного рахунку може мати доступ декілька клієнтів.

3. Як, враховуючи встановлені відношення, дати відповіді на запитання:
   - у скількох клієнтів є декілька поточних рахунків?
   - яка кількість спільних ощадних рахунків?
   - скільки клієнтів, які мають декілька поточних рахунків, мають також і ощадні ?

4. Конкретизувати множину КЛІЄНТ, врахувавши, що клієнти можуть бути як фізичними, так і юридичними особами. Для чого потрібна така конкретизація?

5. Зазначити на схемі атрибути елементів множини КЛІЄНТ, а також її конкретизацій, так, щоб можна було дати відповіді на запитання:
   - який клієнт (юридична чи фізична особа) має найбільші вкладення на ощадному рахунку?
   - який максимальний відсоток нарахувань на ощадні вкладення?
   - скільки юридичних осіб мають поточні рахунки?
   - скільки фізичних осіб мають ощадні рахунки?
   - яка загальна сума коштів на поточних рахунках?

6. Чи можна за створеною схемою дати відповідь на питання:
   - яка кількість жінок має ощадний рахунок більше 10 000 грн.?
   - скільки жінок відкрили поточні рахунки 1 вересня 2005 року?
   - скільки існує ощадних рахунків підприємств (юридичних осіб), на яких працює більше 500 осіб?

7. Модифікувати модель так, щоб можна було отримати відповіді на поставлені запитання. Визначити ключ для кожної об'єктної множини.

8. Яку ще інформацію можна отримати з такої моделі даних? Поставте свої запитання.

9. На основі об'єктних множин ВИКЛАДАЧІ, СТУДЕНТИ побудувати таку концептуальну модель даних університету, щоб можна було отримати відповіді на запитання:
   - скільки студентів спеціальності "Інформатика" пишуть курсову роботу у викладача N?
   - назвати прізвище викладача, який керує найбільшою кількістю курсових робіт. Вказати факультет, де працює цей викладач;
   - скільки студентів навчаються за спеціальністю "Інформатика"?
   - скільки студентів, які пишуть курсову роботу у викладача N, проживають у гуртожитку?
   - назвати прізвища викладачів, які не керують курсовими роботами;
   - скільки студентів відслужили в армії?

10. Модифікувати модель так, щоб можна було дати відповіді на запитання:
   - назвати прізвища викладачів, які керують дипломними роботами;
   - назвати прізвища студентів, котрі пишуть дипломні роботи;
   - назвати прізвища студентів спеціальності "Інформатика", які пишуть дипломні роботи?

   - скільки студентів, котрі пишуть дипломну роботу, є місцевими?
   - скільки викладачів, які керують дипломними роботами, мають ступінь доктора наук?

11. Визначити ключ для кожної об'єктної множини.

12. Додавши об'єктну множину КНИГИ, розширити створену модель так, щоб можна було отримати відповіді на запитання:
   - яку книгу найчастіше беруть студенти з бібліотеки?
   - яку книгу найчастіше беруть викладачі з бібліотеки?
   - назвати прізвища студентів спеціальності "Інформатика", які використовували книгу О.М. Спіріна "Практична інформатика"?
   - скільки разів за поточний навчальний рік брали книгу "Педагогічний словник" за редакцією В.П. Іванова?
   - студенти якої спеціальності найчастіше беруть книгу О.Л. Краскевича "Чисельні методи"?

13. Побудувати таку концептуальну модель даних торгівельної фірми, щоб можна було отримати відповіді на запитання:
   - які товари мають ціну продажу більше, ніж 200 грн.?
   - які з них мають закупівельну ціну меншу, ніж 180 грн.?
   - назвати товари, які вироблено в Китаї;
   - назвати виробника цих товарів;
   - які товари, що мають закупівельну ціну більшу, ніж 1000 грн., зроблені в Україні?
   - хто із продавців продав товари ціною більшою, ніж 2000 грн.?
   - яка базова зарплатня цих продавців?
   - який продавець має найбільшу базову зарплатню?
   - який продавець продав найбільше товарів виробництва України?
   - який прибуток отримала фірма від продажу товарів фірми N?

14. Визначити ключ для використаних об'єктних множин.

Практична робота № 3
Побудова концептуальної моделі даних на основі існуючих звітів. Складені об'єкти.

Мета: набути практичні вміння та навички побудови концептуальних моделей даних на основі готових звітів та з використанням складених об'єктів.

Контрольні питання
   - дати визначення складеної об'єктної множини;
   - чи можуть складені об'єкти мати власні атрибути?
   - чи можуть складені об'єкти змінювати атрибути?
   - чи можуть складені об'єкти брати участь у відношеннях?

Завдання

1. Побудувати концептуальну модель даних бібліотеки університету на основі готових звітів.

ЛОГОТИП *Бібліотека Житомирського державного університету*

Студент
Прізвище _____
Ім'я _____
По батькові _____
Факультет _____
Група _____
Дата народж. _____
Місце прожив. _____

У період з _____ до _____ отримав у бібліотеці такі книги:

Автор _____
Назва _____
Ідентифікаційний номер _____

ЛОГОТИП *Бібліотека Житомирського державного університету*

Студент
Прізвище _____
Ім'я _____
По батькові _____
Факультет _____
Група _____
Дата народж. _____
Місце прожив. _____
Загубив книгу:
Автор _____
Назва _____
Ідентифікаційний номер _____
Видавництво _____
Рік видання _____
Ціна _____

Здійснив заміну _____ (дата) :
Автор _____
Назва _____
Видавництво _____
Рік видання _____
Ціна _____

2. Побудувати модель даних, що зображає відношення між об'єктними множинами, складену об'єктну множину та її атрибути:

- студенти вивчають предмети та отримують за них оцінки;
- лекції з предмета проводяться в певній аудиторії в певний час;
- щодня службовці працюють певну кількість годин;
- абоненти здійснюють передплату на журнал на певний термін;
- пілот має деяку кількість годин польоту на кожному з видів літаків.

3. Створити концептуальну модель даних, що складалася б з об'єктних множин, відношень, атрибутів тощо та давала б відповіді на запитання.

- хто із студентів, які відвідали певний факультатив, має з нього відмінну оцінку?
- хто із студентів, котрі відвідали певний факультатив, пише дипломну роботу під керівництвом викладача, який проводив цей факультатив?

4. Створити концептуальну модель даних, що складалася б з об'єктних множин, відношень, атрибутів тощо для ситуації: у ВНЗ існує декілька видів літньої практики (педагогічна, на агробіостанції, ремонт гуртожитків, благоустрій корпусів), кожна з яких проходить у певний термін. Для кожного з таких видів практик потрібне певне обладнання (визначити на власний розсуд). Студенти повинні відвідати декілька таких практик: педагогічну та гуртожиток або корпус, студенти природничого факультету – ще й практику на агробіостанції.

Практична робота № 4
Реляційна модель БД

Мета: набути практичних умінь та навичок створення реляційних схем БД.

Контрольні питання

Пояснити суть наведених термінів та понять:

- два підходи до проектування реляційної бази;
- реляційна модель даних (реляція, атрибут, ступінь реляції, кортеж);
- область атрибута;
- рекурсивне відношення;
- порожнє значення;
- суперключ;
- реляційна схема БД;
- обмежувальні умови;
- правило категорійної цілісності;
- цілісність на рівні посилань;
- функціональна залежність;
- детермінант;
- нормалізація;
- надлишок даних;
- аномалія оновлення;
- аномалія видалення;
- аномалія введення;
- розбиття таблиць;
- атомарне значення;

- 1НФ, 2НФ, 3НФ, 4НФ;
- Нормальна форма Бойса-Кодда;
- транзитивна залежність;
- складений ключ.

Дати порівняльну характеристику та визначити відмінності:

- ключ та суперключ;
- зовнішній ключ та ключ;
- зовнішній ключ та рекурсивний зовнішній ключ;
- атрибути й області атрибутів;
- потенційні ключі та первинні ключі;
- категорійна цілісність та цілісність на рівні посилань.

Пояснити кожне з правил Кодда для РБД.

Завдання

1. Для наведеної таблиці визначити: чи можуть бути для неї дійсними зазначені функціональні залежності. Кожну відповідь обґрунтувати. Визначити ключ.

КОНТИНГЕНТ СТУДЕНТІВ							
Код	Прізвище	Ім'я	По батькові	Дата народж.	Спеціальність	Факультет	Адреса
1	2	3	4	5	6	7	8

а) 1 → 2
б) 5 → 7
в) 6 → 7
г) 2 → 3
ґ) (2,3,4) → 5
д) 6 → 7
е) 1 → 6
є) 1 → 7
ж) 1 → 8
з) 3 → 8
и) 1 → (2,3,4,5)
і) 5 → (2,3,4)
ї) 7 → 6
й) 4 → 2
к) 1 → (2,3,4)
л) 8 → 1

2. Визначити чи дійсні для наведеної таблиці зазначені функціональні залежності. Відповіді обґрунтувати. Визначити ключ таблиці.

