Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

Методологія інформаційних систем та баз даних: теоретичний і практичний підходи

Навчальний посібник

Укладачі:
Ю.О. Ушенко, М.Л. Ковальчук,
М.С. Гавриляк, А.Л. Негрич

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2021

УДК 004.65 (075.8)
М 54

Друкується за ухвалою Вченої ради
Чернівецького національного університету імені Юрія Федьковича
(протокол № 7 від 30 червня 2021 р.)

Рецензенти:
Вайбуз О.Г. доктор техн.. наук, професор, завідувач кафедри математичного забезпечення ЕОМ Дніпровського національного університету імені Олеся Гончара
Осауленко І.А. доктор. техн. наук, доцент, завідувач кафедри інтелектуальних систем прийняття рішень Черкаського національного університету імені Богдана Хмельницького

Укладачі: Ушенко Ю.О., Ковальчук М.Л.,
Гавриляк М.С., Негрич А.Л.

Методологія інформаційних систем та баз даних: теоретичний і практичний підходи : навч. посібник / уклад. Ю.О. Ушенко, М.Л. Ковальчук, М.С. Гавриляк, А.Л. Негрич. – Чернівці : Чернівецький нац. ун-т ім. Ю. Федьковича, 2021. 240 с.
ISBN 978-966-423-641-3

Видання розкриває інформаційні системи та мережі, їх математичне, інформаційне та програмне забезпечення, способи та методи проектування, експлуатації інформаційних систем у різних галузях, у тому числі в сучасних телекомунікаційних і поліграфічних інформаційних мережах. Посібник містить лабораторний практикум із проектування, створення й керування базами даних MySQL, використовуючи PHP.

Для студентів, які навчаються за спеціальностями 122 Комп'ютерні науки, 152 Метрологія та інформаційно-вимірювальна техніка 172 Телекомунікації та радіотехніка; 186 Видавництво та поліграфія.

УДК 004.65 (075.8)

ISBN 978-966-423-641-3

© Чернівецький національний університет
імені Юрія Федьковича, 2021

Зміст

Вступ

Програмне забезпечення за половину століття, яке воно існувало, зазнало суттєвих змін: від програм, що були здатні виконувати тільки найпростіші арифметичні операції до складних систем керування підприємствами. У розвитку програмного забезпечення завжди можна було виділити два основних напрямки:

- виконання розрахунків;
- накопичення й обробка інформації.

З самого свого початку комп'ютери призначалися для виконання складних математичних розрахунків (у першу чергу для розрахунків, пов'язаних із створенням ядерної зброї й ракетної техніки), а нині домінує другий напрям.

На сьогодні керування підприємством без комп'ютера архаїчне. Комп'ютери тісно й міцно увійшли в такі сфери керування, як бухгалтерський облік, керування складом, асортиментом, закупівлями.

Хоча інформаційні системи є звичайним програмним продуктом, вони мають ряд суттєвих відмінностей від стандартних прикладних програм і систем.

Залежно від предметної області інформаційні системи можуть дуже сильно відрізнятися за своїми функціями, архітектурою, реалізацією. *Предметна область* - частина реального світу, що підлягає вивченню з метою організації керування й, в остаточному підсумку, автоматизації. Предметна область представляється множиною *фрагментів*, наприклад, підприємство - цехами, дирекцією, бухгалтерією й т. і. Кожен фрагмент предметної області можна охарактеризувати як множину *об'єктів* і *процесів*, що використовують об'єкти, а також як множину *користувачів*, які характеризуються різними поглядами на предметну область.

Проте існує ряд спільних властивостей:

- інформаційні системи, що призначені для збору, зберігання та обробки інформації. Тому в основі кожної з них знаходиться середовище зберігання та доступу до даних.

- інформаційні системи, орієнтованя на кінцевого користувача, який не володіє високою кваліфікацією в даній галузі.

Тому під час розробки інформаційної системи необхідно розв'язувати дві основні задачі:

- задачу розробки бази даних, призначеної для зберження інформації;
- задачу розробки графічного дизайну користувача клієнтських додатків.

Модуль 1. «Термінологія інформаційних систем»
Розділ 1. Інформаційні системи

У цьому розділі розглядаються загальні поняття і типи інформаційних систем, визначаються їх базові властивості, а також формулюються задачі, які розв'язуються при розробці цих систем, і проблеми, що при цьому виникають.

1.1. Основні поняття

Під *інформаційною системою* (ІС) мається на увазі прикладна програмна підсистема, призначена для збору, зберігання, пошуку і обробки текстової й/або фактографічної інформації (інформація, що існує у вигляді зареєстрованих фактів). Більшість ІС вхаємодіє з користувачем в діалоговому режиміз.

У типовому випадку, програмні компоненти, що входять до складу ІС, маютьь:

- діалоговий увід-вивід;
- логіку діалогу;
- прикладну логіку обробки даних;
- логіку керування даними;
- операції маніпулювання файлами або базами даних;

Під *корпоративною інформаційною системою* (КІС) ми розуміємо сукупність спеціалізованого програмного забезпечення й обчислювальної техніки, на якій установлено й налаштовано програмне забезпечення.

Фактори, що впливають на розвиток КІС

Останнім часом усе більше співробітників починають чітко розуміти важливість побудови на підприємстві КІС як важливого інструменту для успішного управління бізнесом у сучасних умовах.

Існують три найбільш важливих фактори, які визначають розвиток КІС:

- розвиток методик керування організацією;
- розвиток компетентностей і продуктивності комп'ютерних систем;
- еволюція підходів до технічної й програмної реалізації компонентів ІС.

Розвиток методик керування підприємством

Теорія керування бізнес-структурою являє собою дуже широкий предмет для вивчення та удосконалення. Це зумовлено постійною зміною реалій на світовому ринку. Постійно зростаючий рівень конкуренції стимулює керівників компаній шукати нові методи розширення своєї присутності на ринку й підтримки прибутковості своєї діяльності. Подібними методами виступають диверсифікація, децентралізація, управління якістю та інше. Сучасна ІС повинна відповідати всім інноваціям у теорії й практиці менеджменту. Без сумніву, це вагомий фактор, оскільки побудова сучасної, у технічному планії, системи, яка не відповідає вимогам за функціональністю, немає сенсу.

Прогрес у галузі нарощування потужності й продуктивності комп'ютерних систем, еволюція мережевих технологій і систем передачі даних, широкі можливості взаємодії комп'ютерної техніки з різним обладнанням дозволяють невпинно підвищувати продуктивність ІС і їх функціонал.

Розвиток підходів до технічної та програмної реалізації елементів ІС

Паралельно з розвитком апаратної частини ІС протягом останніх років відбувається постійний пошук нових, більш зручних і універсальних, методів програмно-технологічної реалізації ІС. Можна виділити три найбільш суттєві інновації, які здійснили колосальний вплив на розвиток ІС в останні роки:

- *новий підхід до програмування*: від початку 90-х років об'єктно-зорієнтоване програмування фактично витіснило модульне; дотепер неперервно вдосконалюються методи побудови об'єктних моделей. Завдяки уведенню об'єктно-зорієнтованих технологій програмування суттєво скорочуються строки розробки складних ІС, спрощуються їх підтримка й розвиток;
- завдяки *розвитку мережевих технологій* локальні ІС повсюдно витісняються клієнт-серверними та багаторівневими реалізаціями;
- *розвиток мережі Інтернет* приніс великі можливості роботи з віддаленими підрозділами, відкрив широкі перспективи електронної комерції, обслуговування покупців через Інтернет і

багато іншого. Більше того, деякі переваги дає використання Інтернет-технологій в інтрамережах підприємства (так звані інтранет-технології).

Зауваження: Треба мати на увазі, що використання певних технологій при побудові ІС не являється самоціллю розробника. Вибір технологій повинен вироблятися залежно від реальних потреб.

Основні складові КІС

У складі КІС можна виділити дві відносно незалежні складові:

- *Комп'ютерну інфраструктуру* організації, яка являє собою сукупність мережевої, телекомунікаційної, програмної, інформаційної й організаційної інфраструктур. Дана складова, зазвичай, називається *корпоративною мережею.*
- *Взаємопов'язані функціональні підсистеми,* що забезпечують розв'язування задач організації та досягнення її цілей.

Перша складова відбиває системно-технічну, структурну сторону будь-якої ІС. По суті, це - основа для інтеграції функціональних підсистем, яка повністю визначає властивість інформаційної системи. Вимоги до комп'ютерної інфраструктури єдині й стандартизовані, а методи її будівництва добре відомі та неодноразово перевірені на практиці.

Друга складова КІС цілком відноситься до прикладної сфери й сильно залежить від специфіки задач та цілей підприємства. Дана складова цілком базується на комп'ютерній інфраструктурі підприємства та визначає прикладну функціональність ІС. Вимоги до функціональних підсистем складні й суперечливі, оскільки висуваються спеціалістами з різних прикладних галузей. Але в кінцевому рахунку саме ця складова найбільш важлива для функціонування організації, оскільки для неї і будується комп'ютерна інфраструктура.

Співвідношення між складовими ІС

Взаємозв'язки між двома вказаними складовими ІС досить складні. Наприклад, організація мережі й протоколи, які використовуються для обміну даними між комп'ютерами, абсолютно

не залежать від того, які методи й програми планується використовувати на підприємстві для організації бухгалтерського обліку.

З іншого боку, вказані складові в деякому розумінні залежать одна від одної. Функціональні підсистеми в принципі не можуть існувати без комп'ютерної інфраструктури. Водночас комп'ютерна інфраструктура сама по собі досить обмежена, оскільки не володіє необхідною функціональністю. Неможливо експлуатувати розподілену інформаційну систему за відсутності мережевої інфраструктури. Хоча маючи розвинену інфраструктуру можна надати співробітникам організації ряд корисних загальносистемних служб (наприклад, електронну пошту та доступ до Інтернету), які спрощують роботу і роблять її ефективнішою (в даному випадку за рахунок використання більш розвинених засобів зв'язку).

Отже, розробку ІС доцільно починати з побудови комп'ютерної інфраструктури (корпоративної мережі) як найбільш важливої складової, що спирається на апробовані промислові технології та гарантовано реалізується в розумні строки внаслідок високого ступеня визначеності як у постановці задачі, так і в запропонованих розв'язках.

Зауваження: Безглуздо будувати корпоративну мережу, як деяку самодостатню систему, не беручи до уваги прикладну функціональність. Якщо у процесі створення системно-технічної інфраструктури не проводиться аналіз і автоматизацію управлінських задач, то засоби, інвестовані в розробку корпоративної мережі, не дадуть реальної віддачі.

Корпоративна мережа створюється на багато років уперед, капітальні затрати на її розробку й упровадження на стільки великі, що практично виключають можливість повної або часткової переробки існуючої мережі.

Функціональні підсистеми, на відміну від корпоративної мережі, змінні за своєю природою, оскільки в предметній сфери діяльності організації постійно відбуваються більш або менш значні зміни. Функціональність ІС сильно залежать від організаційно-управлінської структури організації, її функціональності, існуючих технологій документообігу й багатьох інших факторів.

Розробку та впровадження функціональних підсистем можна виконувати поступово. Наприклад, спочатку на більш важливих і відповідальних ділянках виконувати розробки, які забезпечують прикладну функціональність системи (упроваджувати системи фінансового обліку, керування кадрами і т.і.), а потім розповсюджувати прикладні програмні системи й на інші менш важливі сфері керування підприємством.

1.2. Класифікація інформаційних систем

ІС класифікуються за різними ознаками. Розглянемо способи класифікації, які найчастіше використовуються.

Класифікація за масштабом

За масштабом ІС поділяються на такі групи – Рис.1.1.

- одиночні;
- групові;
- корпоративні.

Рис.1.1. Розподіл ІС за масштабом

Одиночні ІС реалізуються, як правило, на автономному персональному комп'ютері (де мережа відсутня). Подібна система включає в себе декілька простих додатків, пов'язаних загальним інформаційним контентом, і призначена для роботи одного користувача або групи користувачів, розділені в часі на одному робочому місці. Такі додатки створюються за допомогою певних *настільних* або *локальних* систем управління базами даних (СКБД). Серед локальних СКБД дуже відомі Clarion, Clipper, FoxPro, Paradox, dBase і Microsoft Access.

Групові ІС призначені для колективного використання інформації учасниками робочої групи й найчастіше будуються на основі обчислювальної мережі. При створенні таких додатків застосовуються сервери баз даних (які називаються SQL-серве-

рами) для потреб робочої групи. Існує надзвичайно велика кількість різних SQL-серверів, як комерційних, так тих і які вільно поширюються. Із них найвідомішими є сервери баз даних Oracle, DB2, Microsoft SQL Server, InterBase, Sybase.

Корпоративні ІС є еволюцією систем для робочих груп, вони призначені для великих компаній і можуть містити територіально рознесені вузли або мережі. Здебільшого вони мають ієрархічну структуру з ієрархією рівнів. Для цих систем характерною є топологія клієнт-сервер зі спеціалізацією серверів або ж багаторівнева топологія. При побудові цих систем використовують ті самі сервери баз даних, що й для розробки групових ІС. Але у великих ІС широкого поширення набули сервери Oracle, DB2, Microsoft SQL Server.

Для групових і корпоративних систем значною мірою підвищуються вимоги до відмовостійкості збереження даних. Ці властивості забезпечують підтримку цілісності даних, посилань і передачі на серверах баз даних.

Класифікація за сферою застосування

За цариною застосування ІС часто поділяються на чотири групи (Рис. 1.2.):

- системи обробки транзакцій;
- системи прийняття рішень;
- інформаційно-довідкові системи;
- офісні інформаційні системи.

Системи обробки транзакцій, у свою чергу, по оперативності обробки даних, поділяються на пакетні ІС і оперативні ІС. В ІС організаційного керування переважає режим оперативної обробки транзакції – OLTP (OnLine Transaction Processing), для відображення актуального стану предметної області в будь-який момент часу, а пакетна обробка займає доволі обмежену частину. Для систем OLTP характерний регулярний (можливо, інтенсивний) потік досить простих трансакцій, які відіграють роль замовлень платежів, запитів і т.і. Важливими вимогами до них є:

- висока продуктивність обробки транзакцій;
- гарантована доставка інформації при віддаленому доступі до БД по телекомунікаціях.

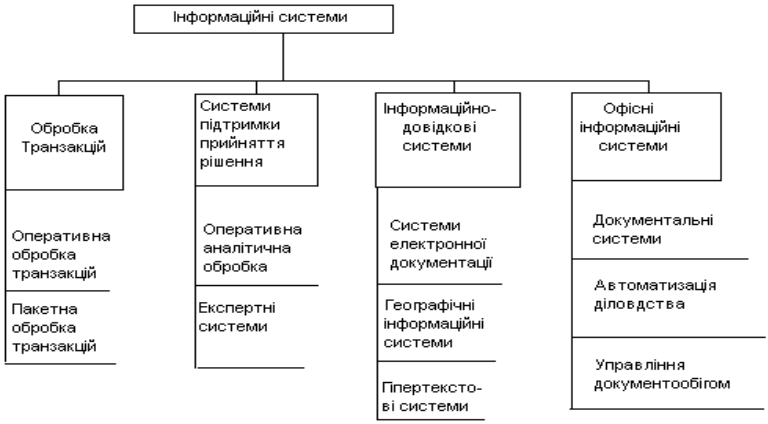


Рис.1.2. Розподіл ІС за сферою застосування

Системи підтримки прийняття рішень – DSS (Decision Support System) – являють собою інший тип ІС, у яких за допомогою досить складних запитів проводиться відбір і аналіз даних у різних розрізах: часових, географічних і за іншими показниками.

Широкий клас *інформаційно-довідкових систем* заснований на гіпертекстових документах і мультимедіа. Найбільший розвиток такі інформаційні системи отримали в мережі Інтернет.

Клас *офісних ІС,* спрямованих на переведення паперових документів у електронний вигляд, автоматизацію діловодства й керування документообігом.

Зауваження: Наведена класифікація за сферою застосування досить умовна. Великі ІС дуже часто володіють ознаками всіх вищеперерахованих класів. Крім того, корпоративні ІС у масштабах підприємства зазвичай складаються з ряду підсистем, які належать до різних сфер застосування.

Класифікація за способом організації

За способом організації групові й корпоративні ІС поділяються на такі класи (Рис. 1.3.). Системи на основі:

- архітектури файл-сервер;
- архітектури клієнт-сервер;

- багаторівневої архітектури;
- Інтернет/інтранет-технологій.

Рис.1.3. Розподіл ІС за способом організації

У будь-якій ІС можна виділити необхідні функціональні компоненти (Табл. 1.1.), які допомагають зрозуміти обмеження різних архітектур ІС. Розглянемо більш детально особливості варіантів побудови інформаційних додатків.

Таблиця 1.1.
Функціональні компоненти ІС

Позначення	Найменування	Характеристика
PS	Presentation Services **(**засоби представлення)	Забезпечуються пристроями, які приймають увід від користувача й відображують те, що повідомляє йому компонент логіки представлення PL, з використанням відповідної програмної підтримки.
PL	Presentation Logic (логіка представлення)	Керує взаємодією між користувачем і ЕОМ (електронною обчислювальною машиною). Обробляє дії користувача при виборі команди в меню, натисканні клавіші або виборі елемента зі списку.
BL	Business or Application Logic (прикладна логіка)	Набір правил для прийняття рішень, розрахунків та операцій, які повинен виконати додаток.
DL	Data Logic (логіка керування даними)	Операції з базою даних (SQL-оператори) які потрібно виконати для реалізації

		прикладної логіки керування даними.
DS	Data Services (операції з базою даних)	Дії СКБД, які викликаються для виконання логіки керування даними такі, як маніпулювання даними, визначення даних, фіксація або відмова транзакції й т.і. СКБД зазвичай компілює SQL-додатки.
FS	File Services (файлові операції)	Дискові операції зчитування й запису даних для СКБД та інших компонентів. Зазвичай є функціями операційної системи (ОС).

Архітектура файл-сервер не має мережевого розділення компонентів діалогу **PS** і **PL** і використовує комп'ютер для функції відображення, що полегшує побудову графічного інтерфейсу. Файл-сервер тільки вилучає дані з файлів, так що додаткові користувачі та додатки додають лише незначне навантаження на центральний процесор.

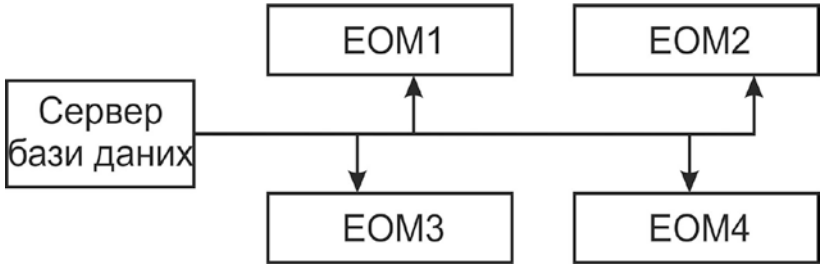


Рис.1.4. Класичний варіант файл-серверної ІС

В архітектурі файл-сервер (Рис. 1.4) БД зберігається на сервері, а СКБД установлюється на кожній ЕОМ. Яскравим прикладом такої архітектури є СКБД Microsoft Access. Продуктивність залежить від комп'ютера користувача. Об'єктами розробки у файл-серверному додатку є компоненти додатку, які визначають логіку діалогу **PL ,** а також логіку обробки **BL** і керування даними **DL.** Розроблений додаток реалізується або у вигляді завершеного

завантаженого модуля, або у вигляді спеціального коду для інтерпретації.

Проте така архітектура має суттєвий недолік – дані зберігаються в одному місці (сервер БД), а обробляються в іншому (ПК клієнта), тобто при виконанні деяких запитів до бази даних клієнту можуть передаватися великі обсяги даних, завантажуючи мережу й приводячи до непередбачуваності часу реакції. Значний мережевий трафік особливо сильно виявляється при організації віддаленого доступу до баз даних на файл-сервері через низькошвидкісні канали зв'язку. Одним з варіантів усунення даного недоліку є віддалене керування файл-серверним додатком у мережі. При цьому в локальній мережі розміщується сервер додатків, суміщений з телекомунікаційним сервером (зазвичай його називають сервером доступу), у середовищі якого виконуються звичайні файл-серверні додатки. Особливість полягає в тому, що діалоговий увід-вивід надходить від віддалених клієнтів через телекомунікації. Додатки не повинні бути занадто складними, оскільки є велика ймовірність перевантаження сервера або ж потрібна дуже потужна платформа для сервера додатків.

Зауваження: Одним із традиційних засобів, на основі яких створюються файл-серверні системи, є локальні СКБД. Однак такі системи, як правило, не відповідають вимогам забезпечення цілісності даних. Тому при їхньому використанні завдання забезпечення цілісності даних покладається на програми клієнтів, що призводить до ускладнення клієнтських додатків. Проте ці інструменти приваблюють своєю простотою, зручністю у використанні та доступністю. Тому файл-серверні інформаційні системи (ІС) до цих пір представляють інтерес для малих робочих груп і, більше того, нерідко використовуються в якості ІС у масштабах підприємства.

Архітектура клієнт-сервер призначена для розв'язання проблем файл-серверних додатків шляхом розділення компонентів додатку і розміщення їх там, де вони будуть функціонувати найбільш ефективно. Особливістю архітектури клієнт-сервер є використання виділених серверів баз даних, які розуміють запити мовою структурованих запитів SQL (Structured Query Language) і які виконують пошук, сортування й агрегу-

вання інформації. БД зберігається на сервері, а СКБД ділиться на дві частини: клієнтську та серверну.

Відмінна риса серверів БД – наявність довідника даних, у якому записана структура БД, обмеження цілісності даних, формати і навіть серверні процедури обробки даних за викликом або за подіями у програмі. Об'єктами розробки в таких додатках окрім діалогу та логіки обробки, є, перш за все, реляційна модель даних і пов'язаний з нею набір SQL-операторів для типових запитів до бази даних.

Більшість конфігурацій клієнт-сервер використовує дворівневу модель, у якій клієнт звертається до послуг сервера. Припускається, що діалогові компоненти **PS** та **PL** розміщуються на клієнті, що дозволяє забезпечити графічний інтерфейс. Компоненти керування даними **DS** і **FS** розміщуються на сервері, а діалог (**PS, PL),** логіка **DL** і **DL** – на клієнті. Дворівневе визначення архітектури клієнт-сервер використовує саме цей варіант: додаток працює у клієнта, СКБД – на сервері (Рис. 1.5)

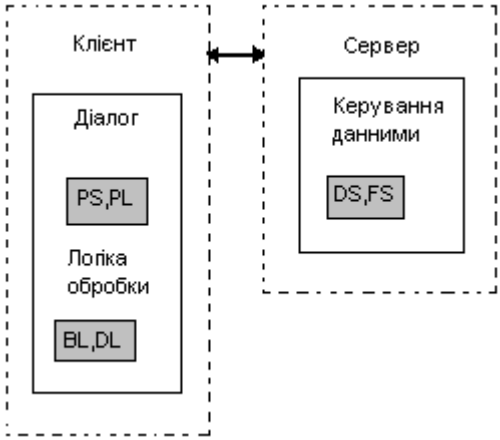


Рис.1.5. Класичний варіант клієнт-серверної ІС

Оскільки ця схема висуває найменші вимоги до сервера, вона володіє найкращою масштабованістю. Проте складні додатки, які викликають значну взаємодію з БД, можуть суттєво завантажити як клієнта, так і мережу. Результати SQL-запиту

повинні повернутися до клієнта для обробки, тому що там знаходиться логіка прийняття рішення. Така схема призводить до додаткового ускладнення адміністрування додатка, який розкиданий по різних клієнтських вузлах.

Для скорочення навантаження на мережу та спрощення адміністрування додатків компонент **BL** можна розмістити на сервері. При цьому вся логіка прийняття рішень оформлюється у вигляді *збережених процедур* (дане питання детально розглядається у параграфі 5.4.5) та виконується на сервері БД.

Збережена процедура – це процедура з операторами SQL для доступу до БД, яку викликають по імені з передачею потрібних параметрів та яка виконується на сервері БД.

Збережені процедури можуть компілюватися, що підвищує швидкість їх виконання та скорочення навантаження на сервер. Вони поліпшують цілісність додатків і БД, гарантують актуальність колективно використовуваних операцій та розрахунків. Поліпшується супровід таких процедур, а також безпека (немає прямого доступу до даних).

Зауваження: Перенавантаження збережених процедур прикладною логікою може перенавантажити сервер, що призведе до втрати продуктивності. Ця проблема особливо актуальна при розробці особливо великих інформаційних систем, у яких до сервера може одночасно звертатись велика кількість клієнтів. То у більшості випадків необхідно приймати компромісні рішення: частину логіки додатків розміщувати на боці сервера, а частину – на боці клієнта. Такі клієнт-серверні системи називаються система з розділеною логікою. Дана схема при вдалому розділенні логіки дозволить отримати більш збалансоване навантаження клієнта і сервера, але при цьому ускладнюється супровід додатків.

Створення архітектури клієнт-сервер можливе і на основі багатотермінальної системи. У цьому випадку в багатозадачному середовищі сервера додатків виконуються програми користувачів, а клієнтські вузли представлені терміналами. Схожа схема інформаційної системи характерна для UNIX.

Дворівнева система архітектури клієнт-сервер може призвести до деяких проблем у складних інформаційних додатках з великою кількістю користувачів і заплутаною логікою. Роз-

в'язком цих проблем може стати використовування багаторівневої системи.

Багаторівнева архітектура являє собою еволюцію архітектури *клієнт-сервер* і у своїй класичний формі містить три рівні (див. Рис. 1.6.):

- нижній рівень являє собою додатки клієнтів, виділені для виконання функцій і логіки відображення **PS** і **PL** та має програмний інтерфейс для виклику додатка на середньому рівні;
- середній рівень – являє собою сервер додатків, який відповідає за виконання прикладної логіки **BL** та з якого логіка даних **DL** викликає операції з базою даних **DS**;
- вищий рівень – віддалений спеціалізований сервер бази даних, виділений для послуг обробки даних **DS** і файлових операцій **FS** (ризики використання збережених процедур відсутні).