A	B	C	D	E
a1	b2	c1	d3	e2
a3	b2	c3	d2	e4
a1	b3	c1	d1	e2
a2	b4	c1	d4	e2

а) A → C
б) E → A
в) D → E
г) C → B
д) C → A
е) B → D
є) E → B
ж) B → A

3. Визначити ті функціональні залежності, що не є дійсними для наведеної таблиці. Відповіді обґрунтувати. Визначити ключ таблиці.

A	B	C	D	E
a1	b2	c1	d3	e2
a2	b2	c3	d3	e4
a1	b3	c2	d1	e2
a2	b4	c5	d1	e5

а) A → C
б) E → A
в) D → E
г) C → B
д) C → A
е) B → D
є) E → B
ж) B → A

4. Визначити, чи є наведені реляції такими, що відповідають 1-4 нормальним формам. Відповіді обґрунтувати. У раз, якщо таблиці не є нормалізованими, розбити їх на декілька. Визначити ключ та зовнішні ключі.

ВИКЛАДАЧІ {Код викладача, Ім'я, Адреса, Телефон, Посада}
ФЗ: Адреса → Телефон

ОДРУЖЕНІ {Код чоловіка, Ім'я_чоловіка, Код_дружини, Ім'я_дружини}
ФЗ: Код_дружини → Ім'я_дружини

УГОДИ {Дата, Клієнт, Товар, Виробник, Країна_Виробника, Торговий_агент}
ФЗ: Клієнт → Торговий_агент
ФЗ: Виробник → Країна_Виробника

Практична робота № 5
Перетворення концептуальної моделі в реляційну

Мета: закріпити практичні вміння та навички складання реляційних схем БД. Набути практичні вміння та навички перетворення різних елементів концептуальної моделі в реляційну схему.

Контрольні питання

Описати етапи перетворення елементів концептуальної моделі в реляційну:

- перетворення об'єктних множин і атрибутів;
- перетворення конкретизацій та узагальнення;
- перетворення відношення один-до-одного;
- перетворення відношення один-до-багатьох;
- перетворення відношення багато-до-багатьох;
- перетворення складених об'єктних множин;
- перетворення рекурсивних відношень.

Порівняти два підходи до моделювання даних.

Завдання

1. Створити реляційні схеми БД для побудованих концептуальних моделей з практичних робіт № 2 та № 3.
2. Створити реляційну схему для наступної ситуації.

Компанія зі страхування автомобілів видає поліси. Для оформлення кожного полісу потрібно знати інформацію:

про власника автомобіля;
про автомобіль;
про агента, який виписав поліс;

номер соціальної страховки власника автомобіля;
номер поліса;
суму щомісячних внесків клієнта та проценти виплат за аварію та за крадіжку автомобіля.

Клієнт може мати декілька полісів, а агент виписувати багато полісів.

Практична робота № 6

Операції реляційної алгебри та конструкції реляційного числення

Мета: набути практичних умінь і навичок виконання запитів засобами реляційної алгебри та реляційного числення.

Контрольні питання

Пояснити суть наведених термінів та понять:

- процедурна та не процедурна мова;
- реляційна алгебра;
- об'єднально-сумісні таблиці;
- перетин;
- об'єднання;
- віднімання;
- декартовий добуток;
- вибірка;
- створення проекції;
- природне поєднання;
- тета-поєднання;
- еквіпоєднання;
- ділення;
- цільовий список;
- визначальний вираз;
- квантор існування;
- квантор загальності.

Завдання

1. Навести приклади використання кожної з операцій реляційної алгебри. Яка з операцій реляційної алгебри не використовується в реляційному численні і чому?
2. Використовуючи наведену реляційну схему:

   ФІРМА (КодФ, Назва, РічнПрибуток)
   ПЕРЕВЕЗЕННЯ (КодП, КодФ, Вага, №Вантажівки, Місто_доставки)
   Зовнішній ключ *КодФ* посилається на ФІРМА

   записати вираз реляційної алгебри для отримання відповідей на запити:
   - які фірми мають річний прибуток більше 1000000 грн?
   - як називається фірма з кодом 100?
   - які вантажівки перевозили вантаж вагою більшою, ніж 5 т?
   - які фірми перевозили вантаж до Дніпропетровська?

   - куди перевозили товари фірми з річним прибутком менше 3000000 грн?
   - яка фірма здійснювала перевезення до всіх міст?
   - який пункт призначення перевезення з кодом 100? Яка вантажівка здійснила це перевезення?

3. Використовуючи наведену реляційну схему:

   ФІРМА (КодФ, Назва, РічнПрибуток)
   ПЕРЕВЕЗЕННЯ (КодП, КодФ, Вага, №Вантажівки, Місто_доставки)
   Зовнішній ключ *КодФ* посилається на ФІРМА
   Зовнішній ключ №Вантажівки посилається на ВАНТАЖІВКА
   Зовнішній ключ Місто_доставки посилається на назва у МІСТО
   ВАНТАЖІВКА (№Вантажівки, Водій)
   МІСТО (Назва, Населення)

   записати вираз реляційної алгебри для отримання відповідей на такі запити:
   - вказати прізвище водія вантажівки № 34-79;
   - вказати міста, куди було перевезено вантаж вагою більшою, ніж 10 т;
   - вказати річний прибуток та назви фірм, що перевозили вантаж вагою більшою за 10 т;
   - назвати імена водіїв, які перевозили вантажі вагою, більшою ніж 5 т;
   - у які міста робили поставки фірми з річним прибутком більше 1000000?
   - назвати фірми з річним доходом, більшим від 5000000, що відправляли вантажі вагою до 1 т;
   - назвати імена водіїв, які здійснювали поставки у всі міста;
   - вказати назву фірми, що перевозила найлегший вантаж;
   - який водій перевозив найважчий вантаж?
   - назвати фірми, що здійснювали поставки у всі міста;
   - назвати фірми, чий вантаж перевозила вантажівка №10-01 до Житомира;
   - назвати фірми, що перевозили вантаж у всі міста з населенням, більшим за 500 000;
   - яка фірма має найбільший річний прибуток?
   - у які міста поставляла вантажі ця фірма?

4. Які інші запити можна запропонувати? Записати вирази в термінах реляційної алгебри для виконання запиту.
5. Які запити можна сформулювати на основі реляційної схеми з другого завдання практичної роботи № 5. Створити не менше 10-15 запитів та записати вирази реляційної алгебри для отримання відповідей на ці запити.
6. Записати у конструкціях реляційного числення основні операції реляційної алгебри.

ЛАБОРАТОРНІ РОБОТИ

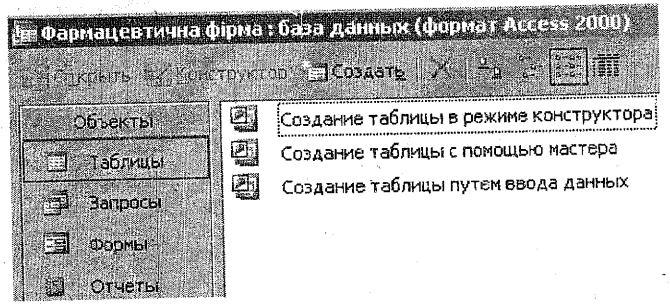
Лабораторна робота №1
Створення БД. Створення та заповнення таблиць. Типи даних.

Мета: навчитися створювати та зберігати нову базу даних; засвоїти основні прийоми роботи з таблицями; опанувати використання різних типів даних в процесі створення таблиць.

Завдання.

1. Завантажити програму Microsoft Access:
   - виконати команду головного меню **Пуск\Програми\Microsoft Office\Microsoft Office Access 2003**.

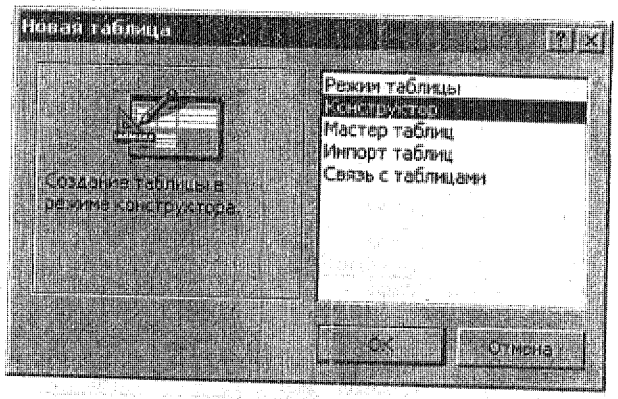
2. Створити нову базу даних та зберегти її на диску А з іменем *Фармацевтична фірма*:
   - виконати команду меню **Файл\Создать**;
   - в області *Создание файла* (права частина вікна) обрати пункт *Новая база данных...* ;
   - у наступному вікні *Файл новой базы данных* зазначити назву бази даних, вказати папку, в якій вона буде зберігатися та натиснути кнопку вікна <*Создать*>;
   - у результаті отримаємо робоче вікно бази даних, частину якого зображено:



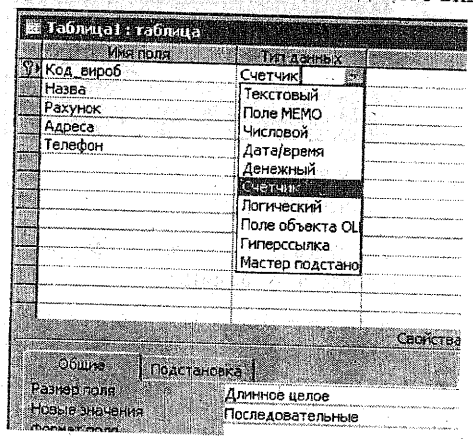
3. Створити таблицю **Виробники** за таким зразком:

	Имя поля	Тип данных
⚷	Код_вироб	Счетчик
	Назва	Текстовый
	Рахунок	Числовой
	Адреса	Поле МЕМО
	Телефон	Текстовый

- обрати серед об'єктів пункт *Таблицы*;
- натиснути кнопку вікна <*Создать*>, після чого з'явиться вікно:



- у вікні *Новая таблица* обрати пункт *Конструктор*;
- натиснути кнопку вікна <*Ok*>;
- у вікні конструктора вказати назви полів у стовпчику *Имя поля* та вибрати із списку типи даних для кожного поля в стовпчику *Тип данных* (у відповідності до наведеного вище зразка):



4. Для поля *Код_вироб* встановити ознаку ключового поля:

- встановити курсор на поле *Код_вироб*;
- на панелі інструментів натиснути кнопку Ключевое поле –  ;
- в результаті злів від назви поля з'явиться зображення ключа[1].

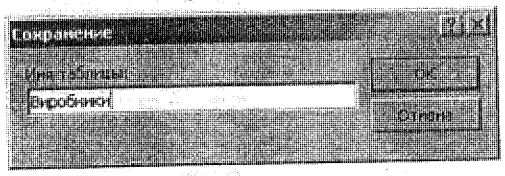
5. Для кожного із полів відповідно до наведеної таблиці на вкладці *Общие* вікна конструктора таблиці вказати такі параметри:

Виробники

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове	Умова на значення	Повідомлення про помилку
Код_вироб	Счетчик	Длинное целое	Код виробника	Да (Совпадения не допускаются)	-	-	-
Назва	Текстовый	40	Назва виробника	Нет	Да	-	-
Рахунок	Числовой	Длинное целое	Розрахунковий рахунок	Нет	Да	>0	Значення повинно бути >0!
Адреса	Поле МЕМО	-	Адреса	Нет	Да	-	-
Телефон	Текстовый	20	Телефон	Нет	Да	-	-

6. Зберегти створену таблицю:

- виконати команду меню **Файл\Сохранить**;
- у вікні *Сохранение* вказати ім'я таблиці – Виробники:



- натиснути кнопку вікна <*Ok*>;
- закрити вікно конструктора таблиці.

У вікні *Фармацевтична фірма: база данных (формат Access 2000)* з'явиться піктограма із назвою створеної таблиці.

7. Аналогічно створити інші таблиці бази даних з параметрами та назвами відповідно до наведених нижче зразків:

[1] Визначити ключове поле можна також за допомогою контекстного меню

Види замовників

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове
Код_виду	Счетчик	Длинное целое	Код виду замовника	Да (Совпадения не допускаются)	-
Вид_замов	Текстовый	25	Вид замовника	Нет	Да

Ключове поле – Код_виду

Замовники

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове	Умова на значення	Повідомлення про помилку
Код_зам	Счетчик	Длинное целое	Код замовника	Да (Совпадения не допускаются)	-	-	-
Назва	Текстовый	40	Назва замовника	Нет	Да	-	-
Адреса	Поле МЕМО	-	Адреса	-	Да	-	-
Телефон	Текстовый	20	Телефон	Нет	Да	-	-
Код_виду_зам	Числовой	Длинное целое	Код виду замовника	Нет	Да		

Для поля *Код_виду_зам* додатково зазначити на вкладці *Подстановка* такі властивості:
тип элемента управления – *Поле со списком*; тип источника строк – *Таблица или запрос*;
источник строк – обрати із списку потрібну таблицю (*Види замовників*);
число столбцов – *2*; заглавия столбцов – *Нет*; ширина столбцов – *0,5 см*.

Ключове поле – Код_зам

Агенти

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове
Код_аг	Счетчик	Длинное целое	Код агента	Да (Совпадения не допускаются)	-
Прізвище	Текстовый	25	Прізвище агента	Нет	Да
Ім'я	Текстовый	25	Ім'я	Нет	Да
По-батькові	Текстовый	25	По батькові	Нет	Нет
Дата_народ	Дата/время	Краткий формат даты	Дата народження	Нет	Нет

Адреса	Поле МЕМО	-	Домашня адреса	Нет	Нет
Тел_роб	Текстовый	20	Робочий телефон	Нет	Нет
Тел_дом	Текстовый	20	Домашній телефон	Нет	Нет
Стать	Текстовий	1	Стать	Нет	Нет
Для поля *Стать* додатково зазначити на вкладці *Подстановка* такі властивості: тип элемента управления – *Список*; тип источника строк – *Список значений*; источник строк – *Ж; Ч* (визначити самостійно). На вкладці *Общие* у полі *Условие на значение* зазначити умову *"Ж" or "Ч"*; у полі *Сообщение об ошибке* записати *Введене значення неправильне.*					
Фото	Поле объекта OLE	-	Фото	-	Нет

Ключове поле – Код_аг

Група препаратів

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове
Назва_гр	Текстовый	25	Група	Да (Совпадения не допускаются)	-

Ключове поле – Назва_гр

Препарати

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове
Код_преп	Счетчик	Длинное целое	Код препарату	Да (Совпадения не допускаются)	-
Назва_преп	Текстовый	50	Назва препарату	Нет	Да
Група_преп	Текстовый	25	Група	Нет	Да
Для поля *Група_преп* додатково зазначити на вкладці *Подстановка* такі властивості: тип элемента управления – *Поле со списком*; тип источника строк – *Таблица или запрос*; источник строк – обрати із списку потрібну таблицю (*Група препаратів*); число столбцов – *1*; заглавия столбцов – *Нет*; ширина столбцов – *0,9 см;* число строк списка – *10* .					

Ключове поле – Код_преп

Види розрахунків

Поле	Тип даних	Розмір (формат)	Підпис	Індексоване	Обов'язкове
Код_виду	Счетчик	Длинное целое	Код виду розрахунку	Да (Совпадения не допускаются)	-
Вид_роз	Текстовый	25	Вид розрахунку	Нет	Да

Ключове поле – Код_виду

Угоди

Поле	Тип даних	Розмір (формат)	Кіл-ть десятк. знаків	Підпис	Індексоване	Обов'язкове	Умова на значення	Повідомлення про помилку
Код_уг	Счетчик	Длинное целое	-	Угода №	Да (Совпадения не допускаются)	-	-	-
Дата	Дата/время	Краткий формат даты	-	Дата укладання угоди	Нет	Да	-	-
Код_зам	Числовой	Длинное целое	-	Код замовника	Нет	Да	-	-
Для поля *Код_зам* додатково зазначити на вкладці *Подстановка* такі властивості: тип элемента управления – *Поле со списком*; тип источника строк – *Таблица или запрос*; источник строк – обрати із списку потрібну таблицю (*Замовники*); число столбцов – *2*; заглавия столбцов – *Нет*; ширина столбцов – *0,5 см;* число строк списка – 10; ширина списка – 3 *см*: Подібні значення вказати і для двох наступних полів та полів Код_аг, Код_вид_ роз (враховувати назву таблиці, з якої обиратимуться значення для заповнення цих полів та кількість рядків списку).								
Код_преп	Числовой	Длинное целое	-	Код препарату	Нет	Да	-	-
Код_вир	Числовой	Длинное целое	-	Код виробника	Нет	Да	-	-
Ціна_од	Денежный	Денежный	2	Ціна за диницю товару	Нет	Да	-	-
Кількість	Числовой	Целое	0	Кількість одиниць товару	Нет	Да	-	-
Код_аг	Числовой	Длинное целое	-	Код агента	Нет	Да	-	-
Код_вид_роз	Числовой	Длинное целое	-	Код виду розрахунку	Нет	Да	-	-
Оплата	Логический	Да/Нет	-	Оплачено (Так/Ні)	Нет	Да	-	-

Ключове поле – Код_уг

8. Заповнити таблиці наведеними нижче даними[1].