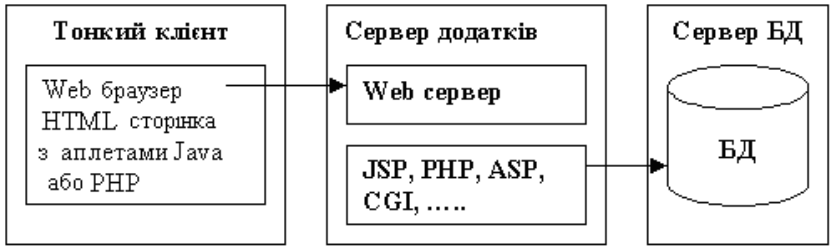


Рис.1.6. Класичний варіант трирівневої ІС

Тонкий клієнт забезпечує взаємодію з користувачем через браузер, вся прикладна логіка обробки виноситься на сервер додатків, що забезпечує й формування запиту до БД. При цьому сервер бази даних і сервер додатків можуть функціонувати в різних операційних системах.

Трирівнева архітектура допускає подальше зростання навантаження на певні вузли та мережу, а також підтримує спеціалізації інструментів для розробки додатків і ліквідує недоліки дворівневої моделі клієнт-сервер.

Централізація логіки додатка спрощує адміністрування та супровід. Чітко розділяються платформи та інструменти для реалізації інтерфейсу та прикладної логіки, що допомогає з найбільшою ефективністю реалізовувати їх силами спеціалістів вузького профілю. Та на останок, зміни прикладної логіки не торкаються інтерфейсу. Але оскільки межа між компонентами **PL, BL** і **DL** розмита, прикладна логіка може з'явитися на всіх трьох рівнях. Сервер додатків за допомогою монітора транзакцій забезпечує інтерфейс із клієнтами, а також з іншими серверами, може керувати транзакціями та гарантувати цілісність розподіленої бази даних. Засоби віддаленого запуску процедур найбільш відповідають концепції розподілених обчислень: вони забезпечують з довільного вузла мережі виклик прикладної процедури, яка розташована на іншому вузлі, передачу параметрів, віддалену обробку та повернення результатів.

Із зростанням систем клієнт-сервер необхідність трьох рівнів стає все більш очевидним. Продукти для трирівневої архітектури, так звані монітори транзакцій, порівняно нові. Ці інструменти в основному зорієнтовані на середовище UNIX, проте прикладні сервери можна будувати на базі Microsoft Windows NT із використанням виклику віддалених процедур для організації зв'язку клієнтів із сервером додатків. На практиці в локальній мережі можуть використовуватися змішані архітектури (дворівневі та трирівневі) з одним і тим сервером бази даних. З урахуванням глобальних зв'язків архітектура може мати більше трьох ланок. На сьогодні вже є нові інструментальні засоби для гнучкої сегментації додатків клієнт-сервер по різних вузлах мережі.

Отже, багаторівнева архітектура розподілених додатків дозволяє підвищити ефективність роботи корпоративної інформаційної системи й оптимізувати розподіл її програмно-апаратних ресурсів. Але поки що на ринку, як завжди, домінує архітектура клієнт-сервер.

У розвитку *технології Інтернет/інтранет* основний акцент поки що покладається на розробку інструментальних програмних засобів. Водночас спостерігається відсутність розвинених засобів розробки додатків, які працюють із базами даних. Компромісним рішенням для створення зручних і

простих у користуванні та супроводі інформаційних систем, ефективно працюючих з базами даних, стало об'єднання Інтернет/інтранет-технології з багаторівневою архітектурою. При цьому структура інформаційного додатка набуває такого вигляду: ***браузер–сервер додатків–сервер баз даних–сервер динамічних сторінок – web-сервер***.

Завдяки інтеграції Інтернет/інтранет-технологій та архітектури клієнт-сервер процес упровадження та супровід корпоративної інформаційної системи суттєво спрощується при збереженні досить високої ефективності та простоти спільного використання інформації.

Розділ 2. Життєвий цикл інформаційних систем

Розробка корпоративної інформаційної системи, як правило, виконується для цілком визначеного підприємства. Особливості предметної діяльності підприємства будуть впливати на структуру інформаційної системи. Та водночас структури різних підприємств у цілому схожі між собою. Кожна організація, незалежно від роду її діяльності, складається з ряду підрозділів, безпосередньо здійснюючих той чи інший вид діяльності компанії. І ця ситуація справедлива практично для всіх організацій, якими видами діяльності вони б не займались.

Тому довільну організацію можна розглядати як сукупність, елементів, що взаємодіють між собою (підрозділів), кожен з яких містить свою, достатньо складну структуру. Взаємозв'язки між підрозділами також досить компліковані. У загальному випадку існують три види зв'язків між підрозділами установи:

- *функціональні зв'язки* – кожен підрозділ виконує специфічні види робіт у межах єдиного бізнес-процесу;
- *інформаційні зв'язки* – підрозділи взаємодіють інформаційному рівні (документами, факсами, письмовими та голосовими розпорядженнями й т.і);
- *зовнішні зв'язки* – певні підрозділи взаємодіють із зовнішніми системами, і їх взаємозв'язок також може бути інформаційним або функціональним.

Спільність структури різних установ дозволяє сформулювати певні єдині алгоритми побудови корпоративних інформаційних систем.

В більшості випадків процес розробки інформаційної системи може розглядатися з двох боків:

- за метою дій розробників (груп розробників). У цьому випадку розглядається статистичний аспект процесу розробки, який формулюється в термінах основних потоків робіт: виконавці, дії, послідовність дій і т. ін.;
- за таймінгом або за етапами життєвого циклу розроблюваної системи. У такому випадкуі розглядається динамічна організація процесу розробки, описана на мові циклів, стадій, ітерацій та етапів.

2.1. Загальні відомості про керування проектами

Інформаційна система організації розробляється як ревний *проект*. Багато особливостей управління проектами та етапи розробки проектів (етапи життєвого циклу) спільні, і незалежні не тільки від предметної області, але й від типу проекту (неважливо, чи це інженерний проект або економічний). Тому є необхідність спочатку розглянути перелік загальних питань керування проектами.

Поняття проекту

Проект – це обмежена в часі цілеспрямована иодифікаціяа окремої системи зі історично визначеною метою, досягнення якої є метою даного проекту, а також з визначеними вимогами до термінів, результатів, ризику, лімітом витрачання коштів та ресурсів і до структури організації.

Зауваження: Зазвичай для складного поняття (яким є поняття проекту) важко дати однозначне формулювання, яке повністю охоплювало всі грані введеного поняття. Тому дане визначення не є вичерпним.

Виділяють такі основні характерні ознаки *проекту*, з точки зору об'єкту керування:

- мінливість – цілеспрямоване переведення системи з існуючого в бажаний стан, що можна описати в термінах цілей проекту;
- обмеження кінцевої мети;
- ліміт тривалості;
- ліміт бюджету;
- ліміт вимог до ресурсів;
- ліміт правового й організаційного забезпечення – створення специфічної організаційної структури на період реалізації проекту.

Розглядаючи планування проектів і керування ними, необхідно чітко усвідомлювати, що мова йде про керування деякими динамічним об'єктом. Тому система керування проектом повинна бути досить гнучкою, щоб дати можливість модифікації без глобальних змін у робочій програмі.

З точки зору системного підходу проект може буди представлений «чорною скринею», входом до якої є технічні вимоги та можливості фінансування, а підсумком роботи – досягнення потрібного результату (Рис. 2.1.). Виконання робіт забезпечується можливістю доступу до необхідних ресурсів:

- за матеріалами;
- за обладнанням;
- наявних людських ресурсів;

Ефективність роботи досягається за рахунок керування процесом реалізації проекту, який забезпечує розподіл ресурсів, координацію послідовності робіт і компенсацію внутрішніх і зовнішніх збурень.

Рис. 2.1. Представлення проекту у вигляді «чорної скриньки»

З погляду теорії систем керування проект як об'єкт керування має спостерігатися та керуватися, тобто виділяються деякі характеристики, за якими можна постійно контролювати хід виконання проекту (властивість *спостереження*). Крім того, необхідні механізми своєчасного впливу на хід реалізації проекту (властивість *керованості).*

Для обґрунтованості доцільності проекту, аналізу ходу його реалізації, а також для заключної оцінки ступеня досяжності поставленої мети проекту та порівняння фактичних результатів із запланованими існує ціла низка характеристик проекту. До найважливіших з них належать техніко-економічні показники:

- об'сяг робіт;
- строки виконання;
- собівартість;

- економічна ефективність, яка забезпечується реалізацією проекту;
- соціальна та суспільна значимість проекту.

Класифікація проектів

Проекти можуть досить сильно відзначатися за сферою застосування, складом, предметною областю, масштабом, тривалістю, складом учасників і т. ін. Проекти можуть бути класифіковані за різними ознаками. Відзначимо основні з них.

Клас проекту визначається щодо його складу і структури. Зазвичай вичленовують:

- монопроекти;
- мультипроекти (комплексний проекти, що складається з кількох монопроектів і потребують застосування багатопроектного керування).

Тип проекту визначається за сферами діяльності, що домінують при здійсненні проекту. Розрізняють п'ять основних типів:

- технічний проект;
- організаційний проект;
- економічний проект;
- соціальний проект;
- змішаний проект.

Масштаб проекту визначений розмірами бюджету та кількістю виконавців:

- дрібний проект;
- малий проект;
- середній проект;
- великий проект.

Можемо також розглядати масштаби проектів у дещо конкретнішій формі – як галузеві, корпоративні, відомчі.

Основні фази проектування інформаційної системи

Кожен проект, незалежно від його складності та об'єму робіт, необхідних для його виконання, проходить у своєму розвитку ряд певних станів: від стану, «зародження проекту», до стану, «проект вичерпано». Сукупність етапів розвитку від гене-

рації ідеї до повного завершення проекту поділяють на *фази (стадії, етапи).*

У визначенні кількості етапів та їх змісту є певні відмінності, оскільки ці характеристики залежать від умов реалізації конкретного проекту та досвіду основних учасників. Тим не менше, логіка й основна мета процесу розробки інформаційної системи практично в усіх випадках тотожні.

Виділяють наступні етапи розвитку інформаційної системи:

- розробка концепції;
- формування технічного завдання;
- фаза проектування;
- етап виготовлення;
- налагодженняв (введення системи в експлуатацію)

Розглянемо кожний з них більш детально.

Другий і частково третій етапи прийнято називати *етапами системного проектування*, а останні два (іноді сюди відносять і етап проектування) – *етапами реалізації.*

Концептуальний етап

Головним змістом роботи на цьому етапі є визначення проекту, розробка його концепції, що передбачає:

- формування ідей, постановку мети;
- формування ключової команди виконавців проекту;
- вивчення вимог та побажань замовника та інших учасників;
- збір початкових даних та аналіз існуючого стану;
- встановлення основних вимог і обмежень, необхідних матеріальних, фінансових і людських ресурсів;
- пошук вірогідних альтернатив;
- висунення пропозицій, їх експертування та затвердження.

Створення технічної пропозиції

Головним змістом цього етапу є: розробити технічну пропозицію на основі переговорів із замовником про укладення контракту. Загальний зміст робіт цього етапу:

- розробка змістовної моделі та базової структури проекту;
- уточнення та затвердження технічного завдання;
- планування, створення базової структурної моделі проекту;
- складання попереднього бюджету проекту, відносно визначеної потреби в ресурсах;
- розробка планів і графіків виконання робіт;
- укладення контракту із замовником;
- увід у дію засобів комунікації учасників проекту та контролю за перебігом робіт.

Проектування

На цьому етапі визначаються підсистеми, їх взаємозв'язки, обираються найефективніші способи виконання проекту та використання наявних ресурсів. Характерні роботи цього етапу:

- проведення базових робіт за проектом;
- розробка персональних технічних завдань виконавців;
- розробка концепції проекту;
- створення технічних специфікацій та інструкцій;
- представлення проектної розробки, експертування та затвердження.

Розробка

На цьому етапі проводяться координація й оперативний контроль робіт за проектом, створюються підсистеми, проводиться їх об'єднання та тестування. Основний зміст:

- розробка програмного забезпечення;
- проведення підготовки до впровадження системи;
- контроль і корегування основних показників проекту.

Уведення системи в експлуатацію

На цьому етапі проводяться випробування, пробна експлуатація системи в реальних умовах, ведуться перемовини про результати виконання проекту щодо можливісті укладання нових контрактів. Основні види робіт:

- комплексні випробування у граничних умовах;
- підготовка користувачів системи, що створюється;
- підготовка технічної документації, здача системи замовнику та введення її в експлуатацію;
- супровід, технічна підтримка, сервісне обслуговування;
- оцінка результатів проекту та підготовка підсумкової документації;
- вирішення спірних ситуацій та закриття робіт за проектом;
- накопичення досвіду для наступних проектів, аналіз наявних даних, визначення перспектив розвитку.

Зауваження: Початкові фази проекту мають вирішальний вплив на досягнутий результат, оскільки в них приймаються основні рішення, які визначають якість інформаційної системи. При цьому принаймні 30% внеску в кінцевий результат проекту вносять фази концепції та пропозиції, 20% - фаза проектування, 20% - фаза виготовлення, 30% - фаза здачі об'єкту та завершення проекту.

Крім того, на виявлення помилок, що були допущенні на стадії системного проектування, витрачається у два рази більше часу, ніж на наступних етапах, а їх виправлення обходиться до п'яти разів дорожче. Тому на початкових стадіях проекту розробку потрібно виконувати особливо ретельно. Найбільш часто на початкових фазах допускаються такі помилки:

- хибне розуміння інтересів замовника;
- концентрація уваги команди на несуттєвих, сторонніх інтересах;
- неправильна інтерпретація вихідної постановки задачі;
- неправильне чи недостатнє розуміння деталей;
- неповна функціональна специфікація;
- прорахунки у визначенні потрібних ресурсів і термінів виконання;
- не достатнь часті перевірки проекту на узгодженість етапів і відсутність контролю з боку замовника.

2.2. Процеси життєвого циклу інформаційних систем

Поняття життєвого циклу – одне із базових понять методології проектування ІС. Життєвий цикл ІС являє собою неперервний процес, що починається з моменту прийняття рішення про створення ІС і закінчується в момент повної конфіскації її з експлуатації.

Існує міжнародний стандарт, що регламентує життєвий цикл ІС – ISO/IEC 12207.

ISO – International Organization of Standardization (міжнародна організація по стандартизації).

IEC – International Electrotechnical Commission (міжнародна комісія по електротехніці).

Стандартом ISO/IEC 12207 визначається структура життєвого циклу, який містить процеси, дії та задачі, які повинні бути виконані під час створення ІС. За цим стандартом структура життєвого циклу заснована на трьох групах процесів:

- основними процесами життєвого циклу є придбання, постановка задач, розробка, експлуатація, супровід;
- допоміжними процесами, є такі процеси, що забезпечують виконання основних процесів, це документування, керування конфігурацією, забезпечення належної якості, верифікація, атестація, оцінка, аудит, вирішення проблем);
- організаційними процесами є керування проектами, створення інфраструктури проекту, визначення, оцінка і покращення самого життєвого циклу, навчання.

Розглянемо кожну з указаних груп більш детально.

Основні процеси життєвого циклу

Серед основних процесів життєвого циклу найбільш важливі три: розробка, експлуатація й супровід. Кожен процес характеризується певними задачами й методами їх вирішення, вихідними даними, отриманими на попередньому етапі, та результатами.

Розробка

Розробка ІС охоплює всі роботи зі створення інформаційного програмного забезпечення та його компонентів відпо-

відно до заданих вимог. Розробка інформаційного програмного забезпечення також передбачає:

- оформлення проектної, а також експлуатаційної документації;
- підготовку матеріалів, потрібних для тестування розроблених програмних продуктів;
- розробку навчальних матеріалів для персоналу.

Розробка – один із найважливіших процесів життєвого циклу ІС і, як правило, охоплює стратегічне планування, аналіз, проектування й реалізацію (програмування).

Експлуатація

Експлуатаційні роботи можна поділити на підготовчі й основні. До підготовчих відносяться:

- конфігурування бази даних і робочих місць користувачів;
- надання користувачам експлуатаційною документацією;
- Навчання персоналу.

Базові експлуатаційні роботи передбачають:

- безпосередньо експлуатацію;
- локалізацію проблем і усунення причин їх появи;
- оновлення програмного забезпечення;
- аналіз на предмет вдосконаленню системи;
- оновлення і модернізацію системи.

Супровід

Служби технічної підтримки відіграють доволі велику роль у житті будь-якої КІС. Наявність кваліфікованого технічного обслуговування на етапі експлуатації ІС є необхідною умовою для виконання поставлених перед нею завдань, причому помилки обслуговуючого персоналу можуть приводити до очевидних або прихованих фінансових утрат, порівняно з вартістю самої ІС.

Основні попередні дії при підготовці до організації технічного обслуговування ІС такі:

• виявлення ключових вузлів системи і визначення для них критичності простою. Це дозволить виділити ключові складові ІС і оптимізувати розподіл ресурсів для їх технічного обслуговування;

• визначення пріоритетних завдань технічного обслуговування та їх розподіл на внутрішні (які виконуються силами обслуговуючого підрозділу) та зовнішні (які виконуються спеціалізованими сервісними організаціями). Тобто проводиться чітке визначення кола виконавчих функцій і розподіл відповідальності;

• аналіз внутрішніх і зовнішніх ресурсів системи, необхідних для організації технічного обслуговування в рамках описаних завдань і розділу компетенції. Основні критерії для аналізу: наявність гарантій на обладнання, стан ремонтного фонду, кваліфікація персоналу;

• підготовка плану організації технічного обслуговування, в якому необхідно визначити етапи виконуючих дій, строки їх виконання, затрати на етапах, відповідальність виконавців.

Забезпечення якісного технічного обслуговування ІС потребує залучення спеціалістів високої кваліфікації, які в змозі вирішувати не тільки щоденні задачі адміністрування, але й у нештатній ситуації швидко відновлювати працездатність системи.

Допоміжні процеси

У переліку допоміжних процесів одне з основних місць займає керування конфігурацією. Це один з допоміжних процесів, які підтримує основні процеси життєвого циклу ІС, передусім процеси розробки й супроводу. При розробці проектів складних ІС, які складаються з багатьох компонентів, кожен з яких може розроблятись незалежно й мати кілька реалізацій або кілька версій однієї й тої самої реалізації, виникає проблема обліку їх зв'язків і функцій, утворення єдиної структури та забезпечення розвитку всієї системи. Керування конфігурацією дозволяє організовувати, систематично враховувати й контролювати внесення змін у різні компоненти ІС на всіх етапах її життєвого циклу.

Організаційні процеси

Керування проектом пов'язане з питаннями планування й організації робіт, утворення колективів розробників і контролю за строками і якістю виконання робіт. Технічне й організаційне забезпечення проекту передбачає:

- вибір методів та інструментальних засобів для реалізації проекту;
- визначення підходів до опису проміжних станів розробки;
- розробку методів і алгоритмів випробовувань створеного програмного забезпечення;
- навчання співробітників.

Забезпечення якості проекту залежить від проблемами верифікації, перевірки й тестування компонентів ІС.

Верифікація – це процес встановлення відповідності поточного стану розробки (досягнень на даному етапі) із вимогами розробки на цьму етапі.

Перевірка – це процес встановлення відповідності параметрів розробки поставленим вимогам. Перевірка частково тотожна з тестуванням, яке проводиться для встановлення розбіжності між справжніми й очікуваними результатами та оцінки відповідності характеристик ІС поставленим вимогам.

2.3. Структура життєвого циклу інформаційних систем

Повний життєвий цикл ІС складається, як правило, зі стратегічного планування, аналізу, проектування, реалізації, впровадження й експлуатації. У загальному випадку життєвий цикл можна розбити на ряд стадій. У принципі, цей поділ на стадії довільний. Ми розглянемо один із варіантів такого поділу, запропонований корпорацією Rational Software. Це одна з провідних фірм на ринку програмного забезпечення засобів розробки ІС (серед яких великою популярністю користується універсальний CASE- засіб Rational Rose). Згідно з методологією запропонованою Rational Software, життєвий цикл ІС містить чотири стадії:

- початкова;
- уточнююча;
- конструювальна;
- перехідна (введення в експлуатацію).

Межі кожної стадії визначені деякими моментами часу, в які необхідно приймати певні критичні рішення та повинні бути досягнуті певні ключові цілі.

На *початковій стадії* встановлюється сфера застосування системи й визначаються граничні умови. Для цього необхідно ідентифікувати всі зовнішні об'єкти, з якими відбувається взаємодія системи, яку ми розробляємо, і встановити характер цієї взаємодії на високому рівні. На початковій стадії визначають всі функціональні можливості системи й розробляється опис найбільш суттєвих з них.

Ділове застосування охоплює:

- критерії успіху розробки;
- оцінка ризику;
- оцінка ресурсів, необхідних для виконання розробки;
- календарний план з указаними строками завершення основних етапів.

На *стадії уточнення* проводиться аналіз прикладної сфери, розробляється архітектурна основа ІС.

При прийнятті будь-яких рішень, які стосуються архітектури системи, необхідно мати на увазі всю систему в цілому. Це означає, що дуже важливо описати більшість функціональних можливостей системи із врахуванням взаємозв'язку між окремими її складовими. Наприкінці стадії уточнення проводиться аналіз архітектурних рішень і способів ліквідації головних елементів ризику, які є в проекті.

На *стадії конструювання* розробляється завершений виріб, який уже можна передати користувачу. По закінченні цієї стадії визначається працездатність розробленого програмного забезпечення.

На *стадії переходу* проводиться передача розробленого програмного забезпечення користувачеві. При експлуатації розробленої системи в реальних умовах часто виникають проблеми різного характеру, які потребують додаткових робіт по внесенню коректівів у розроблений продукт. Це, як правило, пов'язано з виявленням помилок і недоопрацюванням. Наприкінці стадії переходу необхідно визначити, досягнута мета розробки чи ні.

2.4. Моделі життєвого циклу інформаційної системи

Під моделлю життєвого циклу інформаційної системи розумітимемо деяку структуру, яка визначає послідовність впровадження процесів, дій та завдань, які виконуються протягом життєвого циклу інформаційної системи, а також взаємозв'язок між цими процесами, діями та завданнями.

У стандарті ISO/IEC 12207 не конкретизують у деталях методи реалізації та виконання дій і завдань, які входять у процеси життєвого циклу інформаційної системи, а лише описують структури цього процесу. Це досить зрозуміло, оскільки регламенти стандарту є загальними для любих моделей життєвого циклу, методологій та технологій розробки. Модель життєвого циклу залежить від специфіки інформаційної системи та умов, в яких вона створюється та функціонує. Тому немає сенсу пропонувати якісь конкретні моделі життєвого циклу та методі розробки інформаційних систем для загального випадку, без прив'язування до якоїсь визначеної предметної сфері.

До цього часу найбільш поширеними стали дві основні моделі життєвого циклу:

- каскадна модель, іноді її також називають «водоспад» (waterfall);
- спіралеподібна модель.

Каскадна модель життєвого циклу інформаційної системи

Каскадна модель демонструє класичний підхід до розробки різних систем у будь-яких прикладних сферах. Для розробки інформаційних систем дана модель широко використовувалась у 70-х і першій половині 80-х років. Каскадні методи проектування добре описані в зарубіжній та вітчизняній літературі різних напрямків. Таким чином, наявність не тільки теоретичних основ, але й промислових методик і стандартів, а також використання цих методів протягом десятиріч дозволяє називати каскадні методи класичними.

Каскадна модель будується за принципом послідовної організації робіт. При цьому основною характеристикою є поділ всієї розробки на стадії, причому перехід із одної стадії на іншу проходить лише після того, як будуть повністю закінчені всі

роботи на попередній стадії. Кожна стадія завершується випуском повного набору документації, якої досить для того, аби розробка могла бути продовжена іншою групою розробників.

Основні етапи розробки каскадної моделі

За десятиріччя існування моделі «водоспад» розбиття робіт на стадії та назви цих стадій змінювались. Крім того, найбільш розумні моделі та стандарти уникали жорсткого та однозначного приписування певних робіт до конкретних етапів. Тим не менш усе ж таки можна виділити ряд стійких етапів розробки, практично не залежних від предметної сфери (Рис. 2.2):

- аналіз вимог клієнта;
- складання проекту;
- процес розробки;
- процес тестування та випробування;
- представлення готового продукту.

На першому етапі проводиться дослідження проблеми, яка має бути розв'язана, чітко формулюються всі вимоги замовника. Результатом, отриманим на даному етапі, є технічне завдання (завдання на розробку), погоджене з усіма зацікавленими сторонами.

На другому етапі розробляються проектні рішення, які задовольняють усі вимоги, що сформульовані в технічному завданні. Результатом даного етапу є комплект проектної документації, якій містить усі необхідні дані для реалізації проекту.

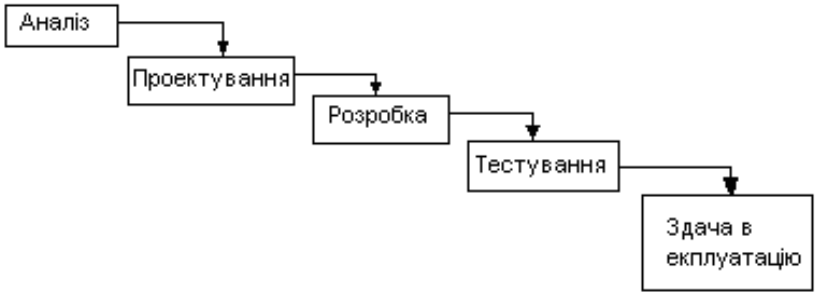


Рис. 2.2. Каскадна модель розробки ІС

Третій етап – реалізація проекту. Тут здійснюється розробка програмного забезпечення відповідно до проектного рішення, отриманого на попередньому етапі. Методи, які використовуються для реалізації, не мають принципового значення. Результатом виконання даного етапу є готовий програмний продукт.

На четвертому етапі проводиться перевірка отриманого програмного забезпечення на предмет відповідності вимогам, які заявлені в технічному завданні. Дослідницька експлуатація дозволяє виявити різного роду недоліки, які проявляються в реальних умовах роботи інформаційної системи.