Вставка даних в останній стовпчик *Фото* (тип даних – Поле об'єкту OLE) проводиться наступним чином:

- встановити курсор на полі *Фото*;
- виконати команду меню **Вставка/Объект…**
- у наступному вікні зазначити *Создать из файла* та за допомогою кнопки <*Обзор*> вибрати фотографію;
- натиснути кнопку вікна <*Ок*>, у результаті чого у полі Фото з'явиться напис *Точечный рисунок*.

Агенти

Код агента	Прізвище агента	Ім'я	По батькові	Дата народження	Домашня адреса	Робочий телефон	Домашній телефон	Стать	Фото
1	Іванов	Іван	Іванович	03.09.71	м. Житомир, вул. Щорса, 1	22-14-56	25-68-01	Ч	
2	Носенко	Ірина	Іванівна	20.02.67	м. Житомир, вул. Космонавтів, 14а, кв. 27	33-15-76	33-56-09	Ж	
3	Портнов	Євген	Петрович	13.12.79	м. Житомир, вул. Київська, 17, кв. 20	22-14-51	22-66-93	Ч	
4	Сомов	Петро	Іванович	04.11.76	м. Житомир, вул. Жукова, 19, кв. 48	33-16-16	24-09-09	Ч	
5	Громова	Раїса	Петрівна	01.01.66	м. Житомир, вул. Малікова, 83, кв. 10	24-14-62	22-88-69	Ж	
6	Бендер	Остап	Ібрагімович	27.03.70	м. Житомир, вул. Космонавтів, 8, кв. 54	33-15-60	34-20-08	Ч	

Після введення даних закрити вікно таблиці.

Види замовників

Код виду замовника	Вид замовника
1	Державне підприємство
2	Приватне підприємство

[1] Перед заповненням таблицю потрібно відкрити подвійним клацанням лівою кнопкою миші на імені таблиці (або вибрати потрібну таблицю та натиснути кнопку вікна <*Открыть*>).

Види розрахунків

Код виду розрахунку	Вид розрахунку
1	Безготівковий
2	Готівкою

Виробники

Код виробника	Назва виробника	Розрахунковий рахунок	Адреса	Телефон
1	Концерн "Стирол"	1234578	м. Донецьк, вул. Леніна, 123	(068)4567890
2	Фірма "Дарниця"	778927462	м. Київ, пр. Миру, 5	(044)5673829
3	Борщагівський фармзавод	1999850021	м.Київ, вул. Борщагівська , 13	264-33-87
4	ЗАТ "Ліктрави"	0235014560	м. Житомир, вул. Промислова, 47	22-77-13
5	ВАТ "Здоров'я"	1223542209	м. Вінниця, вул. Чапаєва, 29	42-17-54

Група препаратів

Група	Група
Лікарські трави	Жарознижуючі
Анальгетики	Шлункові
Сульфаніламіди	Антибіотики
Вітаміни	Заспокійливі
Серцеві	Гормональні

Замовники

Код замовника	Назва замовника	Адреса	Телефон	Код виду замовника
1	Аптека №1			1
2	Аптека № 33			1
3	Діагностичний центр			1
4	Зелена аптека			2
5	Глобал-тренд			2
6	Аптека № 15			1
7	Ескулап			2
8	Лікарські трави			1
9	Обласна лікарня			1

Адреси та телефони заповнити на власний розсуд.

Препарати

Код препарату	Назва препарату	Група	Код препарату	Назва препарату	Група
1	Пеніцилін	Антибіотики	12	Мезим	Шлункові
2	Гендевіт	Вітаміни	13	Жанін	Гормональні
3	Анальгін	Анальгетики	14	Валідол	Серцеві
4	Цитрамон	Анальгетики	15	Вікасол	Вітаміни
5	Гідрокортизон	Гормональні	16	Валеріана	Заспокійливі
6	Ромашка	Лікарські трави	17	Бісептол	Сульфаніламіди
7	Три-квілар	Гормональні	18	Ампіокс	Антибіотики
8	Парацетамол	Жарознижуючі	19	Диметоксин	Сульфаніламіди
9	Шавлія	Лікарські трави	20	Кора дуба	Лікарські трави
10	Пенталгін	Анальгетики	21	Панкреатин	Шлункові
11	Аспірин	Анальгетики	22	Корвалол	Серцеві

Угоди

Угода №	Дата укладання угоди	Код замовника	Код препарату	Код виробника	Ціна за одиницю товару	Кількість одиниць товару	Код агента	Код виду розрахунку	Оплачено (Так/Ні)
1	23.12.2007	6	1	3	1,00	50	6	1	√
2	05.05.2006	5	5	4	2,00	300	4	1	√
3	09.01.2001	1	10	2	3,80	100	3	2	
4	09.09.2007	3	5	1	0,65	150	6	2	√
5	09.09.2007	2	7	5	3,50	130	6	1	

9. Самостійно доповнити таблицю **Угоди** до 40 записів.

Лабораторна робота № 2
Створення зв'язків між таблицями. Схема даних.

Мета: навчитися створювати зв'язки між таблицями, зазначати тип відношень між ними із забезпеченням цілісності даних.

Завдання.

1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

2. Відкрити вікно *Схема данных* та обрати таблиці, які потрібно зв'язати між собою:

- виконати команду меню **Сервис/Схема данных...** або натиснути на панелі інструментів кнопку ;

- у вільному від об'єктів місці активізувати контекстне меню та обрати пункт *Добавить таблицу...*;
- у вікні *Добавление таблицы* на вкладці *Таблицы* обрати таблицю *Замовники* та натиснути кнопку вікна *<Добавить>*;
- так само додати таблицю *Види замовників* та натиснути кнопку вікна *<Закрыть>*;
- після додавання таблиць у вікні схеми даних з'явиться їхнє зображення.

3. Встановити зв'язок між двома доданими таблицями по атрибутах *Код_виду* та *Код_виду_зам*:

- виділити лівою кнопкою миші поле *Код_виду_зам* таблиці *Замовники*);
- утримуючи кнопку миші, перемістити курсор на поле Код_виду таблиці Види замовників;
- у вікні *Изменение связи* впевнитися в правильності встановлення зв'язку (імена таблиць та атрибутів);
- встановити ознаки: *Обеспечение целостности данных, каскадное обновление связанных полей, каскадное удаление сзязанных полей*;
- візуально зв'язок відобразиться у вигляді лінії між відповідними полями зв'язаних таблиць із зазначенням типу відношення.

4. Зберегти створену схему даних:

- виконати команду меню **Файл/Сохранить**.

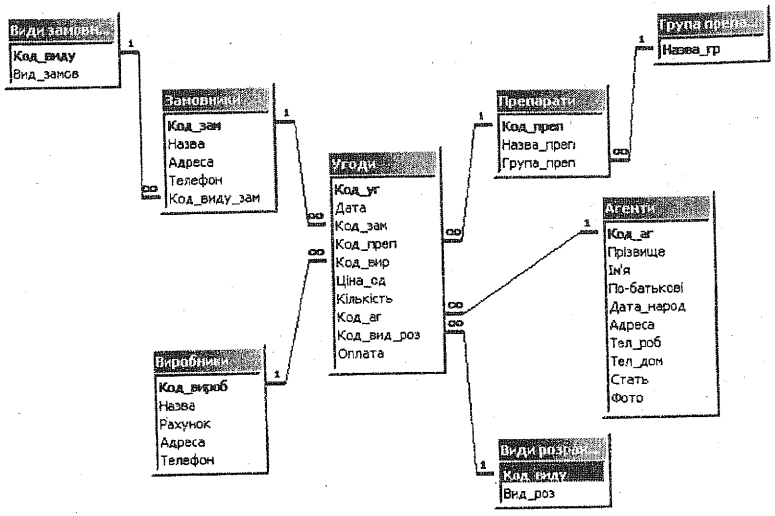
5. Закрити вікно схеми даних.

6. Додати до схеми даних таблицю Виробники:

- відкрити вікно схеми даних;
- на панелі інструментів натиснути кнопку *Отобразить таблицу* (або у контекстному меню обрати пункт *Добавить таблицу*);
- у вікні *Добавление таблицы* обрати таблицю *Виробники*;
- натиснути кнопку вікна *<Добавить>*.

7. Аналогічно додати до схеми даних всі інші таблиці бази даних (у вікні *Добавление таблицы* можна виділити декілька таблиць і додати їх одночасно). **Увага!** Одну і ту саму таблицю до схеми додавати не можна. Щоб вилучити з схеми даних помилково занесену таблицю, потрібно цю таблицю виділити та натиснути на клавіатурі *<Delete>* або виконати команду меню **Правка/Удалить**.

8. Користуючись вже відомим методом з'єднання таблиць, створити зв'язки між всіма таблицями бази даних за такою схемою:



9. Зберегти створену схему даних та закрити вікно додатка.

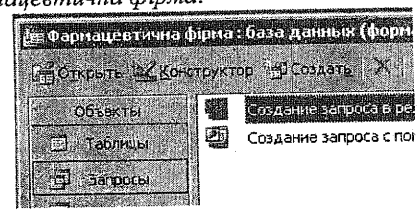
Лабораторна робота № 3

Створення запитів. Сортування даних. Побудова виразів у запитах.

Мета: навчитися створювати на основі однієї або декількох таблиць запити різними методами, засвоїти прийоми сортування даних та оволодіти навичками побудови виразів у запитах.

Завдання.

1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

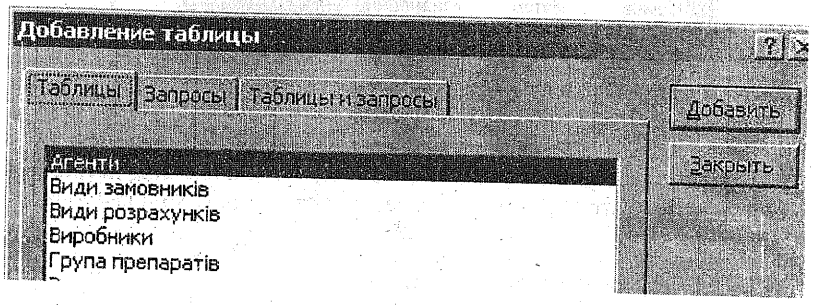


2. У вікні бази даних обрати об'єкт *Запросы.*

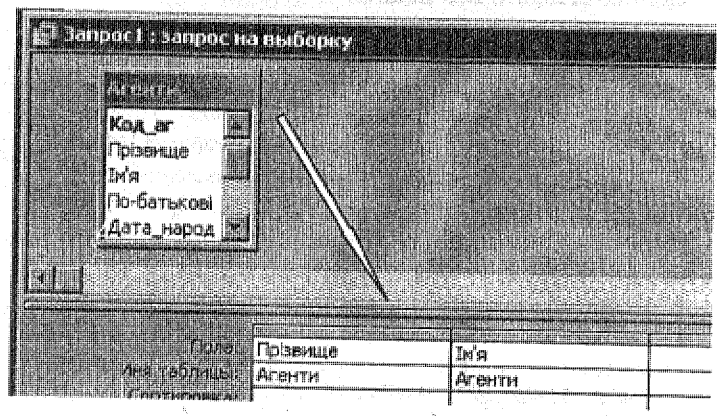
3. Створити за допомогою конструктора запит, який обирав би прізвища, ім'я, по батькові усіх агентів, та їхні фото (на основі цього запиту у подальшому буде сформовано звіт). Запит будується на снові таблиці **Агенти** (атрибути *Прізвище, Ім'я, По батькові, Фото*):

- натиснути кнопку вікна <*Создать*>;

- у вікні *Новый запрос* обрати пункт *Конструктор* та натиснути кнопку вікна <*Ok*>;
- у вікні конструктора запитів потрібно обрати таблицю **Агенти**[1], з якої буде робитися вибірка даних, та натиснути кнопку вікна <*Добавить*>, а потім кнопку вікна <*Закрыть*>;



- у нижню частину вікна конструктора запитів потрібно лівою кнопкою миші перетягнути обрані для запиту атрибути (*Прізвище, Ім'я, По батькові, Фото*);



- після того, як усі атрибути будуть відображені у чотирьох стовпчиках нижньої частини вікна, потрібно протестувати запит. Для цього потрібно натиснути кнопку ! на панелі інструментів;
- на екрані відобразиться результат запиту:

[1] У разі потреби можна обрати декілька таблиць (натискаючи одночасно із вибором таблиці клавішу *Ctrl* клавіатури) та вставити їх одночасно.

Запрос1 : запрос на выборку

Прізвище	Ім'я	По-батькові	Фото
Іванов	Іван	Іванович	Точечный рисунок
Носенко	Ірина	Іванівна	Точечный рисунок
Портнов	Євген	Петрович	Точечный рисунок
Сомов	Петро	Іванович	
Громова	Раїса	Петрівна	Точечный рисунок
Бендер	Остап	Ібрагімович	Точечный рисунок
Вознюк	Тетяна	Григорівна	Точечный рисунок

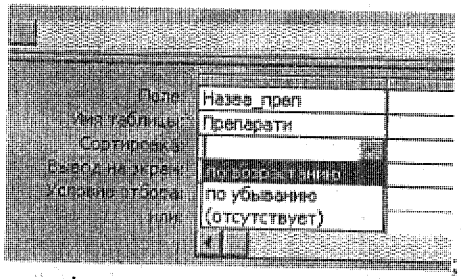
;

- якщо результат виконання запиту неправильний, потрібно змінити його структуру – повернутися до вікна конструктора запиту, натиснувши кнопку панелі інструментів.

4. Зберегти створений запит:
   - обрати пункт меню **Файл/Сохранить**;
   - у вікні *Сохранение* ввести назву запиту – *Фото агентів* та натиснути кнопку вікна <*Ok*>.
5. Аналогічно створити такі запити:
   - *Перелік замовників* (на основі таблиці **Замовники**, атрибути *Назва замовника, Адреса, Телефон*;
   - *Рахунки виробників* (на основі таблиці **Виробники**, атрибути *Назва виробника, Розрахунковий рахунок*;
   - *Робочі телефони агентів* (на основі таблиці **Агенти**, атрибути *Прізвище, Ім'я, По батькові, Робочий телефон*.
6. Створити запит *Перелік препаратів* методом "Простий запит":
   - обрати об'єкт *Запросы* та натиснути кнопку вікна <*Создать*>;
   - у вікні *Новый запрос* обрати пункт *Простой запрос* та натиснути кнопку вікна <*Ok*>;
   - у полі *Таблицы и запросы* обрати таблицю **Препарати**;
   - у наступному вікні серед доступних полів обрати потрібний атрибут *Назва_преп* та натиснути кнопку вікна >;[1]
   - натиснути кнопку вікна <*Далее*>;
   - вказати ім'я запиту – *Перелік препаратів* та натиснути кнопку вікна <*Готово*>.
7. Для запиту *Перелік препаратів* встановити ознаку сортування назви в алфавітному порядку (за зростанням):
   - обрати потрібний запит – *Перелік препаратів*;
   - натиснути кнопку (*Конструктор*);

[1] Для того, щоб обрати усі доступні поля одночасно, потрібно натиснути кнопку вікна >>.

- у вікні конструктора встановити ознаку сортування за зростанням:



;

- зберегти зміни, закрити вікно конструктора.

8. Для запиту *Робочі телефони агентів* встановити ознаку сортування в алфавітному порядку по атрибуту *Прізвище*.

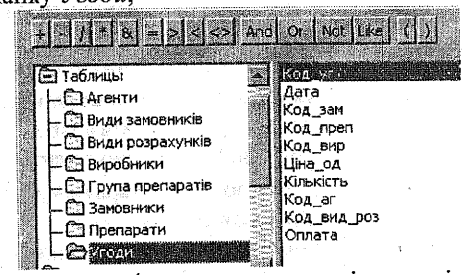
Попередні звіти створювалися на основі даних однієї таблиці. Розглянемо випадок, коли звіт створюється на основі даних декількох таблиць.

9. Створити запит *Замовники за видом*, який містить інформацію про назву та вид замовника. Відсортувати дані за ознакою виду:
   - проаналізувати, з яких таблиць обиратимуться дані для запиту (**Замовники та Види замовників**);
   - за допомогою конструктора створити новий запит;
   - додати потрібні таблиці **Замовники та Види замовників** (зв'язки обраних таблиць відобразяться автоматично, якщо ці зв'язки були зазначені на схемі даних);
   - обрати атрибут *Вид_замов* з таблиці **Види замовників** та атрибут *Назва* з таблиці **Замовники**;
   - встановити сортування за видом замовників (за зростанням) та вторинну ознаку сортування назви замовника (за зростанням);
   - протестувати створений запит (кнопка на панелі інструментів);
   - зберегти запит з ім'ям *Замовники за видом*.
10. Аналогічно на основі декількох таблиць створити такі запити:
    - *Оплата угод замовниками* (містить інформацію про назву замовника, дату оформленої ним угоди та ознаку оплати; встановити сортування за назвою замовника);
    - *Агент-угода* (містить інформацію про прізвище, ім'я та по батькові агента, дату оформленої ним угоди та назву препарату; встановити сортування за прізвищем агента та вторинну ознаку сортування за назвою препаратау);
11. На основі декількох таблиць побудуємо запит, який буде рахувати загальну суму по кожній угоді (на основі цього запиту у подальшому буде створено звіт-накладну):
    - за допомогою конструктора створити новий запит;

- додати таблиці *Угоди*, *Замовники*, *Препарати*, *Виробники*, *Агенти*;
- обрати до запиту атрибути: *Код_уг* та *Дата* (таблиця *Угоди*), *Прізвище* (таблиця *Агенти*), *Назва* (таблиця *Замовники*), *Назва_преп* (таблиця *Препарати*), *Назва* (таблиця *Виробники*), *Ціна_од* та *Кількість* (таблиця *Угоди*).