Останній етап – здача готового проекту. Головне завдання цього етапу – переконати замовника, що всі його вимоги реалізовані в повному обсязіу.

Етапи робіт у рамках каскадної моделі часто називають частинами «проектного циклу» системи. Така назва виникла тому, що етапи складаються з багатьох ітераційних процедур уточнення вимог до системи і варіантів проектних рішень. Життєвий цикл самої системи суттєво складніший і більший. Він може мати довільне число циклів уточнення, зміни й доповнення вже прийнятих і реалізованих проектних рішень. У цих циклах відбувається розвиток ІС і модернізація окремих її компонентів.

Основні переваги каскадної моделі

Каскадна модель має ряд позитивних сторін, завдяки яким вона добре проявила себе під час виконання різного роду інженерних розробок і отримала широке визнання. Розглянемо основні переваги моделі «водоспад»:

- на кожному етапі формується комплект проектної документації, що задовільняє критеріям повноти і є узгодженим. На фінальних етапах також виготовляється документація користувача, що включає всі передбачені стандартами види забезпечення ІС: як організаційного, так і методичного, інформаційного, програмного, апаратного;
- етапи розробки, що проводяться за певною логічною послідовностю дозволяють встановити терміни завершення й відповідні затрати.

Каскадна модель від початку розроблялась для розв'язування різного роду інженерних задач і не втратила свого значення для прикладної області до теперішнього часу. Крім того, каскадний підхід добре зарекомендував себе й при побудові певних ІС. Маються на увазі системи, для яких на самому початку розробки можна доволі точно й повністю сформулювати всі вимоги, щоб надати розробникам свободу вибору реалізації, найліпшої з технічного погляду. До таких ІС частково належать складні розрахункові системи, системи реального часу.

Тим не менше, незважаючи на всі свої переваги, каскадна модель має ряд недоліків, що обмежують її застосування при розробці ІС. Причому ці недоліки роблять її або повністю незастосовною, або приводять до збільшення строків розробки і вартості проекту. На даний час більшість невдач програмних проектів пояснюються саме застосуванням послідовного процесу розробки.

Недоліки каскадної моделі

Перелік недоліків каскадної моделі під час її використання для розробки ІС доволі широкий. Перерахуємо їх:

- відчутна затримка в отриманні результату;
- помилкові або хибні результати на довільній стадії проявляються, зазвичай, на наступних стадіях робіт, що призводить до необхідності повернення на доопрацювання на попередньому етапі;
- важкість послідовного виконання робіт за проектом;
- надмірна інформаційна насиченість на будь-якому з етапів;
- комплікованість керування проектом;
- високі фінансові ризики та ненадійність інвестицій.

Узагалі, робота може бути повернена з будь-якого етапу на будь-який попередній етап, тому в реальному випадку каскадна схема розробки має вигляд, наведений на Рис. 2.3.

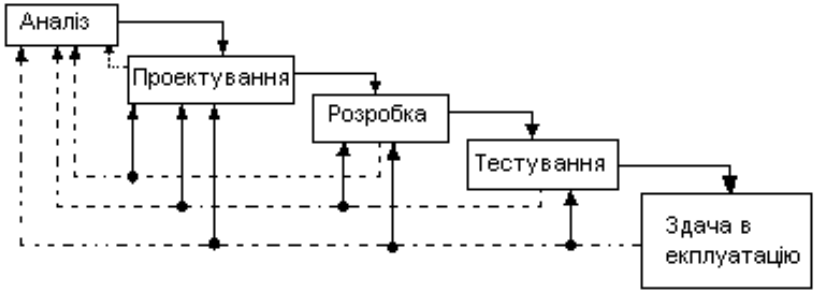


Рис. 2.3. Реальний процес розробки ІС за каскадною схемою

Спіральна модель життєвого циклу

Спіральна модель, на противагу каскадній, передбачає процес розробки ІС, що має свою ієрархію. Важливо, що значення початкових стадій життєвого циклу, як то аналіз і проектування, суттєво зростає. Ці етапи покликані перевірити та обґрунтувати реалізацію технічних рішень методом створення прототипів.

Кожна ітерація становить завершений цикл розробки, що призводить до появи внутрішньої або зовнішньої версії виробу, який удосконалюється із кожною новою ітерацією, і так аж до кінцевої системи (Рис. 2.4).

Тобто, кожен виток спіралі відповідає створенню фрагмента або версії програмного виробу, на ньому уточнюються цілі та характер проекту, визначається його якість, плануються роботи наступного витка спіралі. На кожній ітерації поглиблюються та послідовно конкретизуються деталі проекту, у результаті чого обирається єдиний варіант, який доводиться до кінцевої реалізації.

Використання спіральної моделі дозволяє здійснити перехід на наступний етап виконання проекту, не чекаючи повного завершення роботи на поточному – недороблену роботу можна буде виконати на наступній ітерації. Головне завдання кожної ітерації - якнайшвидше створити працездатний продукт, який можна показати користувачам системи. Тобто, суттєво спрощується процес унесення уточнень і доповнень у проект.

Рис. 2.4. Спіральна модель життєвого циклу ІС

Переваги спіральної моделі

Спіральний підхід до розробки програмного забезпечення дозволяє перебороти більшість недоліків каскадної моделі та забезпечує ряд додаткових можливостей, роблячи процес розробки більш гнучким.

Розглянемо переваги ітераційного підходу більш детально:

- ітераційна розробка більш суттєво спрощує внесення змін у проект при зміні вимог замовника;
- при використанні спіральної моделі окремі елементи ІС інтегруються в єдине ціле поступово. При ітераційному підході інтеграція проводиться практично безперервно. Оскільки інтеграція починається з меншої кількості елементів, то виникає значно менше проблем при її проведенні (за деякими оцінками, при використовуванні каскадної моделі розробки інтеграція займає 40% всіх затрат в кінці проекту);
- зменшення рівня ризику. На Рис. 2.5 наведені для порівняння графіки залежності рівня ризиків від часу розробки при використанні каскадного та ітераційного підходів;

- ітераційна розробка забезпечує велику гнучкість в управлінні проектом, даючи можливість внесення тактичних змін у виріб;

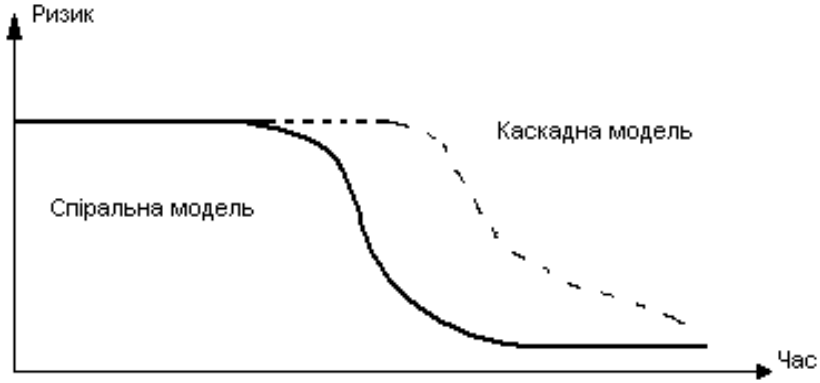


Рис. 2.5. Залежність ризиків

- спіральна модель дозволяє отримати більш надійну і стійку систему. Це пов'язано з тим, що у процесі розвитку системи помилки й слабкі місця виявляються і виправляються на кожній ітерації;
- ітераційний підхід дозволяє вдосконалювати процес розробки - аналіз, що проводиться в кінці кожної ітерації, дозволяє проводити оцінку того, що повинно бути змінено в організації розробки, і покращити її на наступній ітерації.

Проблеми, що виникають при використанні спіральної моделі

Основна проблема спірального циклу – визначення моменту переходу на наступний етап. Для її розв'язування необхідно увести часові обмеження на кожен з етапів життєвого циклу. У противному випадку процес розробки може перетворитись у безперервне вдосконалення вже зробленого. При ітер-аційному підході корисно наслідувати принцип «краще – ворог доброго».

Розділ 3. Методологія й технологія розробки інформаційних систем

Методологія створення інформаційних систем полягає в організації процесу побудови інформаційної системи й забезпеченні керування цим процесом для того, щоб гарантувати виконання вимог, як до самої системи, так і до характеристик процесу розробки.

Основні завдання, виконання яких повинна забезпечувати методологія створення корпоративних інформаційних систем (за допомогою відповідного набору інструментальних засобів), такі:

- забезпечення створення інформаційних систем, що відповідають цілям і задачам підприємства, і відповідних вимог, що пред'являються до них, по автоматизації ділових процесів;
- гарантія створення системи із заданими параметрами протягом заданого часу в рамках обумовленого заздалегідь бюджету;
- простота супроводу, модифікації й розширення системи з метою забезпечення її відповідності мінливим умовам роботи підприємства;
- забезпечення створення корпоративних інформаційних систем, що відповідають вимогам відкритості, мобільності й масштабованості;
- можливість використання в створюваній системі розроблених раніше й використаних на підприємстві інформаційних технологій (програмного забезпечення, баз даних, обчислювальної техніки, телекомунікацій).

Методології, технології й інструментальні засоби проектування (CASE-засоби) складають основу проекту будь-якої інформаційної системи. Методологія реалізується через конкретні технології і стандарти, що підтримують їх, методики й інструментальні засоби, які забезпечують виконання процесів життєвого циклу інформаційних систем.

3.1. Методологія RAD — Rapid Application Development

На ранніх етапах існування комп'ютерних інформаційних систем їх розробка проводилась на традиційних мовах про-

грамування. Проте по мірі зростання складності систем, які розробляються, із збільшенням кількості запитів користувачів (цьому значно посприяв прогрес обчислювальної техніки, а також розробка зручного графічного інтерфейсу користувача системних програм) виникла необхідність у нових засобах, що забезпечують суттєве скорочення термінів розробки. Це стало передумовою створення нового напряму в галузі програмного забезпечення – інструментальних засобів для ефективної розробки додатків. Розвиток даного напряму спричинив появу на ринку програмного забезпечення засобів автоматизації практично на всіх етапах життєвого циклу інформаційних систем.

Основні особливості методології RAD

Методологія розробки інформаційних систем, грунтується на використанні засобів ефективної розробки додатків, стала останнім часом надзвичайно поширеною й отримала назву *методології швидкої розробки додатків* – RAD (Rapid Application Development). Така методологія охоплює всі стадії життєвого циклу сучасних інформаційних систем. RAD – це набір спеціальних інструментальних засобів швидкого створення прикладних інформаційних систем, які дозволяють працювати з певним набором графічних об'єктів, які функціонально відображають певні інформаційні компоненти застосунків.

Методологія швидкої розробки застосунків – як правило, процес розробки інформаційних систем, що грунтується на трьох базових елементах:

- обмеженій групі розробників (від 2 до 10 людей);
- виробничий графік робіт, розрахований на порівняно короткий термін розробки (від 2 до 6 міс.);
- ітераційна модель розробки, заснована на тісній взаємодії із замовником, – при виконанні проекту розробники уточнюють і реалізують у продукті вимоги, що висуваються замовником.

При використанні методології RAD велике значення мають досвід і професіоналізм розробників. Група розробників повинна складатися з професіоналів тих, що мають досвід в аналізі, проектуванні, програмуванні й тестуванні програмного забезпечення.

Основні принципи методології RAD можна звести до такого:

- використовується ітераційна (спіральна) модель розробки;
- не є необхідним повне завершення робіт на кожній стадії життєвого циклу;
- у процесі розробки інформаційної системи необхідна тісна взаємодія із замовником і майбутніми користувачами;
- необхідне застосування CASE-засобів;
- необхідне застосування засобів управління конфігурацією, що полегшує внесення змін у проект та підтримку готової системи;
- необхідне використання прототипів, що дозволяє повніше з'ясувати й реалізувати потреби кінцевого користувача;
- тестування й розвиток проекту здійснюються одночасно з розробкою;
- розробка ведеться малочисленною й добре керованою командою професіоналів;
- необхідні грамотне керівництво розробкою системи, чітке планування та контроль виконання робіт.

Об'єктнозорієнтований підхід

Засоби RAD дали можливість застосувати абсолютно нову технологію (порівняно з традиційною) створення застосунків: інформаційні об'єкти формуються у вигляді діючих моделей (прототипи), їх функціонування погоджується з користувачем, а вже потім розробник може переходити безпосередньо до реалізації закінчених застосувань, не ігноруючи загальну картину спроектованої системи. Можливість використання такого підходу у більшій мірі є результатом застосування принципів об'єктно орієнтованого проектування. Застосування об'єктно орієнтовани методів дозволяє вирішити одну з головних проблем, що постають при розробці великих систем, – величезний розрив між реальним світом (областю дослідженої проблеми) і модельним середовищем.

Використання об'єктно орієнтованих методів дозволяє описати певну область (модель) як сукупность об'єктів, що об'єднують дані і методи їх обробки (процедури). Кожен об'єкт має свою власну поведінку і моделює деякий об'єкт із реального світу.

З такої точки зору об'єкт є цілком матеріальним, таким, що демонструє певну поведінку.

В об'єктному підході наголос зміщується на певні характеристики фізичної або абстрактної системи, яка є предметом програмного моделювання. Об'єкти мають цілісність, яка є непорушною. Тобто, властивості, які характеризують об'єкт та його поведінку, є незмінними.

Широку популярність об'єктно-орієнтоване програмування здобуло з появою візуальних засобів проектування, коли було забезпечене злиття (інкапсуляція) даних з процедурами, що описують поведінку реальних об'єктів. Це дозволило перейти до створення програмних систем, що є максимально подібними до реальних, і домогтися найвищого рівня абстракції. А об'єктнозорієнтоване програмування дозволяє створювати надійніші коди, оскільки у об'єктів програм існує точно визначений і жорстко контрольований інтерфейс. При розробці додатків за допомогою інструментів RAD використовується множина готових об'єктів, що зберігаються в загальнодоступному сховищі. Проте підтримується також можливість розробки нових об'єктів. Важливо, що нові об'єкти можуть розроблятися, як на базі вже існуючих, так і «з нуля».

Візуальне програмування

Застосування принципів об'єктно орієнтованого програмування дало можливість створити принципово нові засоби проектування додатків, що звуться засобами *візуального програмування.* Візуальні інструменти RAD дають можливість створювати складні користувацькі графічні інтерфейси взагалі без необхідності написання кодів програми. При цьому розробник має можливість на будь-якому етапі спостерігати те, що становить основу рішень, які ухвалюються.

Візуальні засоби розробки оперують насамперед зі стандартними об'єктами інтерфеу – вікнами, списками, текстом, що легко можна пов'язати із базами даних і відобразити графічно.

Інша група об'єктів становить стандартні елементи керування – кнопки, перемикачі, прапорці, меню тощо, які здійснюють керування даними, що відображаються. Усі ці об'єкти мають стандартний опис засобами мови програмування.

На сьогодні існує велика кількість різних візуальних засобів для розробки додатків. Але всі вони розділяються на дві групи – універсальні та спеціалізовані. До універсальних систем візуального програмування як найбільш поширені, відносять такі, як Borland Delphi і Visual Basic. Універсальними ми їх називаємо оскільки вони не орієнтовані на розробку додатків баз даних – за їх допомогою проводиться розробка застосунків майже будь-якого типу, Включно й інформаційних. При чому програми, що створюються за допомогою універсальних систем, можуть взаємодіяти майже з будь-якими системами управління базами даних. Що досягається як за допомогою драйверів ODBC або OLE DB, так і шляхом застосуванням спеціалізованих засобів (компонентів).

Спеціалізовані засоби розробки зорієнтовані на створення додатків баз даних. Причому, зазвичай, вони пов’язані із цілком визначеними системами керування базами даних. Як приклад таких систем можна навести Power Builder фірми Sybase (природно, призначений для роботи зі СКБД Sybase Anywhere Server) і Visual FoxPro фірми Microsoft.

3.2. Стандарти й методики

Однією з важливих передумов ефективного застосування інформаційних технологій є імплементація корпоративних стандартів. Корпоративними стандартами є угода про єдині правила організації технології або керування. При цьому за основу корпоративних можуть прийматися галузеві, національні й навіть міжнародні стандарти.

Види стандартів

Стандарти, що існують на сьогоднішній день, можна умовно розділити на декілька груп за наступними ознаками:

- *за предметом стандартизації.* До цієї групи відносять функціональні стандарти (стандарти мов програмування, інтерфейсів, протоколів) і стандарти організації життєво-

го циклу створення і використання інформаційних систем і програмного забезпечення;

- *за затверджуючою організацією*. Тут виділяють офіційні міжнародні, офіційні національні або державні стандарти (наприклад, ДСТУ, ANSI, IDEF0/1), стандарти міжнародних консорціумів і комітетів зі стандартизації (наприклад, консорціуму OMG), фактичні стандарти – офіційно не затверджені, але такі, які фактично діють, наприклад, таким стандартом довгий час були мова взаємодії з реляційними базами даних SQL і мова програмування С), фірмові стандарти (наприклад, Microsoft ODBC);
- *за методичним джерелом*. До цієї групи належать різного роду методичні матеріали провідних фірм-розробників програмного забезпечення, фірм-консультантів, наукових центрів, консорціумів зі стандартизації.

Методика Oracle CDM

Одним з напрямів діяльності фірми ORACLE, що вже склалися, стала розробка методологічних основ та впровадження інструментальних засобів автоматизації процесів розробки складних прикладних систем, орієнтованих на інтенсивне використання баз даних. Методика Oracle CDM є еволюцією давно розробленої версії Oracle CASE-Method, що застосовується в CASE-засобі Oracle CASE (у нових версіях – Designer/2000).

Основу CASE-технології та інструментального середовища фірми ORACLE складають:

- методологія структурного низхідного проектування, у якому розробка прикладної системи постає у вигляді послідовності певних чітко визначених етапів;
- технічна підтримка всіх етапів життєвого циклу прикладної системи, починаючи із загальних описів області і аж до отримання підтримки готового програмного продукту;
- орієнтованість на додатки в клієнт-серверній архітектурі із використанням усіх особливостей сучасних серверів баз даних, включно з декларативними обме-

женнями цілісності, Збережені процедури, баз даних і підтримка клієнтською частиною всіх сучасних стандартів і вимог до графічного інтерфейсу користувача;

- наявність центральної бази даних, *репозитарію,* для зберігання специфікацій проекту прикладної системи на всіх стадіях її розробки. Такий репозитарій є спеціальною базою данихи, що працює під керуванням СКБД ORACLE;
- можливість одночасної роботи з репозитарієм багатьох користувачів. Багатокористувальний режим підтримується стандартними засобами СКБД ORACLE. Централізоване зберігання проекта системи і керування одночасним доступом до нього всіх розробників дозволяють узгоджувати дії розробників і не допускають ситуацію, коли кожен член команди працює своєю версією проекту;
- автоматизація послідовного переходу від одної стадії розробки до наступнї. Для цього передбачені спеціальні утиліти, що дозволяють за специфікаціями концептуального рівня (моделі області) автоматично отримувати первинний варіант специфікації рівня проектування (опис структури бази даних і складу програмних модулів), щоб ґрунтуючись на ньому, після всіх уточнень і доповнень автоматично компілювати готові до виконання програми;
- автоматизація різних стандартних дій з проектування і реалізації застосування: передбачається генерація численних звітів за вмістом репозитарія, що забезпечують документування поточної версії системи на всіх етапах її розробки; за допомогою спеціальних процедур, також надається можливість перевірки специфікацій на повноту і несуперечність.

Методика Oracle CDM визначає такі фази життєвого циклу інформаційної системи:

- стратегія;
- аналіз (формулювання детальних вимог до прикладної системи);
- проектування (перетворення вимог у детальні специфікації системи);
- реалізація (написання й тестування додатку); впровадження (інсталяція нової версії прикладної системи, підготовка до початку експлуатації;
- експлуатація (підтримка додатку й стеження за ним, планування майбутніх функціональних розширень).

Методика Oracle CDM виділяє такі процеси, що протікають на протязі життєвого циклу інформаційної системи:

- визначення виробничих потреб;
- дослідження вже існуючих систем;
- визначення особливостей технічної архітектури;
- проектування та розгортання бази даних;
- проектування та програмна реалізація модулів;
- конвертування даних;
- документування ТЗ;
- тестування ТЗ;
- навчання персоналу;
- оновлення версій системи;
- техпідтримка й супровід.

Особливості методики Oracle CDM

Відзначимо основні особливості методики Oracle CDM, що визначають сферу її застосування й властиві обмеження, що висуваються для неї.

- Ступінь адаптивності CDM лімітується трьома моделями життєвого циклу:
  - *класична* – передбачає всі етапи;
  - *швидка розробка* – зорієнтована на використання інструментів моделювання й програмування Oracle;
  - *полегшений підхід* – рекомендується у разі малих проектів і можливості швидко зробити прототип додатку.

- Методика не передбачає введення додаткових завдань, які не обумовлені в CDM, і їх прив'язку до останніх.
- Усі моделі життєвого циклу фактично є каскадними. Навіть «лайт підхід», не дивлячись на можливість інтеграції виконання дій з прототипування, зберігає загальний послідовний і детермінований порядок виконання завдань.
- Методика не обов'язкова, але вважається фірмовим стандартом. За умови формального застосування ступінь обов'язковості повністю корелює з обмеженнями можливостей адаптації.
- Прикладна система иоже розглядатися в більшості випадків як програмно-технічна система – наприклад, можливість виконувати організаційно-структурні перетворення, що практично завжди відбуваються за умов перехоуі до нової інформаційної системи, у даній методиці відсутня.
- CDM найтіснішим чином спирається на використання інструментарію Oracle, незважаючи на твердження про просте пристосування CDM до проектів, у яких використовується інший комплект інструментальних засобів.
- Методика Oracle CDM є дуже конкретним й деталізованим до рівня заготовок проектних документів, матеріалом, що розрахований на пряме використання в проектах інформаційних систем, що спираються на інструментальні засоби і СКБД фірми Oracle.

Міжнародний стандарт ISO/IEC 12207: 1995-08-01

Перша редакція ISO 12207 була підготовлена в 1995 р. об'єднаним технічним комітетом ISO/IEC JTC1 «Інформаційні технології, підкомітет SC7, проектування програмного забезпечення».

За визначенням, ISO 12207 - базовий стандарт процесів життєвого циклу ПО, орієнтований на різні види ПО й типи проектів автоматизованих систем, в яких ПО є однією зі складо-

вих частин. Стандарт визначає стратегію й загальний порядок у створенні й експлуатації ПО, він охоплює життєвий цикл від концептуалізації ідей до завершення проекту. Доцільність сумісного використання стандартів на інформаційні системи й на ПО обумовлюється одним з положень ISO 12207, відповідно якому процеси, що використовуються під час життєвого циклу ПО, мають бути сумісні з процесами, що використовуються під час життєвого циклу автоматизованої системи.

Згідно ISO 12207, система - це об'єднання одного або декількох процесів, апаратних засобів, програмного забезпечення, устаткування і людей для забезпечення можливості задоволення певних потреб або цілей.

На відміну від Oracle CDM, стандарт ISO 12207 однаково зорієнтований на організацію дій кожної з двох сторін: постачальника (розробника) і покупця (користувача); він може бути застосований і у тому випадку, коли обидві сторони з однієї організації.

У стандарті ISO 12207 не передбачено яких-небудь етапів (фаз або стадій) життєвого циклу інформаційної системи. Даний стандарт визначає лише ряд процесів, причому в порівнянні з Oracle CDM стандарт ISO 12207 складається з крупніших узагальнених процесів: придбання, постачання, розробка і тому подібне.

Згідно з ISO 12207, кожен процес підрозділяється на ряд дій, а кожна дія — на ряд завдань. Важливою особливістю ISO 12207 в порівнянні з CDM є те, що кожен процес, дія або завдання ініціюються й виконуються іншим процесом у міру необхідності, причому немає заздалегідь певних послідовностей.

Особливості стандарту ISO 12207

Вищесказане дозволяє сформулювати такі особливості стандарту ISO 12207:

- Стандарт ISO 12207 має динамічний характер, обумовлений способом визначення послідовності виконання процесів і завдань, при якому один процес при необхідності викликає інший або його частина. Такий характер дозволяє реалізувати будь-яку модель життєвого циклу.

- Стандарт ISO 12207 забезпечує максимальний ступінь адаптивності. Множина процесів і завдань сконструйована так, що можлива їх адаптація відповідно до конкретних проектів інформаційних систем. Ця адаптація зводиться до виключення процесів, видів діяльності і завдань, непридатних в конкретному проекті.
- Стандарт принципово не містить опису конкретних методів дій, а тим більше заготовок розв'язків або документації. Він лише описує архітектуру процесів життєвого циклу програмного забезпечення, але не конкретизує в деталях, як реалізовувати, або виконувати послуги і завдання, включені в процеси. Даний стандарт не вказує імена, формати або точний зміст отримуваної документації. Рішення такого типу приймаються сторонами, що використовують стандарт.
- Ступінь обов'язковості даного стандарту такий: після рішення організації про застосування ISO 12207 в якості умови торгових відносин є її відповідальність за вказівку мінімального набору потрібних процесів і завдань, які забезпечують узгодженість із цим стандартом.
- Стандарт містить гранично мало описів, спрямованих на проектування бази даних. Це можна вважати виправданим, оскільки різні системи й різні прикладні комплекси програмного забезпечення можуть не тільки використовувати доволі специфічні типи баз даних, але і взагалі не використовувати базу даних.

Цінність стандарту ISO 12207 в тому, що він містить набори завдань, характеристик якості, критеріїв оцінки й т. ін., які дають усебічне охоплення проектних ситуацій. Наприклад, при виконанні аналізу вимог до системи передбачається, що:

- розглядається сфера застосування системи для визначення вимог, які висуваються до системи;
- специфікація вимог системи повинна описувати: функції і можливості системи, сфери застосування системи, організаційні вимоги та вимоги користу-

вача, безпеку, захищеність, людські чинники, ергономіку, зв'язки, операції і вимоги супроводу; проектні обмеження та кваліфікаційні вимоги.

Далі, при виконанні аналізу вимог до програмного забезпечення передбачено 11 класів характеристик якості, які використовуються пізніше для забезпеченні якості.