У наступному вільному стовпчику запиту запишемо вираз для підрахунку загальної суми угоди за даними полів *Ціна_од* та *Кількість* таблиці *Угоди*:

- встановити курсор у наступний вільний стовпчик;
- для побудови запиту натиснути кнопку (*Построить*);
- у вікні *Построитель выражений* відкрити папку *Таблицы*, а в ній папку *Угоди*;



- двічі клацнути лівою кнопкою миші по назві поля *Ціна_од*, після чого у області побудови виразу з'явиться напис [Угоди]![Ціна_од];
- серед пропонованих операцій обрати та натиснути знак множення (кнопка вікна <*>);
- двічі клацнути лівою кнопкою миші по назві поля *Кількість;*
- натиснути кнопку вікна <*Ok*>;
- у стовпчику буде записано вираз для підрахунку: Выражение1: [Угоди]![Ціна_од]*[Угоди]![Кількість];
- з клавіатури замінити текст Выражение1 на текст Загальна сума;
- протестувати запит, переконатися у правильності розрахунків;
- зберегти створений запит з ім'ям *Загальна сума*.

12. Створити аналогічний запит *Товарна накладна* (або *Чек*).

13. На основі таблиць *Угоди*, *Замовники*, *Види замовників* та запиту *Загальна сума* побудувати запит *Угоди та сума за видом замовників*, який містив би інформацію про види замовників, дату, номер та загальну суму укладеної угоди. Встановити первинну ознаку сортування в алфавітному порядку за видом замовника та вторинну ознаку сортування (зростання) за датою і загальною сумою укладеної угоди.

14. Завершити роботу з додатком.

Лабораторна робота № 4
Використання умов відбору та групових операцій в запитах. Використання мови SQL для побудови запитів

Мета: оволодіти навичками роботи із полям, що мають тип *Дата-Час*; навчитися створювати запити із зазначенням умов відбору та групових операцій, зокрема, використовуючи мову SQL.

Завдання.

1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

2. Створити запит *Перелік жарознижуючих препаратів*, який містив би назви усіх наявних жарознижуючих препаратів (відсортувати за зростанням):

- у вікні бази даних обрати об'єкт *Запросы*;
- натиснути кнопку вікна <*Создать*>;
- у вікні *Новый запрос* обрати пункт *Конструктор* та натиснути кнопку вікна <*Ok*>;
- додати таблицю **Препарати**, з якої буде робитися вибірка;
- у нижню частину вікна конструктора запитів потрібно лівою кнопкою миші перетягнути атрибути *Назва_преп* та *Група_преп*;
- для атрибуту *Група_преп* у полі *Условие отбора* записати умову відбору *="жарознижуючі"* та зняти прапорець у полі *Вывод на экран* (як показано на малюнку);

Поле:	Назва_преп	Група_преп
Имя таблицы:	Препарати	Препарати
Сортировка:		
Вывод на экран:	☑	☐
Условие отбора:		="жарознижуючі"

- встановити сортування за зростання для поля *Назва_преп*;
- протестувати запит (кнопка на панелі інструментів);
- зберегти створений запит з іменем *Перелік жарознижуючих препаратів.*

3. Аналогічно створити запит *Перелік гормональних препаратів.*

4. Створити подібний запит *Перелік лікарських трав*, використовуючи мову SQL:

- натиснути кнопку вікна <*Создать*>;
- вибрати режим конструктора та не додавати жодної таблиці;
- виконати команду меню **Вид/Режим SQL**;
- ввести у вікно запиту інструкцію SQL:

```
SELECT Назва_преп
FROM Препарати
```

```
WHERE Група_преп="Лікарські трави"
ORDER BY Назва_преп
```

5. Аналогічно мовою SQL створити запити *Перелік вітамінів* та *Перелік серцевих препаратів.*

6. Створити запит *Агент-угода більше 1000 грн.*, який виводив би дані (прізвище, ім'я, по батькові, робочий телефон) агентів, що заключили угоди на суму, більшу за 1000 грн., а також номер угоди та загальну суму:

- у вікні бази даних обрати об'єкт *Запросы*;
- натиснути кнопку вікна *<Создать>*;
- у вікні *Новый запрос* обрати пункт *Конструктор* та натиснути кнопку вікна *<Ok>*;
- додати таблиці **Агенти**, **Угоди**, та запит *Загальна сума*;
- у нижню частину вікна конструктора запитів потрібно лівою кнопкою миші перетягнути атрибути *Прізвище, Ім.я, По батькові, Тел_роб* (з таблиці **Агенти**), атрибут *Код_уг* (з таблиці **Угоди**) та атрибут *Загальна сума* (із запиту *Загальна сума)*;
- для останнього атрибуту у полі *Условие отбора* записати умову відбору >1000;
- протестувати запит та зберегти його з ім'ям *Агент-угода більше 1000 грн.*

7. Аналогічно створити запит *Замовник-угода менше 500 грн.*, який виводив би назви замовників, що уклали угоди загальною сумою менше 500 грн., номери угод, дати їх укладання та загальну суму угоди.

8. Створити запит *Оплачені угоди за останній рік*, який містив би таку інформацію про усі оплачені минулорічні угоди – номер та дата угоди, назва замовника та назва препарату:

- у вікні бази даних обрати об'єкт *Запросы*;
- натиснути кнопку вікна *<Создать>*;
- у вікні *Новый запрос* обрати пункт *Конструктор* та натиснути кнопку вікна *<Ok>*;
- додати таблиці **Угоди**, **Замовники**, **Препарати**;
- у нижню частину вікна конструктора перенести потрібні атрибути *Код_уг*, *Дата* (з таблиці **Угоди**), атрибут *Назва* (з таблиці **Замовники**), атрибут *Назва_преп (з таблиці* **Препарати**) та атрибут *Оплата* (з таблиці **Угоди**);
- для останнього атрибуту у полі *Условие отбора* записати умову відбору =Истина;
- для атрибуту *Дата* у полі *Условие отбора* записати умову відбору: Between #01.01.2007# And #31.12.2007#;
- протестувати запит та зберегти його з ім'ям *Оплачені угоди за останній рік.*

9. Створити запит *Оплачені угоди за вересень-2007*, використовуючи мову SQL. Запит повинен містити інформацію про номери та дату оплачених угод за вересень поточного року:

- натиснути кнопку вікна *<Создать>*;
- вибрати режим конструктора та не додавати жодної таблиці;
- виконати команду меню **Вид/Режим SQL**;
- ввести у вікно запиту інструкцію SQL:

```
SELECT Код_уг, Дата, Оплата
FROM Угоди
WHERE Month (Дата)=09 and
Year (Дата)=2007 and Оплата = True
```

10. Створити запит *Оплачені угоди в день запиту*, використовуючи мову SQL. Запит повинен містити інформацію про номери та дату оплачених угод на день запиту.

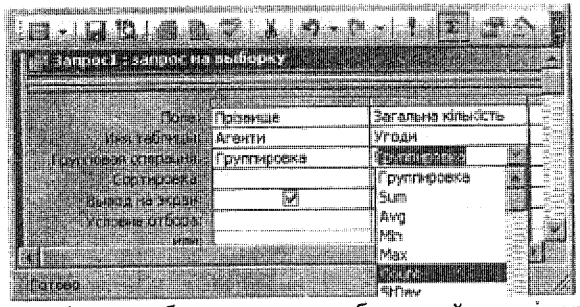
11. Використовуючи мову SQL, створити запит на основі даних декількох таблиць *Агенти-неоплачені угоди*, який містив би прізвища, імена, по батькові, адреси та робочі телефони агентів, що здійснили угоди, які є неоплаченими, а також номери цих угод. Встановити сортування в алфавітному порядку атрибуту *Прізвище*, та ознаку вторинного сортування за зростанням атрибуту *Код_уг*:

```
SELECT Агенти.Прізвище, Агенти.[Ім'я], Агенти.[По-батькові],
       Агенти.Адреса, Агенти.Тел_роб, Угоди.Код_уг, Угоди.Оплата
FROM Агенти INNER JOIN Угоди
            ON Агенти.Код_аг = Угоди.Код_аг
WHERE Угоди.Оплата=FALSE
ORDER BY Агенти.Прізвище, Угоди.Код_уг;
```

12. Аналогічно, використовуючи мову SQL створити запит *Замовник-оплачені угоди*, який містив би назви та телефони замовників, що здійснили оплачені угоди, а також номери та дати цих угод. Встановити ознаку сортування атрибуту *Назва* (за спаданням), та ознаку вторинного сортування атрибуту *Код_уг* (за зростанням).