При цьому розробник повинен установити й задокументувати у вигляді вимог до програмного забезпечення наступні специфікації та характеристики:

- функціональні й можливі специфікації, включаючи виконання, фізичні характеристики і умови середовища експлуатації, при яких одиниця програмного забезпечення має бути виконана;
- зовнішні зв'язки (інтерфейси) з одиницею програмного забезпечення;
- вимоги кваліфікації;
- специфікації надійності, включаючи специфікації, пов'язані з методами функціонування й супроводу;
- специфікації захищеності, включаючи специфікації, пов'язані з компрометацією точності інформації;
- визначення даних і вимог до бази даних;
- документацію користувача;
- робота користувача й вимоги виконання;
- вимоги сервісу користувача.

3.3. Профілі відкритих інформаційних систем

Створення, підтримка та розвиток сучасних великих інформаційних систем грунтується на методології побудови *відкритих* систем. Відкриті інформаційні системи вибудовуються в процесі інформатизації всіх ключових сфер сучасного суспільства: органів державного управління, фінансової сфери, інформаційної трансформації суспільства, виробничої сфери, науки та освіти. Розвиток і застосування відкритих інформаційних систем тісно пов'язані із застосуванням стандартів на основі методології функціональних стандартів інформаційних технологій.

Профіль інформаційної системи

При створенні й розвитку складних, розподілених, тиражованих інформаційних систем потрібне гнучке формування й застосування гармонізованих сукупностей базових стандартів і нормативних документів різного рівня, виділення в них вимог і рекомендацій, необхідних для реалізації заданих функцій системи. Для уніфікації та регламентації такі сукупності базових стандартів повинні адаптуватися і конкретизуватися стосовно певних класів проектів, функцій, процесів і компонентів системи. У зв'язку з цим виділилося і сформувалося поняття *профілю* інформаційної системи як основного інструменту функціональної стандартизації.

Профіль — це сукупність декількох (або підмножини одного) базових стандартів з чітко певними і гармонізованими підмножинами обов'язкових і факультативних можливостей, призначених для реалізації заданої функції або групи функцій.

Профіль формується, виходячи з функціональних характеристик об'єкту стандартизації. У профілі виділяються й установлюються допустимі можливості та значення параметрів кожного базового стандарту або нормативного документа, що входить до профілю.

Профіль не повинен суперечити використаним у ньому базовим стандартам і нормативним документам. Він повинен застосовувати вибрані з альтернативних варіантів необов'язкові можливості і значення параметрів в межах допустимих.

На базі однієї сукупності базових стандартів можуть формуватися і затверджуватися різні профілі для різних проектів інформаційних систем. Обмеження базових документів профілю та їх узгодженість, проведена розробниками профілю, повинні забезпечувати якість, сумісність і коректну взаємодію окремих компонентів системи.

Базові стандарти й профілі, залежно від проблемно-зорієнтованої сфери застосування інформаційних систем, можуть використовуватися як безпосередні директивні, керівні або рекомендаційні документи, а також як нормативна база, необхідна при виборі або розробці засобів автоматизації технологічних етапів або процесів створення, супроводу й розвитку інформаційних систем.

Зазвичай розглядають дві групи профілів:

- регламентуючі архітектуру й структуру інформаційної системи
- регламентуючі процеси проектування, розробки, застосування супроводу й розвитку системи.

Залежно від сфери застосування профілі можуть мати різні категорії та різні статуси твердження:

- профілі конкретної інформаційної системи, що визначають стандартизовані проектні рішення в межах даного проекту;
- профілі інформаційної системи, призначені для виконання деякого класу прикладних завдань.

Принципи формування профілю інформаційної системи

Використання профілів інформаційних систем покликане вирішити наступні завдання:

- зниження трудомісткості проектів;
- підвищення якості компонентів інформаційної системи
- забезпечення розширюваності й масштабованості систем, що розробляються;
- забезпечення можливості функціональної інтеграції в інформаційну систему завдань, які раніше вирішувалися роздільно;
- забезпечення мобільності прикладного програмного забезпечення.

Залежно від того, які з указаних завдань найбільш пріоритетні, проводиться вибір стандартів і документів для формування профілю. Підходи до формування профілів інформаційних систем можуть бути різними. У міжнародній функціональній стандартизації інформаційних технологій прийнято досить жорстке поняття профілю. Вважається, що його основою можуть бути тільки міжнародні й національні, затверджені стандарти. Використання стандартів де-факто й нормативних документів фірм не допускається.

Еталонна модель середовища відкритих систем (OSE/RM) визначає розділення будь-якої інформаційної системи на дві

складові: *додатки* (прикладні програми й програмні комплекси) і *середовище,* в якому ці додатки функціонують.

Між додатками і середовищем визначаються стандартизовані інтерфейси – Application Program Interface (API), які є необхідною частиною профілів будь-якої відкритої системи. Крім того, у профілях можуть бути визначені уніфіковані інтерфейси взаємодії функціональних частин і інтерфейси взаємодії між компонентами середовища системи. Специфікації виконуваних функцій та інтерфейсів взаємодії можуть бути оформлені у вигляді профілів компонентів системи. Отже, профілі інформаційної системи з ієрархічною структурою можуть містити:

- стандартизовані описи функцій, що виконуються даною системою; функції взаємодії системи із зовнішнім для неї середовищем;
- стандартизовані інтерфейси між додатками й середовищем інформаційної системи;
- профілі окремих функціональних компонентів, що входять у систему.

Структура профілів інформаційних систем

Розробка й застосування профілів є органічною частиною процесів проектування, розробки й супроводу інформаційних систем. Профілі характеризують кожну конкретну інформаційну систему на всіх стадіях її життєвого циклу, задаючи узгоджений набір базових стандартів, яким повинні відповідати система та її компоненти.

У профіль конкретної системи включаються специфікації компонентів, розроблених у складі даного проекту, і специфікації використаних готових програмних і апаратних засобів, якщо ці засоби не специфіковані відповідними стандартами. Після завершення проектування й випробувань системи, у ході яких перевіряється її відповідність профілю, профіль застосовується як основний інструмент супроводу системи при експлуатації, модернізації та розвитку.

Загальна структура профілю інформаційної системи

Формування й застосування профілів конкретних інформаційних систем виконується на основі використання міжнародних і національних стандартів, відомчих нормативних документів, а також стандартів де-факто за умови доступності відповідних ним специфікацій. Для забезпечення коректного застосування профілів їх описи повинні містити:

- визначення цілей використання даного профілю;
- точне перерахування функцій об'єкта або процесу стандартизації, що визначається даним профілем;
- формалізовані сценарії застосування базових стандартів і специфікацій, включених у даний профіль;
- зведення вимог до інформаційної системи або її компонентів, що визначають їхню відповідність профілю;
- нормативні посилання на конкретний набір стандартів і інших нормативних документів з точною вказівкою вживаних редакцій і обмежень здатних уплинути на досягнення коректної взаємодії об'єктів стандартизації при використанні даного профілю;
- інформаційні посилання на всі початкові документи.

На стадіях життєвого циклу інформаційної системи вибираються й потім застосовуються основні функціональні профілі:

- профіль прикладного програмного забезпечення;
- профіль середовища інформаційної системи;
- профіль захисту інформації в інформаційній системі;
- профіль інструментальних засобів, вбудованих у інформаційну систему.

Профіль прикладного програмного забезпечення

Прикладне програмне забезпечення завжди є проблемно-орієнтованим і визначає основні функції інформаційної системи. Функціональні профілі системи повинні включати узгоджені базові стандарти. При використанні функціональних профілів інформаційних систем слід ще мати на увазі узгодження цих профілів між собою. Необхідність такого узгодження виникає, зокрема, при використанні стандартизованих API, зокрема інтер-

фейсів додатків із середовищем їх функціонування та із засобами захисту інформації. При узгодженні функціональних профілів можливі також уточнення профілю середовища системи й профілю вбудованих інструментальних засобів створення, супроводу та розвитку прикладного програмного забезпечення.

Профіль середовища інформаційної системи

Профіль середовища інформаційної системи повинен визначати її архітектуру відповідно до вибраної моделі обробки даних.

Стандарти інтерфейсів додатків із середовищем API мають бути визначені по функціональних областях профілів інформаційної системи. Декомпозиція структури середовища функціонування системи на складові частини, що виконується на стадії ескізного проектування, дозволяє деталізувати профіль середовища інформаційної системи по функціональних областях еталонної моделі OSE/RM:

- область графічного інтерфейсу користувача;
- область реляційних або об'єктнозорієнтованих СКБД (наприклад, стандарт мови SQL-92 і специфікації доступу до різних баз даних);
- область операційних систем з урахуванням мережевих функцій, що виконуються на рівні операційної системи;
- область телекомунікаційного середовища в частині послуг і служб прикладного рівня: електронної пошти, доступу до віддалених баз даних, передавання файлів, доступу до файлів і керування файлами.

Профіль середовища розподіленої системи повинен включати стандарти протоколів транспортного рівня, стандарти локальних мереж (наприклад, стандарт Ethernet IEEE 802.3 або стандарт Fast Ethernet IEEE 802.3u), а також стандарти засобів сполучення проектованої інформаційної системи з мережами передавання даних загального призначення. Вибір апаратних платформ інформаційної системи пов'язаний з визначенням їх параметрів: обчислювальній потужності серверів і робочих станцій у відповідності з проектними рішеннями по розділенню функцій між клієнтами і серверами; ступені масштабованості

апаратних платформ; надійності. Профіль середовища повинен містити стандарти, що визначають параметри технічних засобів і способи їх вимірювання (наприклад, стандартні тести вимірювання продуктивності).

Профіль захисту інформації

Профіль захисту інформації повинен забезпечувати реалізацію політики інформаційної безпеки, що розробляється відповідно до необхідної категорії безпеки й критеріями безпеки. Побудова профілю захисту інформації в розподілених системах клієнт-сервер методично пов'язана з точним визначенням компонентів системи, відповідальних за ті або інші функції, служби й послуги, і засобів захисту інформації, вбудованих у ці компоненти. Функціональна область захисту інформації має такі функції захисту, що реалізовуються різними компонентами системи:

- функції, що реалізуються операційною системою;
- захист від несанкціонованого доступу, що реалізується на рівні програмного забезпечення проміжного шару;
- керування даними, що реалізується СКБД;
- захист програмних засобів, включаючи засоби захисту від вірусів;
- захист інформації при обміні даними в розподілених системах, включаючи криптографічні функції;
- адміністрування засобів безпеки.

Профіль захисту інформації повинен містити вказівки на методи й засоби виявлення у вживаних апаратних і програмних засобах можливостей, що не декларуються. Профіль повинен також містити вказівки на методи й засоби резервного копіювання інформації й відновлення інформації при відмовах і збоях апаратури системи.

Профіль інструментальних засобів

Профіль інструментальних засобів, вбудованих в інформаційну систему, повинен відображати рішення щодо вибору методології та технології створення, супроводження та розвитку інформаційної системи. У цьому профілі повинні міститися

посилання на опис вибраної методології й технології, виконаний на стадії ескізного проектування системи.

Склад інструментальних засобів визначається на підставі рішень і нормативних документів про організацію супроводу й розвитку інформаційної системи. При цьому мають бути враховані правила й порядок, що регламентують унесення змін до діючих систем. Функціональна область профілю інструментальних засобів, вбудованих у систему, охоплює функції централізованого керування і адміністрування, пов'язані з:

- контролем продуктивності й коректності функціонування системи в цілому;
- керуванням конфігурацією прикладного програмного забезпечення, тиражуванням версій;
- керуванням доступом користувачів до ресурсів системи і конфігурацією ресурсів;
- переналаштуванням додатків у зв'язку зі змінами прикладних функцій інформаційної системи;
- налаштуванням призначених для користувача інтерфейсів (генерацією екранних форм і звітів);
- веденням баз даних системи;
- відновленням працездатності системи після збоїв і аварій.

Модуль 2. «Проектування та використання баз даних»
Розділ 4. Реляційні бази даних

З розвитком обчислювальної техніки змінювалися й основні напрями її використання. Спочатку засоби обчислювальної техніки планувалося використовувати для виконання різного роду математичних обчислень, які неможливо провести «вручну» за розумний час. Розвиток цього напряму призвів до розвитку розділів математики, пов'язаних з числовими методами обчислень, і до появи алгоритмічних мов і зручних для реалізації алгоритмів числових методів (однією з найбільш популярних мов програмування такого типу є Fortran, до цих пір широко застосовується для наукових розрахунків).

Потім, зі збільшенням можливостей і зменшенням вартості обчислювальних засобів, отримав розвиток другий напрям, пов'язаний із використанням засобів обчислювальної техніки в автоматизованих інформаційних системах. Тут обчислювальні можливості комп'ютерів відходять на другий план — основні функції обчислювальних засобів у інформаційних системах полягають у підтримці надійного зберігання інформації, виконанні специфічних для даного додатку перетворень інформації або обчислень, і наданні користувачам зручного та легкого інтерфейсу.

Незважаючи на те, що складність обчислень, які виконуються в інформаційних системах, несумірно нижча, ніж при проведенні наукових розрахунків, вимоги до обчислювальної потужності комп'ютерів у таких системах збільшуються. Це пов'язано з тим, що обсяги оброблюваної інформації, як правило, достатньо великі, а сама інформація має складну структуру. Цим же пояснюється істотне збільшення вимог до обсягу як оперативної пам'яті, так і пристроїв постійного зберігання інформації.

З часом саме другий напрям, пов'язаний зі зберіганням і обробкою даних, став домінуючим, особливо після появи персональних комп’ютерів.

4.1. Бази даних: основні відомості

Розвиток комп'ютерних технологій, пов'язаних зі зберіганням і обробкою даних, призвів до появи в кінці 60-х — початку 70-х років спеціалізованого програмного забезпечення, що отримало

назву *систем керування базами даних* (СКБД) *(DataBase Management Systems* — DBMS). СКБД дозволяють структурувати, систематизувати і організовувати дані для їх комп'ютерного зберігання і обробки. Саме системи керування базами даних є основою практично будь-якої інформаційної системи.

СКБД можна визначити як певну систему керування даними, яка володіє такими властивостями:

- підтримкою логічно узгодженого набору файлів;
- забезпеченням мови маніпулювання даними;
- відновленням інформації після різного роду збоїв;
- забезпеченням паралельної роботи декількох користувачів.

Зазвичай сучасна СКБД складається з таких компонентів (див. Рис. 4.1):

- *ядро*, що відповідає за керування даними в зовнішній та оперативній пам'яті й журналізацію;
- *процесор мови бази даних*, що забезпечує оптимізацію запитів на отримання й зміну даних і створення, як правило, машинно-незалежного внутрішнього коду, що виконується;
- *підсистему підтримки часу виконання*, яка інтерпретує програми маніпуляції даними, що створюють користувацький інтерфейс зі СКБД;
- а також *сервісні програми* (зовнішні утиліти), які забезпечують ряд додаткових можливостей з обслуговування інформаційної системи

Основні функції СКБД

До основних функцій, що виконуються системами керування базами даних, зазвичай відносять такі:

- безпосереднє керування даними в зовнішній пам'яті;
- керування буферами оперативної пам'яті;
- керування транзакціями;
- протоколювання;
- підтримка мов баз даних.

Розглянемо кожну з указаних функцій детальніше.

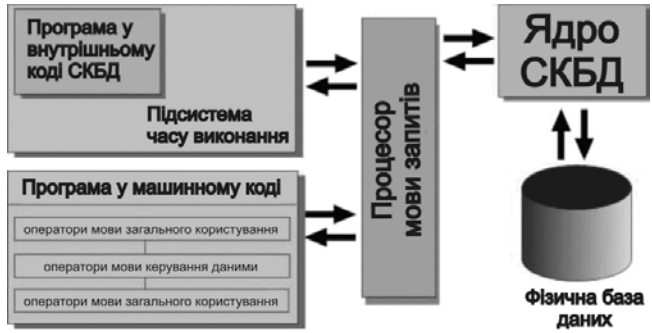


Рис. 4.1. Компоненти СКБД

Безпосереднє керування даними в зовнішній пам'яті

Функція безпосереднього керування даними в зовнішній пам'яті забезпечує необхідні структури зовнішньої пам'яті (постійні запам’ятовуючі пристрої — як правило, магнітні диски) як для зберігання даних, що безпосередньо входять у базу даних, так і для службових цілей, наприклад для прискорення доступу до даних у деяких випадках (зазвичай для цього використовуються індекси). Причому користувачам бази даних у загальному випадку не потрібно знати, чи використовує СКБД файлову систему, і якщо використовує, то як організовані файли. Зазвичай СКБД підтримує власну систему іменування об'єктів бази даних. Залежно від способу реалізації СКБД може або використовувати можливості існуючих файлових систем, або працювати з пристроями зовнішньої пам'яті на низькому рівні.

Керування буферами оперативної пам'яті

Обсяг інформації, що зберігається в базі даних, з якою працює СКБД, зазвичай достатньо великий і практично завжди перевищує доступний об'єм оперативної пам'яті. При цьому час доступу до даних, що зберігаються в оперативній пам'яті, істотно менший, ніж до даних, що зберігаються на пристроях зовнішньої пам'яті. Очевидно, що якщо при зверненні до будь-якого елементу даних буде проводитися обмін із зовнішньою пам'яттю, то вся система працюватиме зі швидкістю пристрою зовнішньої пам'яті.

Збільшення швидкості обміну даними можна досягти, використовуючи буферизацію даних в оперативній пам'яті. При цьому, навіть якщо операційна система проводить загальносистемну буферизацію (як у разі ОС UNIX), цього недостатньо для цілей СКБД, яка має в своєму розпорядженні набагато більшу інформацію про корисність буферизації тієї або іншої частини бази даних. Тому в СКБД зазвичай підтримується власний набір буферів оперативної пам'яті з власним механізмом заміни буферів.

Існує напрям розвитку СКБД, зорієнтований на постійну присутність в оперативній пам'яті всієї інформації з бази даних. Цей напрям ґрунтується на припущенні, що в майбутньому об'єм оперативної пам'яті комп'ютерів буде настільки великий, що буферизація стане непотрібна. Якщо виходити з темпів зниження цін на оперативну пам'ять, то такі СКБД справді зможуть стати актуальними в найбижчому майбутньому.

Керування транзакціями

Транзакцією називається послідовність операцій над базою даних, що розглядається СКБД, як єдине ціле. Якщо всі операції успішно виконані, то транзакція також вважається за успішно виконану й СКБД *фіксує* (COMMIT) усі зміни даних, проведені цією транзакцією (тобто заносить зміни в зовнішню пам'ять). Якщо ж хоча б одна операція транзакції закінчується невдачею, то транзакція вважається за невиконану й проводиться *відкат* (ROLLBACK) — відміна всіх змін даних, проведених у ході виконання транзакції, і повернення бази даних до стану до початку виконання транзакції.

Керування транзакціями необхідне для підтримки логічної цілісності бази даних. Підтримка механізму транзакцій є обов'язковою умовою навіть однокористувацьких, а тим більше для багатокористувацьких СКБД. Та властивість, що кожна транзакція починається при цілісному стані бази даних і залишає цей стан цілісним після свого завершення, робить дуже зручним використання поняття транзакції як одиниці активності користувача по відношенню до бази даних. При відповідному управлінні транзакціями, що паралельно виконуються, з боку СКБД кожен з

користувачів може, в принципі, відчувати себе єдиним користувачем СКБД.

З керуванням транзакціями в багатокористувацькій СКБД пов'язані важливі поняття *серіалізації транзакцій* і *серійного плану виконання суміші транзакцій.* Під серіалізацією паралельно виконуваних транзакції розуміється таке планування їх роботи, при якому сумарний результат суміші транзакцій еквівалентний результату їх деякого послідовного виконання. Серійний план виконання суміші транзакцій - це такий план, який призводить до серіалізації транзакцій. Зрозуміло, що якщо вдається домогтися справді серійного виконання суміші транзакцій, то для кожного користувача, за ініціативою якого утворена транзакція, присутність інших транзакції буде непомітною (якщо не рахувати деякого вповільнення роботи в порівнянні з однокористувацьким режимом).

Існує декілька базових алгоритмів серіалізації транзакцій. У централізованих СКБД найбільш поширені алгоритми, засновані на синхронізаційних захопленнях об'єктів бази даних. При використанні будь-якого алгоритму серіалізації можливі конфлікти між декількома транзакціями по доступу до об'єктів бази даних. У цьому випадку для підтримки серіалізації необхідно виконати відкат однієї або декількох транзакцій. Це один з випадків, коли користувач багатокористувацької СКБД може реально (і достатньо неприємно) відчути присутність у системі транзакцій інших користувачів.

Журналізація

Одна з основних вимог до СКБД – надійність зберігання даних у зовнішній пам'яті. Під надійністю зберігання розуміється те, що СКБД має бути у змозі відновити останній узгоджений стан бази даних після будь-якого апаратного або програмного збою. Апаратні збої зазвичай розділяються на два види:

- *м'які збої,* пов'язані з раптовою зупинкою роботи комп'ютера. Зазвичай вони є наслідком раптового вимкнення живлення або «зависання» операційної системи (що особливо характерно для операційних систем Windows);
- *жорсткі збої* характеризуються втратою інформації на носіях зовнішньої пам'яті.

Програмні збої зазвичай виникають унаслідок помилок у програмах. Причому ці помилки можуть бути як у самій СКБД, що може привести до аварійного завершення її роботи, так і в призначеній для користувача програмі. Перший випадок можна розглядати як різновид м'якого апаратного збою. У другому випадку незавершеною залишається тільки одна транзакція.

У будь-якому випадку для відновлення інформації в базі даних необхідно мати деяку додаткову інформацію. Отже, для підтримки надійності зберігання даних потрібна надмірність даних. Причому та частина інформації, яка використовується для відновлення, повинна зберігатися особливо надійно. Найбільш поширеним методом підтримки такої надмірної інформації є ведення журналу змін бази даних. *Журнал* - особлива частиною бази даних, недоступна користувачам СКБД і підтримувана з особливою ретельністю (іноді використовуються дві копії журналу, що розташовуються на різних фізичних дисках), до якого надходять записи про всі зміни основної частини бази даних.

У різних СКБД зміни бази даних журналізуються на різних рівнях: іноді запис в журналі відповідає деякій логічній операції зміни бази даних, іноді — мінімальній внутрішній операції модифікації сторінки зовнішньої пам'яті. Можуть також використовуватися одночасно обидва підходи. У всіх випадках дотримуються стратегії «випереджуючого» запису в журнал (так званого протоколу Write Ahead Log — WAL). Ця стратегія полягає в тому, що запис про зміну будь-якого об’єкта бази даних має бути занесений у журнал до того, як буде виконана і зафіксована зміна цього об'єкта. Якщо у СКБД коректно дотримується протокол WAL, то за допомогою журналу можна розв’язати всі проблеми відновлення бази даних після будь-якого збою.

Для відновлення бази даних після жорсткого збою використовують журнал і архівну копію бази даних. Архівна копія — це повна копія бази даних до моменту початку заповнення журналу (хоча є багато варіантів трактування сенсу архівної копії). Для нормального відновлення бази даних після жорсткого збою природно, необхідно, щоб журнал не пропав. Тоді відновлення бази даних полягає в тому, що, виходячи з архівної копії, за журналом відтворюється робота всіх транз-

акцій, які закінчилися до моменту збою. У принципі можна навіть відтворити роботу незавершених транзакції та продовжити їх роботу після завершення відновлення. Проте в реальних системах це зазвичай не робиться, оскільки процес відновлення після жорсткого збою достатньо тривалий.

Підтримка мов баз даних

Для роботи з інформацією, що зберігається в базі даних, використовуються спеціальні мови, що носять загальну назву *мов баз даних.* Найчастіше виділяються дві мови:

- мова *визначення схем даних* (Schema Definition Language, SDL) слугує, в основному, для визначення логічної структури бази даних;
- мова маніпулювання даними (Data Manipulation Language, DML) містить набір операторів маніпулювання даними, тобто операторів, які дозволяють заносити дані в базу, а також вилучати, модифікувати або вибирати існуючі дані.

Декілька різних спеціалізованих мов баз даних підтримувалося лише в ранніх СКБД. У сучасних СКБД зазвичай підтримується єдина інтегрована мова, що містить усі необхідні засоби для роботи з базою даних, починаючи від її створення, і забезпечує базовий призначений для користувача інтерфейс з базами даних. Стандартною мовою найбільш поширених на даний час реляційних СКБД є мова SQL (Structured Query Language). Тобто вказані вище мови баз даних на сьогодні фактично є підмножинами єдиної стандартної мови SQL.

Мова SQL дозволяє визначати схему реляційної бази даних і маніпулювати даними. При цьому іменування об'єктів бази даних (для реляційної бази даних – іменування таблиць і їх полів) підтримується на мовному рівні в тому сенсі, що компілятор мови SQL проводить перетворення імен об'єктів у їх внутрішні ідентифікатори на підставі спеціальних підтримуваних службових таблиць-каталогів.

Мова SQL містить спеціальні засоби визначення обмеження цілісності бази даних. Знову ж таки обмеження цілісності зберігаються в спеціальних таблицях-каталогах, і забезпечення контролю цілісності бази даних проводиться на мовному рівні – при компіляції операторів модифікації бази даних компілятор

SQL на підставі наявних у базі даних обмеженнях цілісності генерує відповідний програмний код.

Спеціальні оператори мови SQL дозволяють визначати так звані *представлення* бази даних, що фактично зберігаються в базі даних у вигляді запитів (результатом будь-якого запиту до реляційної бази даних є таблиця) з іменованими стовпцями, що називаються *полями.* Для користувача представлення є такою самою таблицею, як будь-яка базова таблиця, що зберігається в базі даних, але за допомогою представлень можна обмежити або, навпаки, розширити видимість даних для конкретного користувача. Підтримка представлень проводиться також на мовному рівні.

Нарешті, авторизація доступу до об'єктів бази даних проводиться також на основі спеціального набору операторів SQL. Ідея полягає в тому, що для виконання операторів SQL різного вигляду користувач повинен володіти різними повноваженнями. Користувач, що створив таблицю бази даних, володіє повним набором повноважень для роботи з даною таблицею. До числа цих повноважень входить повноваження на передачу всіх або частини повноважень іншим користувачам, включаючи повноваження на передачу повноважень. Повноваження користувачів описуються в спеціальних таблицях-каталогах, контроль повноважень підтримується на мовному рівні.