13. Створити запит *Кількість угод агентів*, який містив би інформацію про загальну кількість угод, проведених кожним агентом:

- створити запит за допомогою конструктора та вставити таблиці **Агенти** та **Угоди**;
- додати поля *Прізвище* (таблиця **Агенти**), та *Код_уг* (таблиця **Угоди**);
- у нижній частині конструктора перейменувати атрибут *Код_уг* на *Загальна кількість:Код_уг*;
- на панелі інструментів натиснути кнопку Σ (*Групповые операции*);
- у рядку *Групповая операция* для стовпчика *Загальна кількість* обрати операцію *Count*;



- перевірити роботу запиту та зберегти його з іменем *Кількість угод агентів*.

14. Використовуючи групові операції, створити у режимі конструктора запит *Кількість угод замовників*. У результаті запиту потрібно отримати перелік назв замовників (відсортованих в алфавітному порядку), вид замовника та кількість здійснених ним угод.

15. Використовуючи групові операції, створити у режимі конструктора-запит *Кількість неоплачених угод замовників*. У результаті запиту потрібно отримати перелік назв замовників (в алфавітному порядку), номери їх телефонів та кількість неоплачених ними угод. Атрибут *Оплата* відображати у запиті не потрібно.

16. Засобами SQL створити запит *Максимальна та мінімальна суми замовлення*.

```
SELECT MAX ([Загальна сума])
          AS [Максимальна сума замовлення],
       MIN ([Загальна сума])
          AS [Мінімальна сума замовлення]
FROM [Загальна сума];.
```

17. Використовуючи конструктор, створити запит *Загальна сума продажу кожного виробника*, який містив би інформацію про загальну суму продажу препаратів кожного з виробників.

18. Використовуючи конструктор створити запит, *Кількість проданих препаратів*, який містив би інформацію про загальну кількість продажу кожного препарату.

19. Використовуючи групові операції, засобами SQL створити запит *Товари, загальна сума продажу яких більша 1000.*

```
SELECT
        Препарати.Назва_преп, Sum([Загальна сума].[Загальна сума])
        AS [Загальна сума продажу]
FROM
     (Препарати INNER JOIN Угоди
                    ON Препарати.Код_преп = Угоди.Код_преп)
                INNER JOIN [Загальна сума]
                    ON Угоди.Код_уг =[Загальна сума].Код_уг
GROUP BY Препарати.Назва_преп
Having  Sum([Загальна сума].[Загальна сума])>1000 ;.
```


20. Записати мовою SQL та протестувати усі запити, розглянуті на лекції, що використовують функції для роботи з полями типу *Дата-Час*.

21. Модифікувати запит *Загальна сума*:

- відкрити запит *Загальна сума* та зберегти його з іменем *Товарна накладна*;
- перейти в режим SQL;
- змінити запит таким чином, щоб виводилася уся інформація про угоду (включаючи і загальну суму) для угоди із конкретним номером за запитом користувача.

21. Завершити роботу з додатком.

Лабораторна робота № 5
Побудова форм за допомогою майстра та конструктора форм. Використання форм.

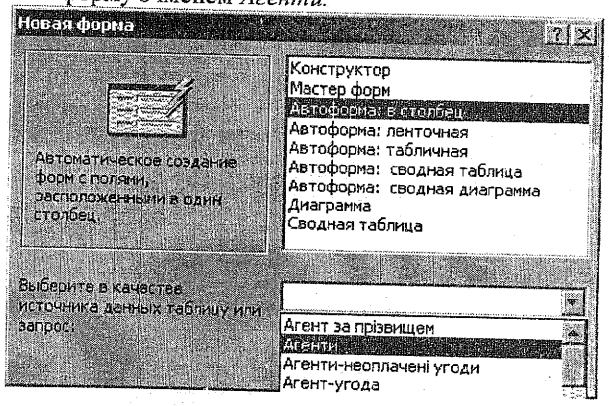
Мета: засвоїти прийоми створення нових та модифікації готових форм різними способами, перегляду та заповнення форм.

Завдання.

1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

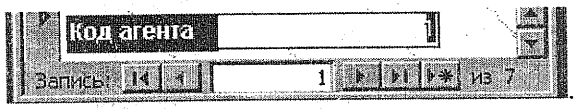
2. За допомогою автоформ створити автоформу *Агенти*:

- у вікні бази даних обрати об'єкт *Формы*;
- натиснути кнопку вікна *<Создать>*;
- вибрати режим *Автоформа: в столбец* (див. малюнок);
- із списку запитів та таблиць обрати таблицю **Агенти**;
- натиснути кнопку вікна *<Ok>*;
- у результаті з'явиться вікно форми, яке потрібно закрити;
- на пропозицію зберегти форму дати ствердну відповідь та зберегти форму з іменем *Агенти.*



3. Відкрити та переглянути створену форму:

- двічі клацнувши по назві форми лівою кнопкою миші;
- для перегляду наступних записів та додавання нових використати кнопки, що знаходяться у нижній частині вікна:



- додати декілька нових записів, скориставшись кнопкою вікна ▶*.

4. Аналогічно створити та доповнити форми (на основі однойменних таблиць):

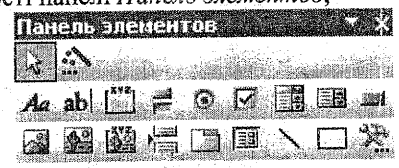
- *Замовники* (Автоформа: в столбец);
- *Препарати* (Автоформа: ленточная).

5. Використовуючи режим майстра форм, створити та доповнити форму *Виробники*, що базується на таблиці **Виробники**:

- натиснути кнопку вікна <*Создать*>;
- вибрати режим *Мастер форм*;
- із списку запитів та таблиць обрати таблицю **Виробники**;
- натиснути кнопку вікна <*Ok*>;
- у вікні *Создание форм* серед доступних полів обрати всі (кнопка вікна >>);
- натиснути кнопку вікна <*Далее*>;
- обрати зовнішній вид форми *В один столбец*;
- натиснути кнопку вікна <*Далее*> та зазначити стиль *Стандартный*;
- натиснути кнопку вікна <*Далее*> та ввести назву форми – *Виробники*;
- натиснути кнопку вікна <*Готово*>;
- додати декілька нових записів.

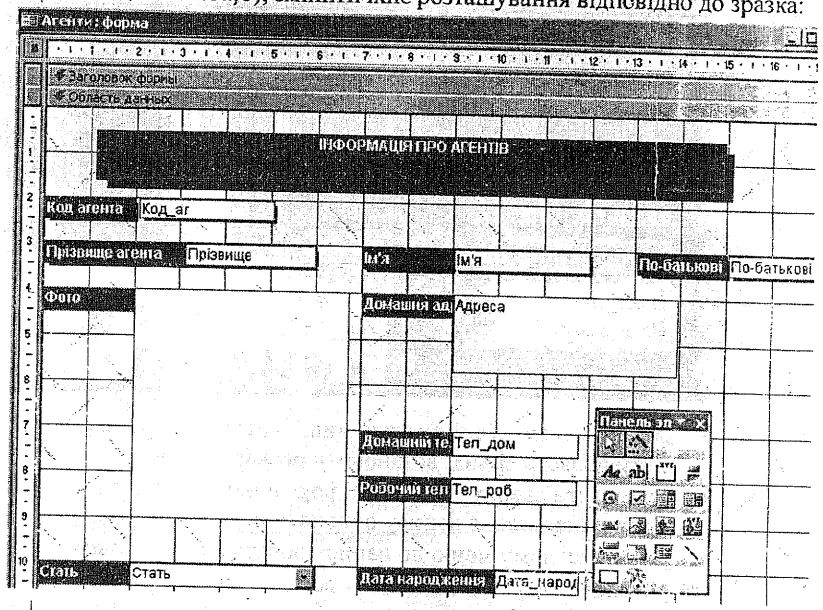
6. Змінити дизайн форми *Агенти*, використовуючи конструктор форм:

- на вкладці *Формы* обрати форму *Агенти*;
- натиснути кнопку вікна  (*Конструктор*) та переконатися у наявності панелі *Панель элементов*;



- у разі відсутності цієї панелі, скористатися кнопкою на панелі інструментів 

- обравши кнопку панелі елементів Aa, вставити новий напис – *Інформація про агентів*;
- перетягуючи кожен із наявних елементів форми (*Прізвище агента, Ім'я тощо*), змінити їхнє розташування відповідно до зразка:



- у контекстному меню встановити параметр *Оформление с тенью* до наступних елементів форми: *Інформація про агентів, Код агента, Прізвище агента, Ім'я, по батькові.*
- за бажанням змінити колір заливки фону та шрифтів кожного з елементів (використовуючи контекстне меню);
- виконати у меню команду **Вид/Режим формы**, переконатися у зміненому дизайні форми, внести один запис та зберегти форму.