4.2. Моделі даних - ієрархічна, мережева, реляційна й постреляційна, багатомірна схема

В основі будь-якої СКБД лежить певна модель даних. Як правило, використовуються мережева, ієрархічна, реляційна модель або комбінація цих моделей.

Ієрархічна й мережева моделі даних почали застосовуватися в системах керування базами даних на початку 60-х років. На початку 70-х років була запропонована реляційна модель даних. Ці три моделі розрізняються в основному способами подання взаємозв'язків між об'єктами. Найстарші системи засновані на ієрархічній моделі даних. До них ставляться найпоширеніші СКБД, розроблені для великих ЕОМ, наприклад СКБД "Ока", СКБД АИСОРИ (розробка ВНИИГМИ-МЦД). Реляційними СКБД є DB2, Oracle, Paradox, Access, FoxPro й ін. Мережевою є СКБД СЕТОР.

Ієрархічна модель даних (Рис. 4.2) будується за принципом ієрархії типів об'єктів, тобто один тип об'єкта є головним, а інші, що перебувають на нижчих рівнях ієрархії, – підлеглими. Ієрархічна модель даних організує дані у вигляді ієрархічної деревоподібної структури. Ця структура будується з вузлів і гілок. Вузол являє собою сукупність атрибутів даних, що описують деякий об'єкт. Найвищий вузол в ієрархічній деревоподібній структурі називається коренем. Залежні вузли розташовуються на більш низьких рівнях дерева. Залежні вузли можуть додаватися як у вертикальному, так й у горизонтальному напрямку без усяких обмежень. Зв'язки (з'єднання) між вузлами унікальні. Тому ієрархічна модель даних забезпечує тільки лінійні шляхи доступу до даних і між головними й підлеглими типами об'єкта встановлюється лінійний взаємозв'язок "один до багатьох". Кожен екземпляр кореневого вузла утворює початок запису логічної бази даних, тобто ієрархічна база даних складається з декількох дерев. До головних переваг ієрархічної моделі даних можна віднести простоту розуміння й використання, оскільки користувачі систем обробки даних добре знайомі з ієрархічними структурами. Недоліки моделі – це громіздкі структури при складних БД і, як правило, - зберігання надлишкових даних. Наприклад, опис однакових комплектуючих (гайки, болти, ін.) у різних блоках (Рис. 4.2).

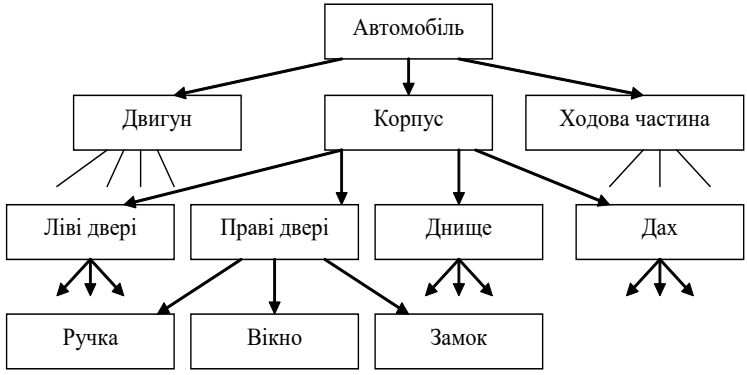


Рис.4.2. Фрагмент ієрархічної БД

У *мережевій моделі* даних (Рис. 4.3) поняття головного й підлеглого об'єктів трохи розширені. Будь-який об'єкт може бути й головним, і підлеглим (у мережевій моделі головний об'єкт позначається терміном "власник набору", а підлеглий - терміном "член набору"). Той самий об'єкт може одночасно, виступати й у ролі власника, і в ролі члена набору. Це означає, що кожен об'єкт може брати участь у будь-якій кількості взаємозв'язків. База даних складається з декількох областей. Область містить записи. А запис складається з полів, набір же, який поєднує записи, може розміщуватися в одній або декількох областях. Перевага мережевої моделі – простота реалізації взаємозв'язків, що часто зустрічаються в реальному світі й закладаються в БД. Основний недолік мережевої моделі полягає в складності керування даними, у тому числі можлива втрата незалежності даних при реорганізації бази даних.

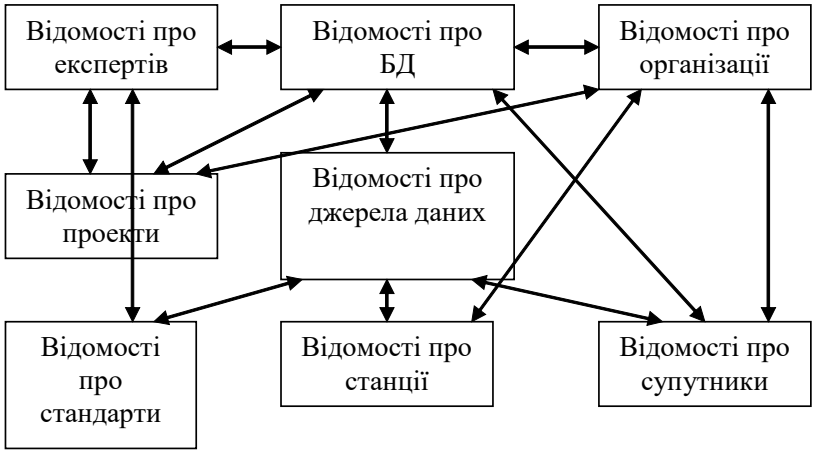


Рис. 4.3. Фрагмент мережевої БД

У *реляційній моделі* даних (Рис. 4.4) об'єкти та взаємозв'язки між ними представляються за допомогою таблиць. Взаємозв'язки також розглядаються як об'єкти (таблиці зв'язків).

Кожна таблиця являє собою один об'єкт. У термінології реляційної моделі таблиця називається відношенням.

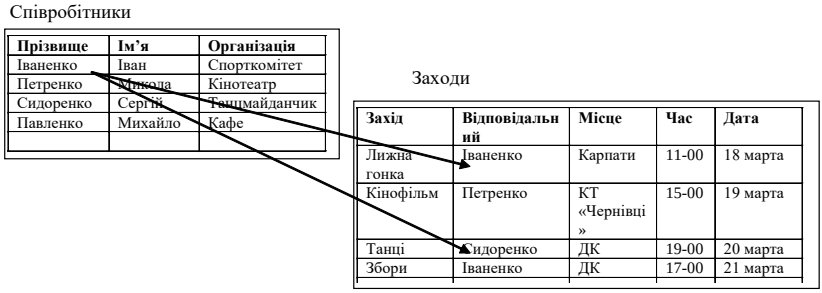
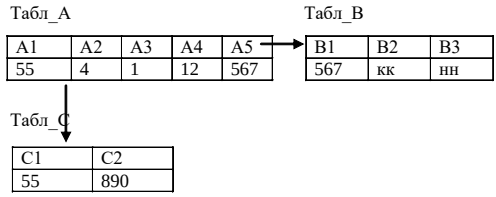


Рис.4.4. Відношення між таблицями

Постреляційна модель - це якісне й кількісне розширення реляційної моделі. Якщо в реляційних моделях використовується перша нормальна форма, то в постреляційних моделях дані описуються «не першою нормальною формою», не потрібно визначати для поля специфічний тип і довжину. Завдяки цьому, можна задати таблицю, у яку вкладені інші таблиці. Якщо всю інформацію звести в одну більшу таблицю, така таблиця неминуче буде містити порожні комірки або надлишкові дані. При використанні декількох таблиць буде потрібно виконувати досить ресурсномістку операцію з'єднання, знижуючи ефективність роботи СКБД.

Модель *багатовимірної бази даних* (Рис. 4.5) Моделлю даних є багатомірний куб, де на вимірах визначені деякі ієрархії, а в клітках цього куба перебувають числові значення. Операції отримання даних з такого куба описуються в термінах поворотів, зрізів і ієрархічного "згортання" вимірів з агрегуванням значень (підсумовування, узяття середнього й ін.). Ця схема добре лягає на табличну організацію даних. Найбільш відомий програмний продукт цього класу - Oracle Express Server.



Багатомірна таблиця

Имя вих. табл.)	Ім'я поля	Значення
Табл_А	А1	55
Табл_А	А2	4
Табл_А	А3	1
Табл_А	А4	12
Табл_А	А5	567
Табл_В	В1	567
Табл_В	В2	кк
Табл_В	В3	нн
Табл_С	С1	55
Табл_С	С2	890

Рис. 4.5. Багатомірна схема БД

4.3. Еволюція систем керування базами даних

На еволюцію СКБД істотний вплив робить бурхливий розвиток мікро- та наноелектронних технологій і пов'язаний з цим розвиток персональних комп'ютерів. Темпи розвитку персональних комп'ютерів за останніх 10-15 років істотно перевищують темпи розвитку «великих» ЕОМ. Сфера застосування персональних комп'ютерів за останні декілька років істотно розширилася. Можна виділити такі основні причини цієї тенденції:

- ціна персональних комп'ютерів значно нижча, ніж великих ЕОМ;
- по функціональних можливостях персональні комп'ютери перевершують великі ЕОМ;
- істотно зменшився розрив між продуктивністю персональних комп'ютерів і великих ЕОМ. Крім того, для багатьох завдань роботи з даними продуктивність комп'ютера не є вирішальним чинником;
- архітектура систем на основі персональних комп'ютерів володіє більшою гнучкістю й мобільністю, а сфера їх використання значно ширша від області застосування великих ЕОМ.

Загальна тенденція руху від окремих mainframe-систем до відкритих розподілених систем зробила величезний вплив на розвиток архітектури СКБД і висунула перед їхніми розробниками ряд складних проблем. Головна проблема полягала в

технологічній складності переходу від централізованого керування даними на одному комп'ютері й СКБД, яка використовувала власні моделі, формати представлення даних і мови доступу до даних, до розподіленої обробки даних у неоднорідному обчислювальному середовищі, що складається зі сполучених у мережу комп'ютерів різних моделей і виробників. Поступовий перехід від обчислювальних систем на основі великих ЕОМ і централізованого керування даними до розподілених систем на основі персональних комп'ютерів, а також упровадження персональних комп'ютерів практично в усі сфери діяльності призвели й до зміни підходів до організації СКБД. В історії розвитку й удосконалення систем керування базами даних можна умовно виділити три основні етапи. Коротко розглянемо кожен з них.

СКБД першого покоління

Перший етап пов'язаний зі створенням першого покоління СКБД, що спиралися на ієрархічну й мережеву моделі даних (на основі специфікацій CODASYL). У цей період часу на ринку обчислювальної техніки домінували великі обчислювальні машини *(mainframe)*, такі як система IBM 360/370 які в сукупності з СКБД першого покоління склали апаратно-програмну платформу великих інформаційних систем. СКБД першого покоління були в переважній більшості закритими системами: був відсутній стандарт зовнішніх інтерфейсів і не забезпечувалася мобільність прикладних програм.

Ранні СКБД, із сьогоднішнього погляду, мали масу недоліків, з яких найбільш істотні такі:

- складність використання;
- необхідність знання фізичної організації бази даних;
- сильна залежність прикладних систем від фізичної організації бази даних
- перевантаження логіки прикладних систем деталями організації доступу до бази даних;
- відсутність засобів автоматизації проектування баз даних;
- дуже висока вартість.

Серед переваг СКБД першого покоління можна відзначити:

- наявність розвинених засобів керування даними в зовнішній пам'яті на низькому рівні;
- можливість побудови ефективних прикладних систем уручну;
- можливість економії пам'яті за рахунок сумісного використання об'єктів (у мережевих системах).

Незважаючи на всі свої недоліки, СКБД першого покоління виявилися вельми довговічними: розроблене на їх основі програмне забезпечення використовується до цього дня, і великі ЕОМ, як і раніше зберігають величезні масиви актуальної інформації. Головною причиною цього є, ймовірно, економічний чинник - свого часу в апаратне й програмне забезпечення великих ЕОМ були вкладені величезні кошти: у результаті багато хто продовжує їх використовувати, не дивлячись на морально застарілу архітектуру. Водночас процес перенесення даних і програм з великих ЕОМ на комп'ютери нового покоління сам по собі – складна технічна проблема, яка вимагає значних витрат.

Реляційні СКБД

Початком другого етапу в еволюції СКБД можна вважати публікації на початку 70-х років ряду статей Е. Кодда, в яких висувалися, по суті, революційні ідеї та суттєво змінені сталі представлення про бази даних.

Математик за освітою, Кодд запропонував використовувати для обробки даних апарат теорії множин (об'єднання, перетин, різниця, декартовий добуток). Він показав, що будь-яке представлення даних зводиться до сукупності двовимірних таблиць особливого вигляду, відомого в математиці як *відношення* (по-англійськи - *relation,* звідси і назва - *реляційні* бази даних).

Одна з головних ідей Кодда полягала в тому, що зв'язок між даними повинен установлюватися відповідно до їхніх внутрішніх логічних взаємозв’язків.

У СКБД першого покоління для зв'язку записів з різних файлів використовувалися фізичні покажчики або адреси на диску. Це означало, що в тому випадку, коли в різних файлах зберігається логічно зв'язана інформація, а фізичний зв'язок між цими файлами відсутній, то для отримання вибірки (отримання

інформації) з такої бази даних необхідно використовувати низькорівневі засоби роботи з файлами. У разі ж реляційної бази даних сама СКБД підтримує отримання інформації з бази даних на основі логічних зв'язків, тому при роботі з базою даних немає необхідності безпосередньо програмувати роботу з файлами. А це істотно спрощує роботу з базами даних.

Другий важливий принцип, запропонований Коддом, полягає в тому, що в реляційних системах однією командою можуть оброблятися цілі файли даних, тоді як у ранніх СКБД однією командою оброблялася тільки один запис. Реалізація цього принципу істотно підвищила ефективність програмування баз даних.

Реалізація реляційних принципів у СКБД уможливила розробку простих мов запитів, доступних для вивчення користувачами, що не є фахівцями в області програмування. Таким чином, завдяки зниженню вимог до кваліфікації істотно розширився круг користувачів баз даних.

На початковому етапі розвитку реляційних баз даних було розроблено декілька мов запитів, серед яких найбільш відомі такі, як QBE — Query by Example (запит за зразком), Quel — Query Language (мова запитів) і SQL — Structured Query Language (структурована мова запитів). Серед цих мов на сьогоднішній день найбільше розповсюдження має SQL, яка в 1986 р. була прийнята як стандарт ANSI мов реляційних баз даних. Останнє оновлення цього стандарту було прийняте в 1992 р., і мова запитів, відповідна цьому стандарту позначається як SQL-92.

На сьогодні реляційні бази даних набули дуже великого поширення і фактично їх можна розглядати як стандарт СКБД для сучасних інформаційних систем.

Об'єктнозорієнтовані СКБД

Незважаючи на велику популярність реляційних СКБД, розвиток технології керування даними на них не зупинився. Розвиток реляційних баз даних і забезпечення можливостей вирішення складніших завдань привели до появи об'єктнозорієнтованих баз даних, для яких характерні використання ідей об'єктнозорієнтованого підходу, керування розподіленими базами даних, активного сервера бази даних, мов програмування

четвертого покоління, фрагментації та паралельної обробки запитів, технології тиражування даних, багатопотокової архітектури й інших революційних досягнень у галузі обробки даних.

Об'єктнозорієнтований підхід має ряд переваг для розробника, з яких можна відзначити такі:

- можливість розбити систему на сукупність незалежних сутностей (об’єктів) і провести їхню чітку незалежну специфікацію;
- простота еволюції системи за рахунок використання таких елементів об'єктного підходу, як спадковість і поліморфізм;
- можливість об'єктного моделювання системи, що дозволяє прослідкувати поведінку реальної сутності предметної області вже на ранніх стадіях розробки.

4.4. Реляційна модель даних

Реляційна модель даних запропонована Е. Коддом, відомим американським фахівцем у сфері баз даних. Основні концепції цієї моделі вперше опубліковані в 1970 р. у статті «A Relational Model of Data for Large Shared Data Banks» (CACM, 1970, Vol. 13, № 6). Реляційна модель дозволила розв’язати одне з найважливіших завдань в управлінні базами даних — забезпечити незалежність представлення й опису даних від прикладних програм, наслідком чого було істотне спрощення проектування та програмування баз даних. Тому після публікації праць Кодда почалися активні дослідження по створенню реляційної системи керування базами даних. У результаті цих досліджень в другій половині 70-х років створено ряд комерційних і некомерційних реляційних СКБД.

До основних переваг реляційного підходу до керування базою даних належать:

- наявність невеликого набору абстракцій, які дозволяють порівняно просто моделювати велику частину поширених предметних областей і допускають точні формальні визначення, залишаючись інтуїтивно зрозумілими;

- наявність простого і водночас могутнього математичного апарату, що спирається, в основному, на теорію множин і математичну логіку, й забезпечує теоретичний базис реляційного підходу до організації баз даних;
- можливість маніпулювання даними без необхідності знання конкретної фізичної організації баз даних у зовнішній пам'яті.

Незважаючи на всі свої переваги, реляційні системи далеко не відразу отримали широке визнання. Хоча вже в другій половині 70-х років з'явилися перші прототипи реляційних СКБД, довгий час вважалося неможливим домогтися ефективної реалізації таких систем. Проте поступове накопичення методів і алгоритмів організації реляційних баз даних і керування ними привели до того, що вже в середині 80-х років реляційні системи практично витіснили зі світового ринку ранні СКБД.

На даний час реляційні СКБД залишаються одними з найбільш поширених, незважаючи на деякі властиві їм недоліки. Тепер основним предметом критики реляційних СКБД є не їх недостатня ефективність, а деяка обмеженість таких систем при використанні в так званих нетрадиційних областях (найбільш поширеними прикладами є системи автоматизації проектування), в яких потрібні гранично складні структури даних. Причому ця обмеженість реляційних СКБД є прямим наслідком їх простоти й виявляється лише в окремих наочних областях. Другим недоліком реляційних баз даних, що часто відзначається, є неможливість адекватного віддзеркалення семантики наочної області - можливості представлення знань про семантичну специфіку предметної області в реляційних системах дуже обмежені. Визначення значеннєвого змісту зареєстрованих даних називається *семантичною інформацією* (або *семантикою*).

На усунення саме цих недоліків в основному й спрямовані дослідження зі створення об'єктнозорієнтованих баз даних.

Базові поняття реляційної моделі даних

Термін «реляційний» (від англійського *relation* — відношення) указує, перш за все, на те, що така модель зберігання даних побудована на взаємовідношенні складових її частин, які

зручно представляти у вигляді двовимірної таблиці. Кодд показав, що набір відношень (таблиць) може бути використаний для зберігання даних про об'єкти реального світу й моделювання зв'язків між ними. Тобто, реляційна модель даних представляє інформацію у вигляді сукупності взаємозв'язаних таблиць, які прийнято називати відношеннями *або* реляціями.

Основними поняттями реляційної моделі даних є:

- тип даних;
- домен;
- атрибут;
- кортеж;
- ключ.

Розглянемо сенс цих понять на прикладі відношення (таблиці) СТУДЕНТИ, що містить інформацію про студентів деякого вузу (таблиця 4.1).

Таблиця 4.1.

Приклад відношення СТУДЕНТИ реляційної бази даних

№_студентського_ квитка	**Ім'я**	**Дата_ народження**	**Курс**	**Спеціальність**
23980282	Федів Д. А.	12.03.1982	2	Біологія
22991380	Яковлєв Н. В.	25.12.1979	4	Фізика
22657879	Буць В. В.	29.02.1979	5	Математика
24356783	Афанас'єв А. В.	19.08.1983	1	Іноземна мова
24350283	Ковалюк В. І.	13.10.1982	1	Фізика
23125681	Смірнов А. Д.	26.03.1981	3	Історія

Тип даних

Поняття *тип даних* у реляційній моделі даних повністю еквівалентно відповідному поняттю в алгоритмічних мовах. Набор підтримуваних типів даних визначається СКБД і може сильно розрізнятися в різних системах. Проте практично всі СКБД підтримують такі типи даних:

- цілочислові;
- дійсні;
- строкові;
- спеціалізовані типи даних для грошових величин;

- спеціальні типи даних для часових величин (дата або час);
- типи двійкових об'єктів (даний тип не має аналога в мовах програмування; зазвичай для його позначення використовується абревіатура BLOB - Binary Large Object).

Достатньо активно розвивається підхід до розширення можливостей реляційних систем абстрактними типами даних (відповідними можливостями володіють, наприклад, системи сімейства Ingres/Postgres).

У даному прикладі використовуються три типи даних - рядковий (стовпці «Ім'я» і «Спеціальність»), часовий тип (стовпець «Дата_народження») і цілочисловий тип («Курс» і «№_студентського_квитка»).

Домен

Найменша одиниця даних реляційної моделі — це окреме *атомарне* (нерозкладне) для даної моделі значення даних. *Доменом* називається множина атомарних значень одного й того самого типу. Іншими словами, доменом є допустима потенційна множина значень даного типу. У нашому прикладі можна для кожного стовпця таблиці визначити домен:

- домени «Імена» і «Спеціальності» для стовпців «Ім'я» і «Спеціальність» відповідно базуватимуться на строковому типі даних - у число їх значень можуть входити тільки ті рядки, які можуть зображати ім'я і назву спеціальності (зокрема, такі рядки не повинні починатися з м'якого знака);
- домен «Дати_народження» для стовпця «Дата_народження» визначається на базовому часовому типі даних - даний домен містить тільки допустимий діапазон дат народження студентів;
- домени «Номери_курсів» і «Номери_студентських_квитків» базуються на цілочисловому типі - у число його значень можуть входити тільки ті цілі числа, які можуть позначати номер курсу універ-

ситету (зазвичай від 1 до 6) і номер студентського квитка (обов'язкове позитивне число).

Зазначимо також семантичне навантаження поняття домену: дані вважаються порівнянними тільки в тому випадку, коли вони відносяться до одного домену. Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, ймовірно, позбавлене сенсу. У нашому прикладі значення доменів «Номери_курсів» і «Номери_студентських_квитків» засновані на одному типі даних - цілочисловому, але не є порівнянними.

Поняття домену використовується далеко не у всіх СКБД. Як приклад реляційних баз даних, що використовують домени, можна привести Oracle і InterBase.

Атрибути, схема відношення, схема бази даних

Стовпці відношення називають *атрибутами,* їм привласнюються імена, по яких до них потім проводиться звернення.

Список імен атрибутів відношення із зазначенням імен доменів (або типів, якщо домени не підтримуються) називається *схемою відношення.* Схема нашого відношення СТУДЕНТ запишеться так:

СТУДЕНТ{№_студентського_квитка
Номери_студентських_квитків
Ім'я Імена.
Дата_народження Дати_народження.
Курс Номери_курсів.
Спеціальність Спеціальності}

Степінь відношення – це число його атрибутів. Відношення степеня один називають унарним, степеня два – бінарним, степеня три – тернарним..., а степеня *n – n-арним.*

Степінь відношення СТУДЕНТИ рівна п'яти, тобто він є 5-арним. *Схемою бази даних* називається множина іменованих схем відношень.

Кортеж

Кортеж, відповідний даній схемі відношення, є множиною пар {ім'я атрибуту, значення}, яке містить одне входження

кожного імені атрибуту, що належить схемі відношення. «Значення» є допустимим значенням домену даного атрибуту (або типу даних, якщо поняття домена не підтримується). Отже, степінь кортежу, тобто число елементів у ньому, дорівнює степеню відповідної схеми відношення. Іншими словами, кортеж - це набір іменованих значень заданого типу.

Схему відношення іноді називають також *заголовком* відношення, а відношення як набір кортежів - *тілом* відношення.

Отже, відношення, по суті, є множиною кортежів, які відповідають одній схемі відношення.

Кардинальним числом, або *потужністю відношення* називається кількість його кортежів. Потужність відношення СТУДЕНТИ дорівнює 6. На відміну від степеня відношення кардинальне число відношення (потужність відношення) змінюється в часі.

Неформальними еквівалентами основних понять є:

Відношення – Таблиця, яка на наступних етапах проектування перетворюється у файл БД, що має унікальне ім'я.

Кортеж – Рядок таблиці, що перетворюється далі в запис файлу БД.

Атрибут – Стовпчик із заголовком (ім'ям) і значеннями, який перетворюється надалі в поле БД, що має ім'я й тип.

Порожні значення

У деяких випадках який-небудь атрибут відношення може бути непридатним. Наприклад, у відношенні СТУДЕНТИ може також зберігатися інформація про потенційних абітурієнтів, відвідуючих підготовчі курси вузу. У цьому випадку непридатними виявляються атрибути «№_студентського_квитка» і «Курс» (оскільки абітурієнти ще не вступили до вузу і, отже, не мають студентського квитка і не можуть бути віднесені до якого-небудь курсу). Крім того, іноді при уведенні інформації в рядок реляційної таблиці деякі дані можуть бути невідомі й з'ясовуватися пізніше. (Для нашого прикладу — під час вступу на підготовчі курси абітурієнт ще не визначився остаточно, на яку спеціальність він вступатиме). В обох указаних випадках у поля, відповідні непридатним або невідомим атрибутам, нічого

не заноситься, і рядок записується в базу даних з порожніми значеннями цих атрибутів.

Наголосимо, що порожнє значення – це не нуль і не порожній рядок, а *невідоме* значення атрибуту, яке *не визначене* в даний момент часу і може бути визначено пізніше.

Для позначення порожніх значень полів використовується слово NULL.

Ключі відношення

Оскільки відношення з математичного погляду є множиною, а множини за визначенням не містять елементів, які збігаються то ніякі два кортежі відношення не можуть бути дублікатами один одного в будь-який довільно заданий момент часу. Тому, у відношенні завжди має бути присутнім деякий атрибут (або набір атрибутів), що однозначно визначає кожен кортеж відношення й забезпечує унікальність рядків таблиці. Такий атрибут (або набір атрибутів) називається *первинним ключем* відношення. Більш строго визначити поняття первинного ключа можна так: якщо R – відношення з атрибутами A_1, A_2..., A_n, то множина атрибутів $K = (A_i, A_j ..., A_k)$ відношення R є первинним ключем цього відношення тоді і тільки тоді, коли задовольняються дві незалежні від часу умови:

- *унікальність:* у довільний момент часу ніякі два різні кортежі відношення R не мають одного і того ж значення для A_i, A_j..., A_k;
- *мінімальність:* жоден з атрибутів A_i, A_j..., A_k не може бути виключений з K без порушення унікальності.

Для кожного відношення властивістю унікальності володіє, принаймні повний набір його атрибутів. Проте потрібно забезпечити й умову мінімальності. Тому, як правило, у відношенні завжди є один атрибут, що володіє, властивістю унікальності, й що є первинним ключем. Залежно від кількості атрибутів, що входять до ключа, розрізняють прості та складні (або складені) ключі.

Простий ключ – ключ, що містить тільки один атрибут. У загальному випадку операції об'єднання виконуються швидше в

тому випадку, коли як ключ використовується найкоротший і найпростіший з можливих типів даних. З цього погляду щонайбільше підходить цілочисловий тип, який має апаратну підтримку для виконання над ним логічних операції. *Складний,* або *складений ключ* – ключ, що складається з декількох атрибутів.

Набір атрибутів, який володіє властивістю унікальності, але не володіє мінімальністю, називається суперключем. *Суперключ* – складний (складений) ключ з великим числом стовпців необхідних для того, щоб бути унікальним ідентифікатором. Такі ключі нерідко використовуються на практиці, оскільки надмірність може виявитися корисною користувачеві.

Залежно від того, чи містить атрибут, що є первинним ключем, яку-небудь інформацію, розрізняють штучні й природні ключі. *Штучний,* або *сурогатний ключ* – ключ, створений самою СКБД, або користувачем за допомогою деякої процедури, який сам по собі не містить інформації. Штучний ключ використовується для створення унікальних ідентифікаторів рядків, коли сутність (див. Додаток 1) має бути описана повністю, щоб однозначно ідентифікувати конкретний елемент. Штучний ключ часто використовують замість значущого складного ключа, який є дуже громіздким, щоб використовуватися в реальній базі даних. Система підтримує штучний ключ, але він ніколи не показується користувачеві.

Природний ключ – ключ, у який включені значущі атрибути і який, отже, містить інформацію.

У прикладі, що розглядається нами, як первинний ключ відношення СТУДЕНТИ можна розглядати атрибут №_студентського_квитка. Причому даний ключ буде природним, оскільки він несе цілком певну інформацію.

Кожен з типів первинних ключів має свої переваги і недоліки; їх обговоренню присвячена велика кількість публікацій. Відзначимо лише основні плюси й мінуси кожного з видів ключів.

Основною перевагою природних ключів є те, що вони несуть цілком певну інформацію і їх використання не приводить до необхідності додавати в таблиці атрибути, значення яких не мають ніякого сенсу і використовуються лише для зв'язку між відношеннями. Іншими словами, використання природних ключів дозволяє отримати компактнішу форму таблиць (у яких не

буде надмірних, неінформативних даних) і більш природний зв'язок між ними.

Основним же недоліком природних ключів є те, що їх використання утруднене в разі мінливості предметної області. Треба розуміти, що значення атрибутів первинного ключа не повинні змінюватися. Тобто одного разу задане значення первинного ключа для кортежу не може бути пізніше змінене. Така вимога висувається в основному для підтримки цілісності бази даних. Зв'язок між сутностями зазвичай установлюється саме по первинному ключу, і його зміна приведе до порушення цих зв'язків або до необхідності зміни записів в декількох таблицях. Навіть у порівняно простих базах даних це може викликати ряд проблем, які важко розв’язуються.

У деяких реляційних СКБД допускається зміна первинного ключа. Іноді це буває справді корисно. Проте вдаватися до цього треба лише в разі крайньої необхідності.

Типовим прикладом мінливої предметної області, в якій для сутності неможливо визначити незмінний природний ключ, є будь-яка область, де як сутність виступає людина. Справді, неможливо визначити для людини набір атрибутів, які були б унікальні і незмінні впродовж усього її життя.

Другий, досить істотний недолік природних ключів полягає в тому, що, як правило, унікальні природні ключі є складеними і містять строкові атрибути. Як уже наголошувалося вище, максимальна швидкість виконання операцій над даними забезпечується при використанні простих цілочислових ключів. Отже, з погляду швидкодії системи природні ключі часто виявляються неоптимальними.

Обидва недоліки природних ключів можна подолати, визначивши у відношеннях сурогатні ключі, що є деяким універсальним атрибутом, як правило, цілочислового типу, який не залежить ні від предметної області, ні, тим більше, від структури відношення, яке він ідентифікує. Отже, можна забезпечити унікальність і незмінність ключа. Проте за це доводиться платити надмірністю даних у таблицях.

Зауважимо, що в багатьох практичних реалізаціях реляційних СКБД допускається порушення властивості унікальності кортежів для проміжних відношень, що породжуються

неявно при виконанні запитів. Такі відношення є не множинами, а мультимножинами, що в ряді випадків дозволяє домогтися певних переваг, але іноді приводить до серйозних проблем.

У будь-якій з таблиць може опинитися декілька наборів атрибутів, які можна вибрати в якості ключа. Такі набори називаються *потенційними* або *альтернативними* ключами.

Нерідко у відношеннях визначаються так звані *вторинні ключі.* Вторинний ключ є комбінацією атрибутів, відмітною від комбінації, що складає первинний ключ. Причому вторинні ключі не обов'язково володіють властивістю унікальності. При їхньому визначенні можуть задаватися такі обмеження:

- UNIQUE – обмеження унікальності, значення вторинних ключів при даному обмеженні не можуть дублюватися;
- NOT NULL – при даному обмеженні жоден з атрибутів, що входять до складу вторинного ключа, не може набувати значення NULL.

4.4.1. Пов'язані відношення

У реляційній моделі дані представляються у вигляді сукупності *взаємозв'язаних* таблиць. Подібне взаємовідношення між таблицями називається *зв'язком (relationship).* Отже, ще одним важливим поняттям реляційної моделі є зв'язок між відношеннями.

Для розгляду пов'язаних відношень скористаємося розглянутим раніше прикладом – відношенням СТУДЕНТИ. Дане відношення може бути пов'язане з відношенням УСПІШНІСТЬ, у якому містяться дані про успішність студентів з різних предметів. Фрагмент такого відношення може мати вигляд, наведений у таблиці 4.2

Таблиця 4.2.

Фрагмент відношення УСПІШНІСТЬ, пов'язаного з відношенням СТУДЕНТИ

№_студентського_квитка	**Предмет**	**Оцінка**
…	…	…
23980282	Вища математика	4
23980282	Філософія	5
22991380	Вища математика	3
22991380	Філософія	NULL
22657879	Загальна фізика	5
24356783	Загальна фізика	NULL
…	…	…

Атрибут "№_студентського_квитка" таблиці УСПІШНІСТЬ містить ідентифікатор студента (у даному прикладі в якості такого ідентифікатора використовується номер студентського квитка). Якщо потрібно дізнатися ім'я студента, що відповідає рядкам у таблиці УСПІШНІСТЬ, то треба пошукати це ж значення ідентифікатора студента в полі «№_студентського_квитка» таблиці СТУДЕНТИ і в знайденому рядку прочитати значення поля «Ім'я». Тобто, зв'язок між таблицями СТУДЕНТИ і УСПІШНІСТЬ установлюється за атрибутом «№_студентського_квитка».

При розгляді пов'язаних таблиць важливе значення має поняття *зовнішнього ключа.* Розглянемо його детальніше.

Зовнішні ключі відношення

У базах даних одні й ті самі імена атрибутів часто використовуються в різних відношеннях. У даному прикладі атрибут «№_студентського_квитка» присутній як у відношенні СТУДЕНТИ, так і у відношенні УСПІШНІСТЬ. У даному прикладі атрибут «№_студентського_квитка» ілюструє поняття *зовнішнього ключа (foreign key).*

Зовнішній ключ – це атрибут (або множина атрибутів) одного відношення, що є ключем іншого (або того ж самого) відношення.

Зовнішні ключі використовуються для встановлення логічних зв'язків між відношеннями. Зв'язок між двома таблицями встановлюється шляхом привласнення значень зовнішнього ключа однієї таблиці значенням ключа іншої.

Як і будь-які інші ключі, зовнішні ключі можуть бути простими або складеними.

Часто зв'язок між відношеннями встановлюється за первинним ключем, тобто значенням зовнішнього ключа одного відношення привласнюються значення первинного ключа іншого відношення. Проте це не є обов'язковим – у загальному випадку зв'язок може встановлюватися також і за допомогою вторинних ключів. Крім того, при встановленні зв'язків між таблицями необов'язкова вимога унікальності ключа, по якому встановлюється зв'язок.

Атрибути зовнішнього ключа не обов'язково повинні мати ті самі імена, що й атрибути ключа, яким вони відповідають. Наприклад, у нашому прикладі можна було дати атрибуту «№_студентського_квитка» таблиці УСПІШНІСТЬ інше ім'я, наприклад, «Студентський_квиток».

Зовнішній ключ може посилатися й на ту саму таблицю, до якої він належить. У цьому випадку зовнішній ключ називається *рекурсивним.*

Умови цілісності даних

Щоб інформація, яка зберігається в базі даних, була однозначною і несуперечливою, в реляційній моделі встановлюються деякі *обмежуючі умови.* Обмежуючі умови – це правила, що визначають можливі значення даних. Вони забезпечують логічну основу для підтримки коректних значень даних у базі. Обмеження цілісності дозволяють звести до мінімуму помилки, які виникають при оновленні й обробці даних.

Найважливішими обмеженнями цілісності даних є:

- категорійна цілісність;
- посилальна цілісність.

Обмеження категорійної цілісності полягає у наступному. Кортежі відношення представляють у базі даних елементи певних об'єктів реального світу або, відповідно до термінології реляційних СКБД, *категорії.* Наприклад, рядок таблиці СТУДЕНТИ представляє конкретного студента. Первинний ключ таблиці однозначно визначає кожен кортеж і, отже, кожен елемент категорії. Тобто, для отримання даних, що містяться в рядку таблиці або для маніпулювання цими даними, необхідно знати значення ключа для цього рядка. Тому рядок не може бути занесений у базу даних до тих пір, поки не будуть визначені всі атрибути його первинного ключа. Це правило називається правилом *категорійної цілісності* і коротко формулюється таким чином: жоден атрибут первинного ключа рядка не може бути порожнім. Друга умова накладає на зовнішні ключі обмеження для забезпечення цілісності даних, що називаються *посилальною цілісністю.*

Якщо дві таблиці пов’язано між собою, то зовнішній ключ таблиці повинен містити тільки ті значення, які вже є серед значень ключа, по якому здійснюється зв'язок. Якщо коректність значень зовнішніх ключів не контролюється СКБД, то може, порушиться посилальна цілісність даних. Це можна пояснити на даному прикладі таким чином. Якщо видалити з таблиці СТУДЕНТИ рядок (наприклад, при відрахуванні студента), що має хоч би один пов'язаний з ним рядок в таблиці УСПІШНІСТЬ, то це приведе до того, що в таблиці УСПІШНІСТЬ залишаться записи про успішність студента, який вже відрахований. Така сама ситуація буде спостерігається і в тому випадку, якщо зовнішньому ключеві таблиці УСПІШНІСТЬ помилково буде привласнена значення, відсутнє в значеннях ключа пов'язаної таблиці. Обмеження категорійної й посилальній цілісності повинні підтримуватися СКБД. Для дотримання цілісності сутності досить гарантувати відсутність будь-якому відношенні кортежів з одним і тим же значенням первинного ключа. Що ж до посилальної цілісності, то тут забезпечення цілісності виглядає дещо складніше.

При оновленні відношення, що посилається (при вставці нових кортежів або модифікації значення зовнішнього ключа в існуючих кортежах), досить стежити за тим, щоб не з'являлися

некоректні значення зовнішнього ключа. А ось при вилученні кортежу з відношення, на яке веде посилання можна використовувати один з трьох підходів, кожен з яких підтримує цілісність по посиланнях:

- перший підхід полягає в тому, що забороняється проводити видалення кортежу, на який існують посилання (тобто спочатку потрібно або видалити кортежі що посилаються, або відповідним чином змінити значення їх зовнішнього ключа);
- при другому підході при видаленні кортежу, на який є посилання, у всіх кортежах, що посилаються, значення зовнішнього ключа автоматично стає невизначеним;
- третій підхід (званий також *каскадним видаленням)* полягає в тому, що при видаленні кортежу з відношення, на яке веде посилання, з відношення, що посилається, автоматично видаляються всі кортежі, що посилаються.

У розвинених реляційних СКБД зазвичай можна вибрати спосіб підтримки посилальної цілісності для кожної окремої ситуації визначення зовнішнього ключа. Звичайно, для ухвалення такого рішення необхідно аналізувати вимоги конкретної прикладної області.

Типи зв'язків між таблицями

При встановленні зв'язку між двома таблицями одна з них буде *головною* (master), а друга - *підлеглою* (detail). Відмітність між ними спрощено можна пояснити таким чином. У головній таблиці завжди доступні всі записи, що містяться в ній. У підлеглій же таблиці доступні тільки ті записи, у яких значення атрибутів зовнішнього ключа збігається зі значенням відповідних атрибутів поточного *запису головної таблиці*. Причому зміна поточного запису головної таблиці приведе до зміни множини доступних записів підлеглої таблиці, а зміна поточному запису в підлеглій таблиці не викличе ніяких змін у жодній з таблиць.

На практиці часто пов'язують більше двох таблиць. Одна й та сама таблиця може бути головною по відношенню до однієї таблиці і підлеглою по відношенню до іншої. Або в однієї

головної таблиці може знаходитися в підпорядкуванні не одна, а декілька таблиць. Проте підлегла таблиця не може керуватися двома таблицями. Тому у головної таблиці може бути декілька підлеглих, але в підлеглої таблиці може бути лише одна головна.

Розрізняють чотири типи зв'язків між таблицями реляційної бази даних:

- *один до одного(1:1)* – кожному запису однієї таблиці відповідає тільки одна запис іншої таблиці;
- *один до багатьох (1:N)* – одному запису головної таблиці можуть відповідати декілька записів підлеглої таблиці;
- *багато до одного (N:1)* – декільком записам головної таблиці може відповідати один і той самий запис підлеглої таблиці;
- *багато до багатьох (N:M)* - один запис головної таблиці пов'язаний з декількома записами підлеглої таблиці, а один запис підлеглої таблиці пов'язаний з декількома записами головної таблиці.

Відмітність між типами зв'язків «один до багатьох» і «багато до одного» залежить від того, яка з таблиць вибирається як головна, а яка – як підлегла. Наприклад, якщо з пов'язаних таблиць СТУДЕНТИ і УСПІШНІСТЬ в якості головної вибрати таблицю СТУДЕНТИ, то одержуємо тип зв'язку «один до багатьох». Якщо ж у якості головної таблиці вибирається УСПІШНІСТЬ, отримаємо тип зв'язку «багато до одного».

4.4.2. Основні властивості відношень

Розглянемо тепер деякі найважливіші властивості відношень реляційної моделі даних.

Відсутність упорядкованості кортежів

У таблицях реляційної бази даних інформація зберігається в неупорядкованому вигляді. Упорядковування в принципі не підтримується СКБД, і таке поняття, як порядковий номер кортежу, не має ніякого сенсу. Властивість відсутності впорядкованості кортежів відношення також є наслідком визначення відношення як множини кортежів. Відсутність вимоги до

підтримки порядку на множини кортежів відношення дає СКБД додаткову гнучкість при зберіганні баз даних у зовнішній пам'яті й при виконанні запитів до бази даних.

Відсутність упорядкованості атрибутів

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

Атомарність значень атрибутів

Значення всіх атрибутів атомарні. Це випливає з визначення домену як потенційної множини значень простого типу даних, тобто серед значень домену не може міститися множина значень (відношення).

4.5. Реляційна система керування базами даних

Реляційна база даних — це сукупність відношень, що містять усю інформацію, яка повинна зберігатися в базі даних. Проте користувачі можуть сприймати таку базу даних як сукупність таблиць. Таким чином, реляційну базу даних можна розглядати як сховище даних, що містить набір двовимірних пов'язаних таблиць. Набор засобів для керування подібним сховищем називається *реляційною системою керування базами даних.* Реляційна СКБД може містити утиліти, додатки, служби, бібліотеки, засоби створення додатків й інші компоненти.

Ще раз підкреслимо, що в реляційній базі даних таблиці пов'язані між собою; це дозволяє за допомогою єдиного запиту знайти все необхідні дані (які можуть знаходитися в декількох таблицях). Будучи зв'язаною за допомогою загальних ключових

полів, інформація в реляційній базі даних може об'єднуватися з множини таблиць в єдиний результуючий набір.

Властивості таблиць реляційної бази даних

Оскільки таблиці в реляційній СКБД є відношеннями реляційної моделі даних, то й властивості цих таблиць є властивостями відношень, які ми вже розглянули вище.

Коротко сформулюємо ці властивості ще раз:

- кожна таблиця складається з однотипних рядків і має унікальне ім'я;
- рядки мають фіксовану кількість полів (стовпців) і значень (багатозначні поля й групи, що повторюються, недопустимі). Інакше кажучи, у кожній позиції таблиці на перетині рядка і стовпця завжди є в точності одне значення або NULL;
- рядки таблиці обов'язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок;
- стовпцям таблиці привласнюються унікальні імена, і в кожному з них розміщуються однорідні значення даних (дати, прізвища, цілі числа або грошові суми);
- повний інформаційний зміст бази даних представляється у вигляді явних значень даних, і такий метод представлення є єдиним. Зокрема, не існує яких-небудь спеціальних «зв'язків» або покажчиків, що сполучають одну таблицю з іншою;
- при виконанні операцій з таблицею її рядки і стовпці можна обробляти у будь-якому порядку незалежно від їх інформаційного змісту. Цьому сприяє наявність імен таблиць і їх стовпців, а також можливість виділення будь-якого рядка або будь-якого набору рядків з указаними ознаками.

Індекси

Вище ми розглянули поняття ключів таблиць бази даних. У більшості реляційних СКБД ключі реалізуються за допомогою об'єктів, що називаються *індексами.* Індексом є покажчик на дані, розміщені в реляційній таблиці. Можна провести аналогію індексу таблиці бази даних з покажчиком, що зазвичай по-

міщається в кінці книги. Щоб знайти у книзі сторінки, що відносяться до деякої теми, найпростіше звернутися до покажчика, в якому встановлюється відповідність між перерахованими в алфавітному порядку темами і номерами сторінок, і відразу визначити сторінки, які слід проглянути. Щоб без покажчика знайти всі сторінки, що відносяться до потрібної теми, довелося б переглядати всю книгу. Індекс бази даних призначений для аналогічних цілей – щоб прискорити пошук інформації в таблиці бази даних. Індекс надає інформацію про точне фізичне розташування даних у таблиці.

Відзначалося, що записи в реляційних таблицях невпорядковані. Проте будь-який запис у конкретний момент часу має цілком певне фізичне місцеположення у файлі бази даних, хоча воно і може змінюватися при зміні інформації, що зберігається в базі даних.

При створенні індексу в ньому зберігається інформація про місцезнаходження записів, що відносяться до індексованого стовпця таблиці. При додаванні в таблицю нових записів або видаленні існуючих індекс також модифікується. При виконанні запиту до бази даних, в умову пошуку якого входить індексований стовпець, пошук значень проводиться насамперед в індексі. Якщо цей пошук виявляється успішним, то в індексі встановлюється точне місцеположення шуканих даних в таблиці бази даних.

Розглянемо приклад індексу. На рис. 4.6 показаний фрагмент таблиці СТУДЕНТИ й індексу, побудованого по полю «Ім'я» даної таблиці. При виконанні пошуку по імені студента, проглядаючи індекс, можна відразу визначити порядковий номер запису, що містить необхідну інформацію, і потім швидко знайти в таблиці самі дані. Якби у таблиці був відсутній індекс по полю «Ім'я», то виконання пошуку по імені студента потребувало б проглядання всієї таблиці. Отже, використання індексів знижує час вибірки даних.

Розрізняють декілька типів індексів. Найчастіше виділяють три типи:

- прості;
- складені;
- унікальні.

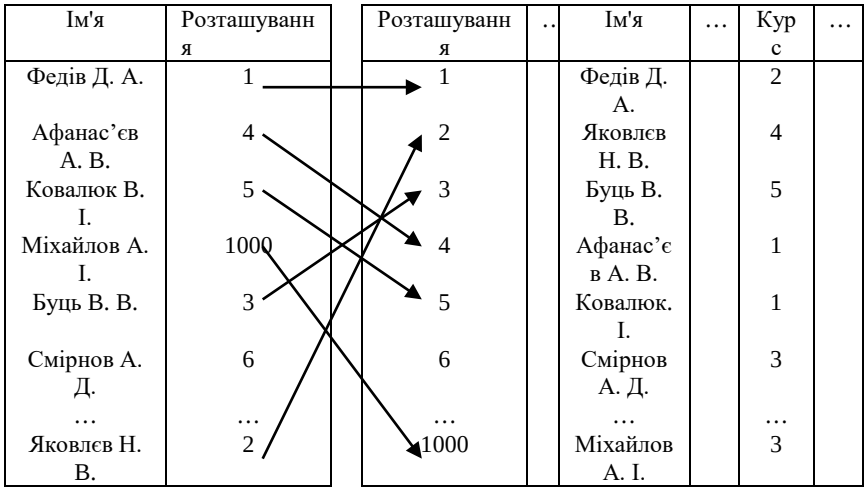


Рис. 4.6. Пошук інформації в таблиці за допомогою індексу

Прискорення пошуку інформації при використанні індексу може здаватися неочевидним – адже кількість записів у індексі збігається з кількістю записів у таблиці.

Проте слід ураховувати дві обставини:

- звернення до індексу виконується швидше, ніж до таблиці;
- в індексі записи зберігаються у впорядкованому вигляді (наприклад, в алфавітному порядку) і, тому під час пошуку інформації в індексі немає необхідності проглядати всі дані до кінця індексу.

Прості індекси є простим і разом із тим найбільш поширеним типом індексів. Простий індекс будується на основі тільки одного стовпця реляційної таблиці (індекс, приведений на Рис. 4.6, є простим).

Складені індекси будуються по двох і більш стовпцях реляційної таблиці. При створенні складеного індексу необхідно брати до уваги, що послідовність стовпців, по яких створюється індекс, впливає на швидкість пошуку даних.

Послідовність стовпців у складеному індексі вказується при його створенні й ніяк не пов'язана з послідовністю стовпців у таблиці.

Можна назвати дві умови оптимальності слідування стовпців у складеному індексі:

- першим треба розміщати стовпець, що містить найбільш обмежуюче значення (тобто що містить меншу кількість повторів);
- першим слід поміщати стовпець, що містить дані, які найбільш часто задаються в умовах пошуку.

Сформульовані умови оптимальності часто є суперечливими, так, що між ними слід знаходити розумний компроміс.

Потрібно серйозно ставитися до планування індексів. Неправильне застосування індексів може привести до зниження продуктивності системи. Ми вже говорили про те, що фізичне місцеположення записів може змінюватися у процесі редагування даних користувачами, а також у результаті маніпуляцій з файлами бази даних, що проводяться самою СКБД (таких як стиснення даних, збирання «сміття» та ін.). Зазвичай при цьому відбуваються відповідні зміни й в індексі, а це збільшує час, потрібний СКБД для проведення таких операцій. Тому зазвичай не варто індексувати:

- стовпці, дані в яких схильні до частої зміни;
- стовпці, що містять велику кількість порожніх значень;
- стовпці, що містять невелику кількість унікальних значень;
- невеликі таблиці;
- поля великого розміру.

Унікальні індекси не допускають уведення в таблицю дублюючих значень. Унікальні індекси використовуються не тільки з метою підвищення швидкості пошуку, але й для підтримки цілісності даних. Унікальний індекс може бути як простим, так і складеним.

4.6. Нормалізація даних

Нормалізація – це процес реорганізації даних шляхом ліквідації груп, що повторюються, й інших суперечностей з метою приведення таблиць до вигляду, який дозволяє здійснювати несуперечливе і коректне редагування даних.

Остаточна мета нормалізації зводиться до отримання такого проекту бази даних, в якому *кожен факт з'являється лише в одному місці,* тобто, виключена надмірність інформації. Таким чином, нормалізацію можна також визначити як процес, направлений на зменшення надмірності інформації в реляційній базі даних.

Цілі нормалізації

Надмірність інформації усувається не стільки з метою економії пам'яті, скільки для виключення можливої суперечливості даних, що зберігаються, і спрощення керування ними.

Використання ненормалізованих таблиць може привести до порушення цілісності даних (суперечливості інформації) у базі даних. Зазвичай розрізняють наступні проблеми, що виникають при використанні ненормалізованих таблиць:

- надмірність даних;
- аномалії оновлення;
- аномалії вилучення;
- аномалії уведення.

Щоб проілюструвати проблеми, що виникають при роботі з ненормалізованими базами даних, розглянемо як приклад таблицю СПІВРОБІТНИКИ, що містить інформацію про співробітників певної організації. Структура цієї таблиці наведена на Рис. 4.7.

Код співробітника
Ім'я
Прізвище
По батькові
Дата народження
Адреса
Телефон
Посада
Розряд
Зарплатня
Рейтинг
Дата прийому
Дата звільнення

Рис. 4.7. Структура ненормалізованої таблиці СПІВРОБІТНИКИ

Надмірність даних

Надмірність даних виявляється в тому, що в декількох записах таблиці бази даних повторюється одна й та сама інформація. Наприклад, одна людина може працювати на двох (або навіть більше) посадах. Але в таблиці, наведеній на рис. 4.7, кожній посаді відповідає запис, і в цьому записі міститься інформація про особисті дані співробітника, що цю посаду того займає. Тобто, якщо співробітник працює на декількох посадах, то його особисті дані дублюватимуться кілька разів, що призведе до невиправданого збільшення зайнятого обсягу зовнішньої пам'яті.

Аномалії оновлення

Аномалії оновлення тісно пов'язані з надмірністю даних. Припустимо, що в співробітника, який працює на декількох посадах, змінилася адреса. Щоб інформація, що міститься в таблиці, була коректною, необхідно буде внести зміни в декілька записів. Якщо ж виправлення буде внесено не до всіх записів, то виникне невідповідність інформації, яка називається *аномалією оновлення*.