7. Використовуючи режим конструктора, створити нову форму *Види замовників* на основі однойменної таблиці:

- обрати об'єкт *Формы*;
- обрати пункт *Создание формы в режиме конструктора*;
- натиснути кнопку вікна *Создать*;
- обрати режим конструктора та зазначити таблицю для побудови форми;
- у результаті з'явиться вікно конструктора, панель елементів та панель з переліком полів таблиці:

- за допомогою лівої кнопки миші методом перетягуванні збільшити область даних: встановити розмір 4х9 см;
- розмістити зверху по центру форми напис *Види замовників.* Колір та оформлення напису виконати на власний розсуд;
- у контекстному меню до напису обрати пункт *Свойства*;
- у діалоговому вікні *Надпись* встановити такі параметри тексту: шрифт Times New Roman, напівжирний, розмір 14 пт, розташування тексту – у центрі напису; закрити діалогове вікно;
- з панелі полів таблиці перенести поля *Код_виду* та *Вид_замов*, розташувавши їх один під одним;
- виконавши команду меню **Вид/Режим формы**, переглянути створену форму та повернутися знову в режим конструктора;
- змінити розміри елементів *Код виду замовника* та *Вид замовника* такими, щоб і назва поля і його уміст повністю відображалися;
- ще раз переглянути та зберегти форму.

8. Аналогічно за допомогою конструктора створити форми *Види розрахунків* та *Група препаратів* на основі однойменних таблиць.

9. Створити та оформити форму *Угоди* довільним способом (форма повинна містити усі поля та заголовок).

10. Завершити роботу з додатком.

Лабораторна робота № 6
Створення звітів на основі таблиць та запитів

Мета: навчитися створювати звіти на основі даних таблиць та запитів.

Завдання.

1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

2. Перейти до об'єктів *Отчеты.*

3. За допомогою майстра створити звіт для виведення інформації про препарати у вигляді таблиці:

- встановити курсор на пункті *Создание отчета с помощью мастера*;
- натиснути кнопку вікна <*Создать*>;
- у вікні *Новый отчет* обрати пункт *Мастер отчетов*, зазначити таблицю **Препарати** та натиснути кнопку вікна <*Ok*>;
- обрати усі поля, крім поля *Код_преп* та натиснути кнопку <*Далее*>;
- у наступному вікні рівні групування визначати не потрібно, натиснути кнопку вікна <*Далее*>;
- у наступному вікні зазначити сортування за зростанням поля *Назва_преп* та натиснути кнопку <*Далее*>;
- у наступному вікні обрати макет *По левому краю 1*, та зазначити книжкову орієнтацію. Натиснути кнопку <*Далее*>;
- обрати стиль *Строгий* та натиснути кнопку вікна <*Далее*>;
- вказати ім'я звіту - *Препарати*;
- натиснути кнопку вікна <*Готово*>;
- переглянути готовий звіт, переконатися у його правильності та закрити вікно створено звіту.

4. Модифікувати створений звіт, таким чином, щоб у ньому відображалася лише назва групи препаратів (Жарознижуючі, Анальгетики тощо) та назви препаратів (Бісептол, Гендевіт тощо) і не відображалися назви полів (*Група_преп* та *Назва препарату*). Для назви групи препаратів встановити жирний шрифт 12 пт. Для назви препаратів встановити шрифт звичайний 10 пт. Для модифікації звіту потрібно зайти у режим конструктора та скористатися вже відомими методами модифікації елементів.

5. За допомогою конструктора на основі таблиць **Агенти**, **Замовники** та **Виробники** створити звіти з однойменними назвами. Оформлення звітів виконати за власним бажанням.

6. Створити автозвіт у стовпчик на основі запиту *Товарна накладна*. Звіт назвати *Товарний чек*. За допомогою конструктора змінити зовнішній вигляд звіту, максимально наблизивши його до вигляду товарного чека.

7. Завершити роботу з додатком.

Лабораторна робота № 7 Створення головної кнопкової форми[1]. Зв'язок Access з Word та Excel

Мета: навчитися створювати головну кнопкову форму та здійснювати зв'язок Access з Word та Excel.

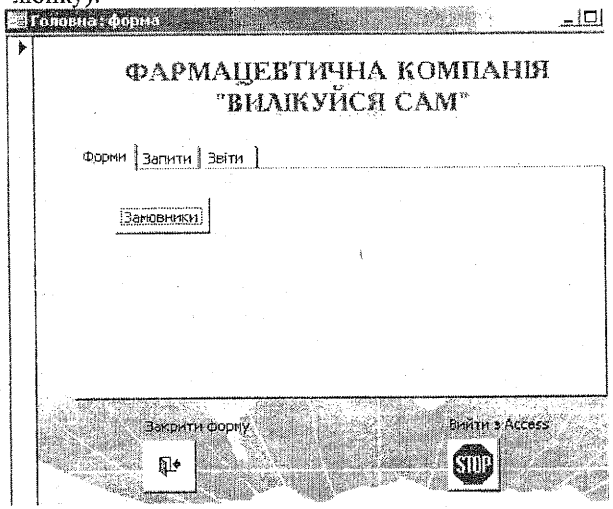
1. Завантажити програму Microsoft Access та відкрити робочий файл *Фармацевтична фірма.*

2. Для створення головної кнопкової форми скористаємося конструктором форм:

- у вікні бази даних обрати об'єкт *Формы*;
- подвійним клацанням мишки активізувати пункт *Создание формы в режиме конструктора*;
- за допомогою лівої кнопки миші методом перетягуванні збільшити область даних: встановити розмір 12х6 см;
- виконати команду меню **Формат/Автоформат...** та обрати один із запропонованих варіантів оформлення;
- розмістити зверху по центру форми напис *Фармацевтична компанія.* Оформлення напису виконати на власний розсуд, за бажанням можна розмістити також малюнок з логотипом.

3. Проведемо групування елементів бази даних за запитами, формами та звітами, оформивши кожну категорію окремою вкладкою:

- обрати на панелі елементів елемент *Вкладка* та розмістити її у центрі форми (приблизний загальний вигляд зображено на малюнку):



[1] Головна кнопкова форма служить для групування складових частин бази даних за функціями, які вони виконують, та результатами виконання.

- за замовчуванням буде встановлено дві вкладки. Активізувати контекстне меню в області створених вкладок та обрати пункт *Добавить вкладку*;
- активізувати контекстне меню на назві першої вкладки, обрати пункт *Свойства* та у полі *Имя* (вкладка *Все)* ввести нову назву – **Форми**;
- аналогічно перейменувати дві інші вкладки, надати їм імена **Запити** та **Звіти**.

4. На вкладці **Форми** розмістити кнопку для виведення форми *Агенти*:

- на панелі елементів обрати елемент *Кнопка*;
- на вкладці **Форми** створити кнопку (так само, як напис);
- у вікні *Создание кнопок*, що з'явиться, обрати категорію *Работа с формой*, дію *Открыть форму* та натиснути кнопку вікна *<Далее>*;
- обрати форму – *Агенти*; натиснути кнопку вікна *<Далее>*;
- обрати пункт *Открыть форму и показать все записи*; натиснути кнопку вікна *<Далее>*;
- обрати для напису на кнопці текст *Агенти*; натиснути кнопку вікна *<Далее>*;
- надати ім'я кнопці – *Агенти*; натиснути кнопку вікна *<Готово>*.

5. Аналогічно створити кнопки для готових форм *Виробники, Замовники, Препарати, Угоди.*

6. На вкладці **Запити** розмістити кнопки для п'яти довільних запитів, обраних за власним бажанням (категорія – *Разное*, дія – *Выполнить запрос*).

7. На вкладці **Звіти** розмістити кнопки для трьох звітів, обраних за власним бажанням (категорія – *Работа с отчетом*, дія – *Просмотр отчета*).

8. У вільному місці форми (поза межами створених вкладок) створити кнопку виходу з головної кнопкової форми (категорія – *Работа с формой*, дія – *Закрыть форму*). Біля кнопки встановити пояснювальний напис – **Закрити форму**.

9. У вільному місці форми (поза межами створених вкладок) створити кнопку виходу з Access (категорія – *Приложение*, дія – *Выйти из приложения*). Біля кнопки встановити пояснювальний напис – **Вийти з Access**.

10. На прикладі запиту *Кількість угод замовників* розглянути можливості передачі даних в MS Excel:

- у вікні бази даних обрати об'єкт *Запросы*;
- перейти до запиту *Кількість угод замовників*;
- виконати команду меню **Сервис/Связи с Office/Анализ в Microsoft Office Exel**;
- переглянути створений файл електронних таблиць, переконатися у правильності експортованих даних та зберегти його з іменем file_1.xls на диску.

Навчальне видання

ЗАРИЦЬКА Оксана Леонідівна

БАЗИ ДАНИХ ТА ІНФОРМАЦІЙНІ СИСТЕМИ

Методичний посібник

Надруковано з оригінал-макету автора
Підписано до друку 31.08.2010. Формат 60x90/16. Ум.друк арк. 8,25. Обл.вид. арк. 7,86
Гарнітура Time New Roman. Зам. 163.

Видавництво Житомирського державного університету імені Івана Франка
ЖТ №10 від 07.12.04 р.
м. Житомир, вул. Велика Бердичівська, 40
електронна пошта (E-mail): zu@zu.edu.ua