Аномалії вилучення

Аномалії вилучення виникають при вилученні записів з ненормалізованої таблиці. Нехай, наприклад, в організації проводиться скорочення штатів і деякі посади анулюються. При цьому слід видалити відповідні записи в даній таблиці. Проте видалення приведе до втрати інформації про співробітника, що обіймав цю посаду. Така втрата інформації й називається аномалією вилучення. (Для нашого випадку можна привести й інший приклад - видалення записів при звільненні співробітника приведе до втрати інформації про посаду, яку він займав.)

Аномалії уведення

Аномалії уведення виникають при додаванні в таблицю нових записів і зазвичай виникають, коли для деяких полів таблиці задані обмеження NOT NULL. У таблиці, що розглядається як приклад, є поле «Рейтинг», в якому міститься інформація про рівень кваліфікації співробітника, що встанов-

люється за результатами його роботи. При прийомі на роботу нового співробітника встановити рівень його кваліфікації неможливо, оскільки він ще не виконував жодних робіт в організації. Якщо для цього поля задати обмеження NOT NULL, то в таблицю не можна буде увести інформацію про нового співробітника. Це й називається аномалією уведення.

Висновки

Очевидно, що аномалії оновлення, вилучення та уведення вкрай небажані. Для того, щоб звести до мінімуму можливість появи такого роду аномалій і використовується нормалізація.

4.6.1. Нормальні форми

Теорія нормалізації заснована на концепції *нормальних форм.* Кожній нормальній формі відповідає деякий певний набір обмежень, і відношення знаходиться в деякій нормальній формі, якщо воно відповідає властивому даній формі набору обмежень.

У теорії реляційних баз даних зазвичай виділяється така послідовність нормальних форм:

- перша (1NF);
- друга (2NF);
- третя (3NF);
- нормальна форма Бойса-Кодда (BCNF);
- четверта (4NF);
- п'ята, або нормальна форма проекції-з'єднання (5NF або PJ/NF).

Основні властивості нормальних форм:

- кожна наступна нормальна форма в деякому розумінні краща від попередньої;
- при переході до наступної нормальної форми властивості попередніх нормальних властивостей зберігаються.

В основі процесу проектування лежить метод нормалізації – декомпозиція відношення, що знаходиться в попередній нормальній формі, у два або більше відношення, що задовольняють вимогам наступної нормальної форми. Найбільш важливі

на практиці нормальні форми відношень ґрунтуються на фундаментальному в теорії реляційних баз даних понятті *функціональної залежності.* Функціонально залежним вважається такий атрибут, значення якого однозначно визначається значенням іншого атрибуту. Функціонально залежні атрибути позначаються так: $X \to Y$. Цей запис означає, що якщо два кортежі в таблиці мають одне й те саме значення атрибуту X, то вони мають одне й те саме значення атрибуту Y. Атрибут, що вказується в лівій частині, називається *детермінантом.*

Первинний ключ таблиці є детермінантом, оскільки його значення однозначно визначає значення будь-якого атрибуту таблиці.

Перша нормальна форма

Обмеження першої нормальної форми – значення всіх атрибутів відношення мають бути атомарними. Дана вимога є базовою вимогою класичної реляційної моделі даних, тому будь-яка реляційна таблиця (у тому числі й таблиця, структура якої зображена на рис. 4.7) за визначенням уже знаходиться в першій нормальній формі.

Друга нормальна форма

Відношення знаходиться в другій нормальній формі в тому і лише у тому випадку, коли це відношення знаходиться в першій нормальній формі, і кожен неключовий атрибут повністю залежить від первинного ключа.

Неключовим називається будь-який атрибут відношення, що не входить до складу первинного ключа.

Щоб перейти від першої нормальної форми до другої, потрібно виконати такі кроки:

1. визначити, на які частини можна розбити первинний ключ, так щоб деякі з неключових полів залежали від однієї з цих частин (причому ці частини можуть містити декілька атрибутів);
2. створити нову таблицю для кожної такої частини ключа й групи залежних від неї полів перемістити в цю таблицю. Частина колишнього первинного ключа стане при цьому первинним ключем нової таблиці.

3. вилучити з початкової таблиці поля, переміщені в інші таблиці, окрім тих, які стануть зовнішніми ключами.

У нашому прикладі для приведення таблиці СПІВРОБІТНИКИ до другої нормальної форми її треба розділити на дві таблиці. Первинний ключ початкової таблиці складається з двох атрибутів — «Код співробітника» і «Посада». Усі особисті дані про співробітників залежать тільки від атрибуту «Код співробітника». Атрибути, що відповідають цим даним, ми й виділимо в якості однієї таблиці, яку назвемо ФІЗИЧНІ ОСОБИ. Інформацію ж про посади та їх оплату винесемо в іншу таблицю, якій присвоїмо ім'я СПІВРОБІТНИКИ.

Схема приведення таблиці до другої нормальної форми подана на рис. 4.8.

Отримані дві таблиці пов'язані між собою за полем «Код фізичної особи», яке є первинним ключем для таблиці ФІЗИЧНІ ОСОБИ і зовнішнім ключем для таблиці СПІВРОБІТНИКИ. Дане поле було відсутнє в початковій таблиці й було додане при проведенні нормалізації.

Рис. 4.8. Приведення таблиці до другої нормальної форми

Третя нормальна форма

Розглянемо таблицю СПІВРОБІТНИКИ, отриману після приведення початкової таблиці до другої нормальної форми. Для цієї таблиці існує функціональний зв'язок між полями «Код співробітника» і «Зарплатня». Проте цей функціональний зв'язок є *транзитивним*.

Функціональна залежність атрибутів X і Y відношення R називається транзитивною, якщо існує такий атрибут Z, що присутніми є функціональні залежності $X \to Z$ і $Z \to Y$, але відсутня функціональна залежність $Z \to X$.

Транзитивність залежності полів «Код співробітника» і «Зарплатня» означає, що заробітна плата насправді є характеристикою не співробітника, а посади, яку він займає. У результаті ми не зможемо занести в базу даних інформацію, яка характеризує заробітну плату на посаді, до тих пір, поки не з'явиться хоча б один співробітник, що займає цю посаду (оскільки первинний ключ не може містити невизначене значення). При видаленні кортежу, що описує останнього співробітника, що обіймає дану посаду, ми позбудемося інформації про заробітну плату, що відповідає цій посаді. Крім того, щоб узгодженим чином змінити заробітну плату, що відповідає посаді, буде необхідно заздалегідь знайти всі записи, що описують співробітників, що обіймають дану посаду. Отже, в таблиці СПІВРОБІТНИКИ, як і раніше, існують аномалії. Їх можна усунути шляхом подальшої нормалізації – приведення бази даних до третьої нормальної форми.

Відношення R знаходиться в третій нормальній формі в тому і лише в тому випадку, якщо воно знаходиться в другій нормальній формі і кожен неключовий атрибут нетранзитивно залежить від первинного ключа.

Щоб перейти від другої нормальної форми до третьої, потрібно виконати такі кроки:

1. визначити всі поля (або групи полів), від яких залежать інші поля;
2. створити нову таблицю для кожного такого поля (або групи полів) і групи залежних від нього полів перемістити в цю таблицю. Поле (або група полів), від якого

залежить решта всіх переміщених полів, стане при цьому первинним ключем нової таблиці;

3. Вилучити переміщені поля з початкової таблиці, залишивши лише ті з них, які стануть зовнішніми ключами.

Зверніть увагу, що ми знову додали новий атрибут – «Код посади», який є первинним ключем для відношення ПОСАДИ й зовнішнім ключем для відношення СПІВРОБІТНИКИ. Додавання нових атрибутів при нормалізації дозволяє отримати таблиці з простими первинними ключами, що полегшує виконання операції об'єднання таблиць. Такі первинні ключі, як правило, штучні.

Приведемо базу даних, що розглядається як приклад, до третьої нормальної форми. Для цього розділимо таблицю СПІВРОБІТНИКИ на дві – СПІВРОБІТНИКИ й ПОСАДИ (Рис. 4.9).

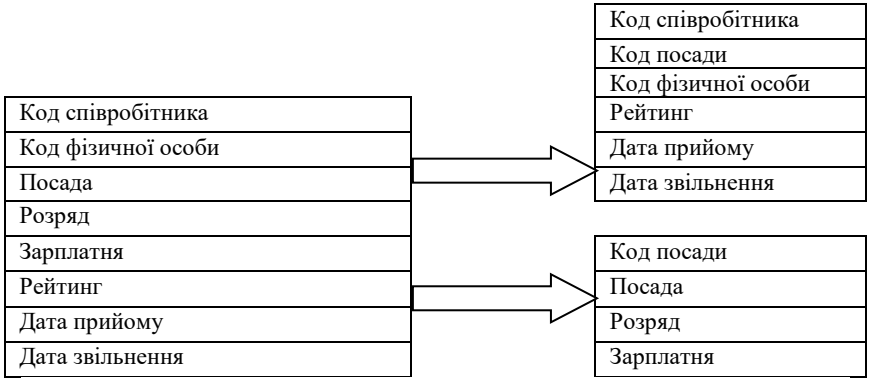


Рис. 4.9. Приведення бази даних до третьої нормальної форми

На практиці третя нормальна форма схем відношень у більшості випадків є достатньою, і приведенням до третьої нормальної форми процес проектування реляційної бази даних зазвичай закінчується. Тому ми не будемо розглядати інші нормальні форми, тим більше, що в роботі вони використовуються порівняно рідко.

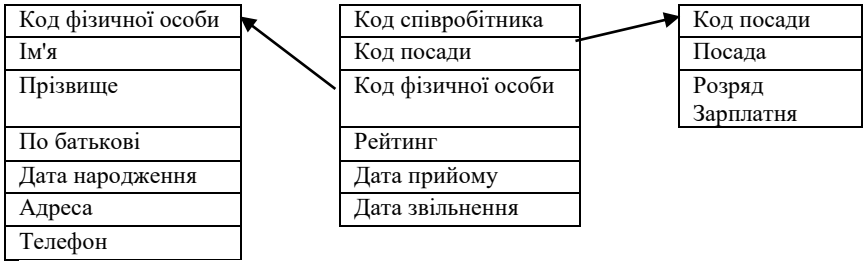


Рис. 4.10. Структура бази даних, приведеної до третьої нормальної форми

На закінчення приведемо схему бази даних, приведеної до третьої нормальної форми, що розглядається як приклад (Рис. 4.10).

Розділ 5. Керування реляційними базами даних

Використання реляційних баз даних можливе тільки за наявності ефективних засобів керування ними. Тому після публікацій статей Кодда, де пропонувалася реляційна модель даних, стали активно проводитися дослідження зі створення мов керування реляційними даними. У результаті цих дослідів було запропоновано ряд мов, серед яких варто виділити три:

- SQL — Structured Query Language (структурована мова запитів);
- QBE — Query By Example (запит за зразком);
- QUEL — Query Language (мова запитів).

Тепер найбільшого поширення набула мова SQL, яка є єдиною мовою реляційних баз даних, прийнятим як стандарт ANSI.

Зауваження: Хоча SQL і називається мовою запитів, вона містить, окрім засобів запитів, і всі необхідні засоби з керування базами даних. У даному розділі ми розглянемо можливості мови SQL по керуванню об'єктами реляційної бази даних і адмініструванню.

5.1. Коротка історія мови SQL

Мова реляційних баз даних SQL розроблена в середині 70-х років у рамках дослідницького проекту експериментальної реляційної СКБД System R компанії IBM. Даний проект включав розробку реляційної системи керування базами даних і мови SEQUEL (Structured English Query Language). Початкова назва SEQUEL тільки частково відображала суть цієї мови. Незважаючи на те, що мова була зорієнтована в основному на зручне й зрозуміле користувачам формулювання запитів до реляційної бази даних, вона вже була повноцінною мовою реляційної бази даних, що містить, окрім операторів формулювання запитів і маніпулювання базою даних, такі елементи:

- засоби визначення схеми бази даних і маніпулювання нею;
- засоби визначення обмежень цілісності й тригерів;
- засоби створення представлень бази даних;
- можливості визначення структур фізичного рівня, що підтримують ефективне виконання запитів;

- засоби авторизації доступу до відношень і їх полів;
- засоби підтримки точок збереження транзакції і відкатів.

У кінці 70-х років модифікований варіант мови SEQUEL, що отримав назву SQL, був випущений корпорацією Oracle як мова комерційної системи керування базами даних. У 1983 р. компанія IBM випустила SQL як мову керування СКБД DB2. Американський національний інститут стандартів (ANSI) прийняв мову SQL як стандарт у 1986 р. З тих пір цей стандарт переглядався двічі — в 1989р. були внесені деякі незначні зміни, а в 1992 р. стандарт SQL був досить істотно розширений і в даний час відомий йод назвою ANSI SQL-92 або SQL/92.

Зауваження: Майте на увазі, що ANSI SQL - усього лише стандарт, що не є реальною мовою. Кожен виробник систем керування базами даних, як правило, пропонує власну реалізацію мови SQL. Причому в таких реалізаціях можуть бути як розширення існуючого стандарту, так і відхилення від нього, зокрема, відсутність деяких стандартних елементів мови. Проте незалежно від реалізації, основа SQL зберігається, тому при вивченні мови SQL головним є розуміння базових концепцій і команд ANSI SQL-92.

5.2. Типи команд SQL

Команди мови SQL зазвичай поділяються на декілька груп. Основні типи команд такі:

- DDL (Data Definition Language) - мова визначення даних. Команди даної групи використовуються для створення і зміни структури об'єктів бази даних (наприклад, для створення і видалення таблиць);
- DML (Data Manipulation Language) - мова маніпулювання даними. Команди DML використовуються для маніпулювання інформацією, що міститься в об'єктах бази даних;
- DCL (Data Control Language) - мова керування даними. Відповідні команди призначені для керування доступом до інформації, що зберігається в базі даних;
- DQL (Data Query Language) - мова. Це найчастіше використовувані команди, призначені для формування

запитів до бази даних (*запит* - це звернення до бази даних для отримання відповідної інформації);

- команди адміністрування бази даних призначені для здійснення контролю за виконуваними діями й аналізом здійснюваних операцій;
- команди керування транзакціями.

5.3. Типи даних SQL/92

Типи даних, використовувані в стандартному SQL, можна розділити на такі групи:

- рядкові типи;
- числові типи;
- типи для представлення дати і часу.

Розглянемо ці типи даних детальніше.

5.3.1. Рядкові типи

В SQL/92 визначені два рядкові типи:

- символьні рядки фіксованої довжини;
- символьні рядки змінної довжини.

Символьні рядки фіксованої довжини

Дані, що зберігаються у вигляді символьних рядків фіксованої довжини, завжди займають один і той самий обсяг пам'яті, що визначається під час оголошення поля, незалежно від реального розміру рядка, занесеного в поле. Оголошення рядка фіксованої довжини, згідно з ANSI SQL-92 має вигляд:

CHARACTER(n) або CHAR(n)

де *n* - довжина рядка, що визначає розмір поля, до якого це оголошення відноситься. При використанні рядків фіксованої довжини порожні місця зазвичай заповнюються пропусками. Наприклад, якщо розмір поля заданий рівним 10, а в нього введений рядок, що складається з 3 символів, то решта 7 символів заповнюються пропусками.

Зауваження: Не слід використовувати тип CHARACTER для полів, призначених для зберігання довгих рядків, довжина яких може сильно варіюватися, - це приведе до невиправданої витрати доступної зовнішньої пам'яті (дискового простору).

Символьні рядки змінної довжини

Довжина рядків змінної довжини не є постійною для всіх даних, а залежить від реального розміру рядка, що зберігається в полі таблиці бази даних. Оголошення рядка змінної довжини має вигляд:

VARCHAR(n)

де *n* - число, що визначає максимально можливу довжину рядка. На відміну від типу CHARACTER використання VARCHAR забезпечує економніше витрачання дискового простору. Незалежно від того, який розмір рядка вказаний в оголошенні, поле займатиме стільки місця, скільки необхідно для зберігання занесеної в нього інформації. Наприклад, якщо оголошено поле VARCHAR(10) і в нього занесений рядок завдовжки 3 символи, то для зберігання цього рядка буде використано тільки три байти, а не 10, як у випадку рядка фіксованої довжини.

5.3.2. Числові типи

Числові типи поділяються на:

- цілочислові типи;
- дійсні типи з фіксованою крапкою;
- дійсні типи з плаваючою крапкою;
- двійкові рядки фіксованої і змінної довжини.

Цілочислові типи

Стандартом ANSI SQL-92 встановлюються два цілочислові типи:

- **INTEGER** або **INT** - ціле число зі знаком, що використовує 4 байти. Може представляти числа в діапазоні від -2 147 483 648 до 2 147 483 647;

- **SMALLINT** - коротке ціле число зі знаком, що використовує 2 байти. Може представлять цілі числа в діапазоні від -32 768 до 32 767.

Дійсні типи з фіксованою крапкою

Дійсні типи з фіксованою крапкою призначені для точного представлення дробових чисел. Найчастіше ці типи використовуються в тому випадку, коли недопустимими є похибки, неминучі при представленні важливих чисел з плаваючою комою в двійковій формі (наприклад, при зберіганні значень грошових величин). Дійсні типи з фіксованою комою, по суті, є цілочисловими типами, в яких відображається десяткова крапка.

Синтаксис оголошення типу з фіксованою комою наступний:

DECIMAL(n,m)

де *n* - точність; m - масштаб. Точність - це загальна довжина числового значення. Масштаб - кількість знаків, розташованих праворуч від десяткової крапки.

Дійсні типи з плаваючою крапкою

Типи з плаваючою крапкою зазвичай використовуються в наукових і інженерних розрахунках. При використанні цих типів треба враховувати, що в процесі занесення в базу даних деякого числа при його перетворенні у двійкову форму з плаваючою крапкою завжди вноситься деяка похибка. І хоча ця похибка дуже мала, в деяких випадках вона є неприпустимою і може внести серйозну помилку, наприклад, при підсумовуванні великої кількості значень. Тому типи з плаваючою крапкою непридатні для зберігання значень грошових величин.

Найчастіше використовуються два дійсні типи з плаваючою крапкою:

- **FLOAT** - числа з одинарною точністю;
- **DOUBLE** - числа з подвійною точністю.

Двійкові рядки

Двійкові рядки використовуються порівняно рідко. Зазвичай поля такого типу застосовуються як прапори або двійкові маски. Так само, як і символьні рядки, двійкові рядки бувають фіксованими і змінної довжини. Двійкові рядки фіксованої довжини оголошуються так:

BIT(n)

де *n* - довжина рядка в байтах. Оголошення рядків змінної довжини виглядає так:

BIT VARYING(n)

де *n* - максимальна довжина рядка в байтах.

Типи для представлення дати й часу

Очевидно, що дані типи використовуються для зберігання інформації, що стосується дат і часу.

Зауваження: Іноді типи даних, призначені для зберігання часу й дати, називаються темпоральними.

У стандарті SQL визначені наступні типи даних для зберігання інформації про дату й час:

- **DATE** - використовується для зберігання дати;
- **TIME** - використовується для зберігання часу;
- **TIMESTAMP** - зберігає дату й час;
- **INTERVAL** - зберігає проміжок часу між двома датами або між двома моментами часу.

Зауваження: Зазначимо, що в більшості реалізацій SQL підтримуються деякі додаткові типи даних. Крім того, синтаксичні і семантичні властивості типів, визначених у стандарті ANSI SQL-92, можуть розрізнятися в окремих реалізаціях SQL. Необхідно завжди мати на увазі, що, хоча з використанням мови SQL можна визначити схему бази даних, яка містить дані будь-якого з розглянутих типів, можливість використання цих даних у прикладних системах залежить від застосованої мови програмування.

5.4. Керування об'єктами бази даних

Об'єкт бази даних - це будь-який об'єкт, визначений у базі даних для зберігання інформації або для звернення до інформації. Прикладами об'єктів бази даних можуть бути таблиці, представлення і індекси. Для керування об'єктами бази даних використовується підмножина команд DDL мови SQL.

5.4.1. Створення, модифікація і видалення таблиць

Таблиця є основним об'єктом для зберігання інформації в реляційній базі даних. При створенні таблиці обов'язково вказуються імена полів, що містяться в таблиці, і типи даних, відповідні полям. Крім того, при створенні таблиці для полів можуть указуватися обмежувальні умови й значення, що задаються за замовчанням.

Обмежувальні умови - це правила, що обмежують значення величин у полі таблиці бази даних. Значення за замовчанням - значення, яке автоматично уводиться в поле таблиці бази даних при додаванні нового запису, якщо користувач не вказав значення цього поля.

Під час опису команд передбачається, що:

- текст, набраний великими літерами (наприклад, **CREATE TABLE**) є обов'язковим;
- текст, набраний малими літерами (наприклад, **ім'я_таблиці**) позначає змінну, що уводить користувач;
- у квадратних дужках (наприклад, **[NOT NULL]**) міститься необов'язкова частина команди;
- взаємовиключні елементи команди розділяються вертикальною рискою (наприклад, **[UNIQUE | PRIMARY KEY]**).

Оператор CREATE DATABASE/DROP DATABASE

Для створення (вилучення) бази даних використовуються наступні оператори, що мають наступний синтаксис:

CREATE DATABASE ім'я_бази_даних
DROP DATABASE ім'я_бази_даних

Оператор CREATE TABLE

Для створення таблиці використовується оператор CREATE TABLE. Синтаксис цього оператора має вигляд:

```
CREATE TABLE ім'я_таблиці (
ім'я_поля_1 тип_даних ,
ім'я_поля_2 тип_даних ,
.....
ім'я_поля_N тип_даних)
```

Для прикладу розглянемо оператор, що створює таблицю ФІЗИЧНІ ОСОБИ, розглянуту в попередньому розділі:

```
CREATE TABLE Фізичні_особи (
Код_фізичної_особи INTEGER,
Ім'я VARCHAR(25),
Прізвище VARCHAR(25),
По батькові VARCHAR(25),
Дата_народження DATE,
Адреса VARCHARC50),
Телефон VARCHAR(25))
```

Зауваження: У прикладі, що наводиться, щоб уникнути плутанини й неоднозначності, ми використовували українські імена таблиці й полів. При створенні ж реальних таблиць реальної бази даних треба мати на увазі, що далеко не всі СКБД допускають використання символів кирилиці в іменах полів і таблиць. Більш того, навіть якщо СКБД дозволяє використовувати українські (російські) літери, бажано все ж таки використовувати в іменах об'єктів бази даних тільки латинські символи, особливо, якщо для створення інтерфейсної частини інформаційної системи передбачається використовувати засоби розробки "третіх" фірм. Це дозволить уникнути цілого ряду важкорозв'язуваних проблем.

Оператор ALTER TABLE

Створена таблиця може бути модифікована з використанням оператора ALTER TABLE. За допомогою цього оператора

можна додавати й видаляти поля таблиці, змінювати тип даних полів, додавати й вилучати обмеження.

Зауваження: Оператор ALTER TABLE не визначений в стандарті ANSI. Проте він підтримується в більшості реалізацій SQL, забезпечуючи істотно велику гнучкість керування структурою бази даних. Якщо ж використовувана СКБД не підтримує ALTER TABLE, то можна просто створити нову таблицю зі зміненою структурою і потім перенести в неї дані із старої таблиці, після чого стару таблицю можна буде видалити.

У загальному вигляді синтаксис оператора ALTER TABLE виглядає так:

> ***ALTER TABLE*** ім'я_таблиці [MODIFY] [ім'я_поля тип_даних] [ADD] [ім'я_поля тип_даних] [DROP] [ім'я_поля]

Дія, що виконується оператором ALTER TABLE, визначається ключовим словом, яке вказується після імені таблиці:

- **MODIFY** - змінює визначення поля;
- **ADD** - додає нове поле в таблицю;
- **DROP** - видаляє поле з таблиці.

Для зміни типу даних поля використовується такий синтаксис оператора ALTER TABLE:

> ***ALTER TABLE*** ім'я_таблиці ADD (ім'я_поля тип_даних)

Наприклад, щоб додати в таблицю ФІЗИЧНІ ОСОБИ поле, в якому міститиметься адреса електронної пошти співробітника, треба використовувати такий оператор:

> ***ALTER TABLE*** Фізичні_особи ADD (Email CHARACTER(25))

Якщо ж потрібно змінити тип даних існуючого поля, то треба використовувати оператор ALTER TABLE в парі з ключовим словом MODIFY:

ALTER TABLE ім'я_таблиці MODIFY (ім'я_поля тип_даних)

Нехай, наприклад, після того, як ми додали в таблицю ФІЗИЧНІ ОСОБИ поле Email, з'ясувалося, що використання типу CHARACTER для цього поля неефективне - у багатьох співробітників немає електронної пошти і, отже, частина дискового простору витрачається даремно. Доцільніше застосувати для цього поля тип даних VARCHAR. Для зміни типу даних викличемо оператор ALTER TABLE:

ALTER TABLE Фізичні_особи MODIFY (Email VARCHAR(25))

Вилучення існуючого поля виконується викликом оператора ALTER TABLE з ключовим словом DROP:

ALTER TABLE ім'я_таблиці DROP (ім'я_поля)

Зауваження: Треба бути дуже обережним при використанні оператора ALTER TABLE. Непродумане внесення змін до таблиць уже працюючої бази даних може призвести до порушення роботи всієї системи в цілому.

Оператор DROP TABLE

Для вилучення таблиць використовується оператор DROP TABLE. Синтаксис цього оператора має такий вигляд:

DROP TABLE ім'я_таблиці [RESTRICT | CASCADE]

Якщо при виклику оператора DROP TABLE використовується ключове слово RESTRICT і на таблицю, що вилучається, посилається яке-небудь представлення або обмеження, то при виконанні оператора вилучення таблиці буде згенероване повідомлення про помилку. Якщо ж використовувати ключове слово CASCADE, то вилучення таблиці буде виконано й разом з таблицею будуть вилучені всі представлення й обмеження, що посилаються на неї.

5.4.2. Задавання обмежень

Обмеження використовуються для того, щоб забезпечити достовірність і несуперечність інформації в базі даних. Існує достатньо велика кількість різного роду обмежень, з яких ми розглянемо лише основні:

- NOT NULL;
- первинного ключа;
- UNIQUE;
- зовнішнього ключа;
- CHECK.

Обмеження NOT NULL

Обмеження NOT NULL може бути встановлене для будь-якого поля реляційної таблиці. За наявності цього обмеження забороняється уведення значень NULL в полі, для якого це обмеження встановлене.

Зауваження: **З**начення NULL не еквівалентне до нульового значення для числових полів ні пропуску для текстових полів - якщо в поле занесене значення «0» (або « »), то поле не порожнє, а містить число 0 (або рядок, що складається з одного пропуску). Якщо ж значення поля дорівнює NULL, то це означає, що поле містить невизначене значення (поле порожнє), тобто до нього не було занесено жодної інформації.

Обмеження NOT NULL встановлюється при створенні таблиці за допомогою оператора CREATE TABLE. Щоб задати обмеження NOT NULL для деякого поля, треба просто вказати NOT NULL після вказівки типу поля:

```
CREATE TABLE ім'я_таблиці (
ім’я_поля_1 тип_даних NOT NULL,
ім’я_поля_2 тип_даних NULL,
.....
ім’я_поля_N тип_даних NOT NULL)
```

Якщо ж після задавання типу даних поля іде слово NULL, то дане поле може містити порожні значення. Проте атрибут

NULL зазвичай установлюється за замовчанням, тому вказувати його явно немає необхідності.

Обмеження NOT NULL установлюється для тих полів, до яких при занесенні даних до таблиці обов'язково повинна бути введена яка-небудь інформація. Наприклад, у таблиці, що містить особисті дані про співробітників організації, можна задати обмеження NOT NULL для полів, в яких міститимуться імена й прізвище співробітників. Тому оператор створення таблиці ФІЗИЧНІ ОСОБИ треба видозмінити так:

```
CREATE TABLE Фізичні_особи (
Код_фізичної_особи INTEGER,
Ім'я VARCHAR(25) NOT NULL,
Прізвище VARCHAR(25) NOT NULL,
По батькові VARCHAR(25),
Дата_народження DATE,
Адреса VARCHAR(50),
Телефон VARCHAR(25))
```

Зауваження: При додаванні нового поля в не порожню таблицю з використанням оператора ALTER TABLE не можна встановлювати обмеження NOT NULL для поля, що додається. Це цілком очевидно - уже існуючі записи в таблиці не можуть мати в новому стовпці непорожні значення. Проте це обмеження можна подолати так:

1. Додайте в таблицю поле без обмеження NOT NULL.
2. Заповніть значення нового поля для всіх існуючих записів.
3. Змініть визначення нового поля за допомогою команди ALTER TABLE, задавши йому обмеження NOT NULL.

Обмеження первинного ключа

Первинні ключі вказуються при створенні таблиці. Оскільки поля, що входять до складу первинного ключа, не можуть набувати значення NULL, то для них обов’язкове обмеження NOT NULL.

Обмеження первинного ключа може бути задане двома шляхами.

- У тому випадку, коли первинний ключ складається тільки з одного поля, то він може бути заданий за допомогою ключових слів PRIMARY KEY, що вказуються при описі поля в операторі CREATE TABLE:

```
CREATE TABLE ім'я_таблиці (
ім'я_поля_1 тип_даних NOT NULL PRIMARY KEY,
ім'я_поля_2 тип_даних,
.....
ім'я_поля_N тип_даних NOT NULL)
```

Зверніть увагу на те, що вказівка обмеження NOT NULL для поля, що є первинним ключем, є обов'язковим.

- Первинний ключ може бути також заданий у кінці опису таблиці, після визначень усіх полів. Для цього також використовується ключова фраза PRIMARY KEY, після якої в круглих дужках указується ім'я поля, що становить первинний ключ:

```
CREATE TABLE ім'я_таблиці (
ім'я_поля_1 тип_даних NOT NULL,
ім'я_поля_2 тип_даних,
.....
ім'я_поля_N тип_даних NOT NULL,
PRIMARY KEY (ім'я_поля_1))
```

Другий спосіб особливо зручний для задавання складених первинних ключів. У цьому випадку в дужках треба указати через кому всі поля, складові первинного ключа:

```
CREATE TABLE ім'я_таблиці (
ім'я_поля_1 тип_даних NOT NULL,
ім'я_поля_2 тип_даних,
ім'я_поля_3 тип_даних NOT NULL,
.....
ім'я_поля_N тип_даних NOT NULL,
PRIMARY KEY (ім'я_поля_1, ім'я_поля_3))
```

Зауваження: При використанні складеного первинного ключа обмеження NOT NULL повинне бути задане для всіх полів, що входять до його складу.

Обмеження UNIQUE

Обмеження UNIQUE схоже на обмеження первинного ключа, оскільки при наявності даного обмеження для деякого поля всі значення, що містяться в цьому полі, повинні бути унікальними. Проте на відміну від первинного ключа, обмеження UNIQUE допускає наявність порожніх значень поля (якщо, звичайно, для цього поля не встановлено обмеження NOT NULL).

Обмеження UNIQUE задається при створенні таблиці за допомогою ключового слова UNIQUE, що вказується при описі поля:

```
CREATE TABLE ім'я_таблиці (
ім'я поля 1 тип даних NOT NULL PRIMARY KEY,
ім'я_поля_2 тип_даних UNIQUE,
ім'я_поля_3 тип_даних NOT NULL,
.....
ім'я_поля_N тип_даних NOT NULL UNIQUE)
```

Можна також задати обмеження UNIQUE не для одного поля, а для групи полів. Оголошення групи полів унікальною відрізняється від оголошення унікальними індивідуальних полів, оскільки саме комбінація значень, а не просто індивідуальні значення, зобов'язана бути унікальною. Тобто значення кожного поля, що входить до групи, не обов'язково повинне бути унікальним, а комбінація значень полів завжди повинна бути унікальною.

Обмеження UNIQUE для групи полів, так само як і складений первинний ключ, задається після опису всіх полів таблиці:

```
CREATE TABLE ім'я_таблиці (
ім'я_поля_1 тип_даних NOT NULL PRIMARY KEY,
ім'я_поля_2 тип_даних,
ім'я_поля_3 тип_даних NOT NULL,
.....
ім'я_поля_N тип_даних NOT NULL UNIQUE,
UNIQUE (ім'я_поля_2, ім'я_поля_3))
```

Обмеження зовнішнього ключа

Обмеження зовнішнього ключа є основним механізмом для підтримки посилальної цілісності бази даних. Поле, що визначається як зовнішній ключ, використовується для посилання на поле іншої таблиці, що зазвичай називається батьківським ключем, а таблиця, на яку зовнішній ключ посилається, називається батьківською таблицею (батьківський ключ часто є первинним ключем батьківської таблиці).

Типи полів зовнішнього й батьківського ключа обов'язково повинні бути ідентичні. А ось імена полів можуть бути різними. Проте щоб уникнути плутанини бажано й імена полів для зовнішнього й батьківського ключів задавати однаковими.

Зовнішній ключ не обов'язково повинен складатися тільки з одного поля. Подібно до первинного ключа, зовнішній ключ може складатися з будь-якої кількості полів, які обробляються як єдиний об'єкт. Поля батьківського ключа, на який посилається складений зовнішній ключ, повинні йти в тому самому порядку, що й у зовнішньому ключі.

Коли поле таблиці є зовнішнім ключем, воно у певний спосіб пов’язано з таблицею, на яку цей ключ посилається. Це фактично означає, що кожне значення зовнішнього ключа безпосередньо прив'язане до значення в батьківському ключі. Як ілюстрація використання обмеження зовнішнього ключа візьмемо приклад з попереднього розділу - базу даних по обліку співробітників деякої організації (Рис. 5.1). Дана база даних складається з трьох таблиць:

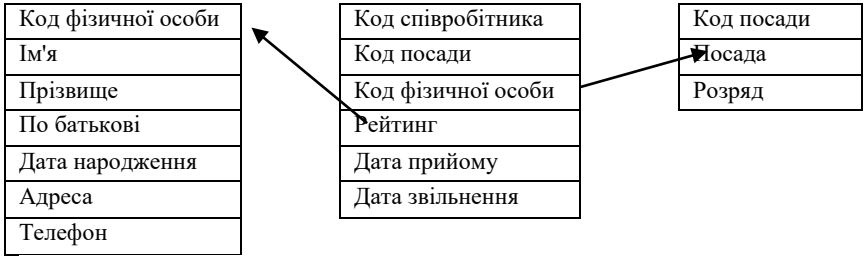


Рис. 5.1. Структура бази даних співробітників організації

- СПІВРОБІТНИКИ - містить інформацію про професійні дані співробітників;
- ФІЗИЧНІ ОСОБИ - містить інформацію про особисті дані співробітників;
- ПОСАДИ - містить інформацію про посади організації.

Основною таблицею в цій базі даних є таблиця СПІВРОБІТНИКИ, яка посилається на дві інші таблиці і, відповідно, повинна мати два зовнішніх ключа. В якості батьківських ключів у таблицях ФІЗИЧНІ ОСОБИ й ПОСАДИ використовуються первинні ключі.

Обмеження зовнішнього ключа (FOREIGN KEY) може бути задане або в операторі CREATE TABLE, або за допомогою оператора ALTER TABLE. Синтаксис обмеження FOREIGN KEY має вигляд:

FOREIGN KEY ім'я_зовнішнього_ключа (список полів зовнішнього ключа)
REFERENCES ім'я_батьківської_таблиці (список полів батьківського ключа)

Перший список полів - це список з одного або декількох полів таблиці, що розділені комами. Другий список полів - це список полів, які будуть складати батьківський ключ. Списки полів, указані як зовнішній і батьківський ключі, повинні бути сумісні:

- вони повинні мати однакову кількість полів;
- порядок слідування полів у списках повинен співпадати. Причому збіг визначається не іменами полів, які можуть бути різні, а типами даних і розміром полів.

Розглянемо приклад створення бази даних із пов'язаними таблицями:

```
CREATE TABLE Фізичні_особи (
Код_фізичної_особи INTEGER NOT NULL PRIMARY KEY,
Ім'я VARCHAR(25) NOT NULL,
Прізвище VARCHAR(25) NOT NULL,
По батькові VARCHAR(25),
Дата_народження DATE,
Адреса VARCHAR(50),
Телефон VARCHAR(25))
```

CREATE TABLE Посади (
Код_посади INTEGER NOT NULL PRIMARY KEY,
Посада VARCHAR(50) NOT NULL UNIQUE,
Розряд INTEGER NOT NULL,
Зарплатня DECIMAL (7,2) NOT NULL)
CREATE TABLE Співробітники (
Код_співробітника INTEGER NOT NULL PRIMARY KEY,
Код_посади INTEGER.
Код_фізичної_особи INTEGER NOT NULL,
Рейтинг DECIMAL(4,2),
Дата_прийому DATE NOT NULL,
Дата_звільнення DATE,
FOREIGN KEY Фіз_ВК (Код_фізичної_особи)
REFERENCES Фізичні_особи (Код_фізичної_особи),
FOREIGN KEY Посад_ВК (Код_посади)
REFERENCES Посади (Код_посади))

Зовнішній ключ може бути доданий і після створення таблиці - за допомогою оператора ALTER TABLE (природно, тільки в тому випадку, якщо дана реалізація SQL підтримує даний оператор).

Синтаксис оператора ALTER TABLE, що використовується для створення зовнішнього ключа, має вигляд:

ALTER TABLE ім’я_таблиці
ADD CONSTRAINT ім’я_зовнішнього_ключа FOREIGN KEY (список полів зовнішнього ключа)
REFERENCE ім'я_батьківської_таблиці (список полів батьківського ключа)

Зауваження: Слід мати на увазі, що при використанні оператора ALTER TABLE для створення зв'язку між таблицями необхідно, щоб пов’язувані таблиці знаходилися в стані посилальної цілісності. Інакше при спробі виконання оператора буде видано повідомлення про помилку.

Зовнішній ключ обмежує значення, які можна увести в таблицю. Для того щоб у поля, що складають зовнішній ключ, можна було увести деяке значення, необхідно, щоб це значення

вже було уведене в батьківській таблиці. Наприклад, щоб занести до таблиці СПІВРОБІТНИКИ нового співробітника, необхідно, щоб у таблиці ФІЗИЧНІ ОСОБИ вже існував запис про його особисті дані - інакше неможливо буде заповнити обов'язкове поле «Код_фізичної_особи».

Для зовнішнього ключа може бути задане обмеження NOT NULL, але це необов'язково, а в деяких випадках навіть небажано. Наприклад, припустимо, що в організацію приймається на роботу новий співробітник, але ще не визначена однозначно посада, яку він займе. У цьому випадку можна занести все необхідні дані про нього в таблицю ФІЗИЧНІ ОСОБИ й до таблиці СПІВРОБІТНИКИ, нічого не вказуючи в полі «Код_посади», яке буде заповнено пізніше.

Обмеження зовнішнього ключа також робить вплив на вилучення й модифікацію записів батьківської таблиці. Жодне значення батьківського ключа, на яке посилається який-небудь зовнішній ключ, не може бути видалене або змінено. Це означає, наприклад, що не можна вилучити з таблиці ФІЗИЧНІ ОСОБИ запис про співробітника, якщо вона пов'язана із записом у таблиці СПІВРОБІТНИКИ. Це цілком зрозуміло - якщо в таблиці СПІВРОБІТНИКИ присутній запис про співробітника фірми, а з таблиці ФІЗИЧНІ ОСОБИ запис про цього співробітника видалений, то інформація про його особисті дані буде втрачена. Якщо ж співробітник звільнився й запис про нього з таблиці СПІВРОБІТНИКИ вилучений, то немає необхідності зберігати інформацію про його особисті дані, і відповідний запис з таблиці ФІЗИЧНІ ОСОБИ також може бути вилучений.

Аналогічно не можна змінювати значення батьківського ключа, на який посилається який-небудь зовнішній ключ, - це також приведе до втрати інформації й порушенню посилальної цілісності бази даних.

Зауваження: У деяких реалізаціях SQL є можливість задавати для зовнішніх ключів каскадне вилучення й каскадне оновлення. Це означає, що при спробі видалити або модифікувати значення батьківського ключа, на яке посилається зовнішній ключ, відповідні записи зовнішнього ключа також будуть вилучені (каскадне видалення) або змінені (каскадне оновлення). Дані можливості відсутні в стандарті ANSI SQL-92.

Один із синтаксичних варіантів задавання каскадного оновлення й вилучення такий:

UPDATE ***OF*** ім'я_батьківської_таблиці CASCADES
DELETE ***OF*** ім'я_батьківської_таблиці CASCADES

Ключові фрази UPDATE OF і DELETE OF указуються в операторі CREATE TABLE. Замість ключового слова CASCADES можна вказати слово RESTRICTED - у цьому випадку зміна й видалення значень батьківського ключа, на які посилається зовнішній ключ з даної таблиці, буде заборонене.

CREATE TABLE Співробітники (
Код_співробітника INTEGER NOT NULL PRIMARY KEY,
Код_посади INTEGER,
Код_фізичної_особи INTEGER NOT NULL,
Рейтинг DECIMAL (4,2).
Дата_прийому DATE NOT NULL,
Дата_звільнення DATE,
FOREIGN KEY Фіз_ВК (Код_фізичної_особи)
REFERENCES Фізичні_особи (Код_фізичної_особи),
FOREIGN KEY Посад_ВК (Код_посади)
REFERENCES Посади (Код_посади),
UPDATE OF Фізичні_особи CASCADES
DELETE OF Фізичні_особи RESTRICTED)

Обмеження CHECK

Обмеження CHECK використовується для перевірки допустимості даних, що уводяться в поле таблиці.

Обмеження CHECK складається з ключового слова CHECK, супроводжуваного реченням предикату, який використовує вказане поле. Будь-яка спроба модифікувати або вставити значення поля, яке могло б зробити цей предикат хибним, буде відхилена.

Зауваження: Перевірка коректності значень, що заносяться в базу даних, може також виконуватися в призначених для користувача додатках. Проте використання обмеження CHECK забезпечує додатковий рівень захисту від помилок.

Задавання обмеження CHECK проводиться при створенні таблиці. Для цього після опису полів таблиці вказується ключова фраза:

CONSTRAINT ім'я_обмеження ***CHECK*** (обмеження)

У прикладі бази даних співробітників організації обмеження, що розглядається нами, може бути задано, наприклад, для поля «Розряд» таблиці ПОСАДИ. Припустимо, що розряд не може перевищувати 20. Тоді оператор створення таблиці ПОСАДИ, в якому задано це обмеження, матиме вигляд:

```
CREATE TABLE Посади (
Код_посади INTEGER NOT NULL PRIMARY KEY,
Посада VARCHAR(50) NOT NULL UNIQUE,
Розряд INTEGER NOT NULL,
Зарплатня DECIMAL(7,2) NOT NULL,
CONSTRAINT CHK_RATE CHECK (Розряд<=20))
```

Можна задавати обмеження й для декількох полів. Для цього потрібно просто внести їх в обмежувальну умову. Для формування складного обмеження, що включає декілька умов, використовуються логічні оператори AND і OR.

У таблиці ПОСАДИ можна, наприклад, увести ще обмеження на мінімальну зарплатню:

```
CONSTRAINT CH_RATE CHECK (Розряд<=20 AND Зарплатня>=1000))
```

5.4.3. Задавання значень за замовчанням

Для полів таблиці можна задавати значення за замовчуванням, які будуть заноситися в поля при додаванні нового запису в таблицю, якщо значення цих полів невизначені.

Зауваження: Значення NULL фактично є значенням за замовчуванням, прийнятим для кожного поля таблиці, для якого не задано обмеження NOT NULL і яке не має іншого значення за замовчуванням.

Для задавання значення за замовчанням використовується директива DEFAULT, яка вказується в команді CREATE TABLE при описі поля, для якого встановлюється значення за замовчуванням:

```
CREATE TABLE (
.....
ім'я_поля_N тип_даних DEFAULT = значення_за_замовчуванням
.....
)
```

У даному прикладі значення за замовчуванням може бути, наприклад, установлено для поля «Рейтинг» таблиці СПІВРОБІТНИКИ:

```
CREATE TABLE Співробітники (
Код_співробітника INTEGER NOT NULL PRIMARY KEY, Код_посади INTEGER,
Код_фізичної_особи INTEGER NOT NULL,
Рейтинг DECIMAL(4,2) DEFAULT=0,
Дата_прийому DATE NOT NULL,
.....)
```

5.4.4. Створення й вилучення індексів

Стандарт ANSI на даний час не підтримує індекси. Проте індекси широко застосовуються практично в усіх базах даних, тому роботу з ними не можна обійти увагою. Синтаксис оператора створення індексу може істотно різнитися залежно від використовуваної реалізації SQL. Найчастіше зустрічається така синтаксична форма команди створення індексу:

```
CREATE INDEX ім'я_індексу
ON ім'я_таблиці (ім'я_поля_1, [ім'я_поля_2, ...])
```

Зауваження: Наведена форма оператора CREATE INDEX може бути доповнена низкою численних параметрів, які сильно різняться в різних реалізаціях SQL. Ці параметри використовуються, наприклад, для впорядковування інформації про зростання або убування (параметри ASC і DESC).

Створення простого індексу

Простий індекс є простим і разом з тим поширеним різновидом індексів. Простий індекс складається тільки з одного поля (стовпця) таблиці, тому він часто також називається одностовпчиковим індексом. Найбільш типовий синтаксис команди створення простого індексу має вигляд:

```
CREATE INDEX ім'я_індексу
ON ім'я_таблиці (ім'я_стовпчика)
```

Наприклад, для таблиці ФІЗИЧНІ ОСОБИ можна було б створити індекс по полю, що містить прізвища співробітників, за допомогою такого оператора:

```
CREATE INDEX NAME_IDX
ON Фізичні_особи (Прізвище)
```

Унікальні індекси

Унікальний індекс не допускає введення до таблиці значень, що дублюються. Отже, унікальні індекси використовуються не тільки з метою підвищення продуктивності, але й для підтримки цілісності даних.

Типовий синтаксис оператора створення унікального індексу має вигляд:

```
CREATE UNIQUE INDEX ім'я_індексу
ON ім'я_таблиці (ім'я_поля)
```

Наприклад, для таблиці ПОСАДИ можна створити унікальний індекс по полю «Посада» за допомогою такої команди:

```
CREATE UNIQUE INDEX POST_IDX
ON Посади (Посада)
```

Зауваження: Створити унікальний індекс для існуючої таблиці можна тільки в тому випадку, якщо в індексованому полі не містяться значення, що повторюються.

Складені індекси

Складеними називаються індекси, побудовані по двох і більш полях. При створенні складеного індексу необхідно враховувати, що порядок слідування полів у складеному індексі робить істотний вплив на швидкість пошуку даних. У загальному випадку поля в індексі належить розташовувати в порядку зменшення обмежувальних значень. Синтаксис задавання складеного індексу має наступний вигляд:

```
CREATE INDEX ім'я_індексу
ON ім'я_таблиці (ім'я_поля_1, ім'я_поля_2, ...)
```

У нашому прикладі має сенс створити складений індекс для полів «Прізвище» і «Ім'я» таблиці ФІЗИЧНІ ОСОБИ. Оператор створення такого індексу має вигляд:

```
CREATE INDEX FULLNAME_IDX
ON Фізичні_особи (Прізвище, Ім'я)
```

Зауваження: Зверніть увагу на те, що порядок розташування полів в останньому прикладі повинен бути саме таким, оскільки поле «Прізвище» накладає сильніше обмеження, ніж поле «Ім'я» - вірогідність того, що в декількох співробітників будуть однакові імена, вище, ніж вірогідність збігу прізвищ.

Видалення індексів

Видалення індексів не викликає жодних проблем. Для вилучення необхідно знати тільки ім'я індексу (і, зрозуміло, володіти відповідними правами). Синтаксис оператора вилучення індексу має вигляд:

```
DROP INDEX ім'я_індексу
```

Вилучення індексу ніяк не впливає на інформацію, що міститься в індексованих полях. Після вилучення індекс може бути створений знову.

Зауваження: Типовою причиною вилучення індексів є спроба збільшення продуктивності бази даних. Для досягнення оптимальної продуктивності часто потрібні проведення тривалих експериментів з індексами - їх створення й вилучення з можливим подальшим відтворенням в колишньому або зміненому вигляді.

5.4.5. Робота з представленнями

Представлення (View) є об'єктом бази даних, робота з яким нічим не відрізняється від роботи зі звичайною таблицею. Відмітність представлень від таблиць полягає в наступному. Звичайні таблиці баз даних містять дані. Представлення ж даних не містять, а їх уміст вибирається з інших таблиць (або інших представлень). Таблиці (або представлення), на основі яких формуються представлення, прийнято називати базовими таблицями (або базовими представленнями).

Фактично представлення є запитом, який виконується кожний раз, коли відбувається звернення до представлення. Результат виконання цього запиту в кожен момент часу є змістом представлення. При зміні даних у базових таблицях представлення змінюється й свій зміст. Зміна даних у представленні також приводить до зміни даних у таблицях, на основі яких це представлення створене. На рис. 5.2 схематично пояснюється процес формування представлення:

Використання представлень трохи відрізняється від використання таблиць. Вибірка даних з представлення виконується точно так, як і зі звичайної таблиці. Допускаються також операції маніпулювання даними представлення, хоча тут є деякі обмеження.

Представлення, на відміну від таблиць, не займають дискового простору (або, точніше, дисковий простір, що займається представленнями, дуже малий - тільки той , що потрібний для зберігання запиту).

Рис. 5.2. Схема формування представлення

Області застосування представлень

Представлення в основному застосовуються в двох випадках:

- з метою захисту даних;
- для формування підсумкових даних.

У першому випадку представлення застосовуються для того, щоб надати користувачеві інформацію не зі всієї таблиці, а лише з деяких її полів.

Розглянемо наступний приклад. Нехай, наприклад, інформація про рейтинги співробітників, що зберігається в полі «Рейтинг» таблиці СПІВРОБІТНИКИ, вважається конфіденційною і право доступу до неї мають лише керівники організації. Але частина інформації, що зберігається в цій же таблиці, необхідна працівникам відділу кадрів - дані про імена співробітників і дати їх прийому на роботу. У цьому випадку для розмежування доступу до однієї таблиці зручно використовувати представлення, відібравши для нього тільки ту інформацію, до

якої повинні мати доступ службовці відділу кадрів. При цьому вони зможуть виконувати свої службові обов'язки в повному обсязі й не матимуть доступу до конфіденційної інформації.

Представлення можуть бути використані для обмеження доступу не тільки до полів, але й до записів таблиці. Для цього достатньо в запиті на вибірку даних, на основі якого створюється представлення, вказати відповідну обмежуючу умову. Наприклад, у розглянутому вище прикладі з робітниками відділу кадрів можна при створенні представлення задати умову, яка виключатиме з представлення співробітників, що займають певні посади.

Зауваження: Як уже наголошувалося, представлення будуються на основі запитів до бази даних. Тому в цьому розділі ми наведемо лише загальні відомості про представлення, не вдаючись до подробиць формування запитів.

Представлення також використовуються для формування підсумкових результатів при формуванні звітів. У тому випадку, коли потрібно часто роздруковувати звіт, що формується на основі таблиць з часто змінною інформацією, зручно використовувати представлення. Оскільки представлення може бути створене на основі запиту, що містить пропозиції групування, то можна створити представлення, що одержує інформацію з ряду базових таблиць і групує її в необхідний спосіб, а при виведенні звіту звертатися до цього представлення як до простої таблиці. У цьому випадку не потрібно буде кожного разу при виведенні звіту формувати складний SQL-запит. Крім того, у цьому випадку частина логіки опиниться винесеною на сторону сервера бази даних, оскільки формування звіту не буде залежати від клієнтського застосування.

Створення представлень

Для створення представлень використовується оператор CREATE VIEW. Представлення може бути створене на основі однієї або декількох таблиць і/або інших представлень. Найбільш типовий синтаксис оператора створення представлень має вигляд:

CREATE VIEW ім'я_представленняі AS
{оператор вибірки даних}

Оператор вибірки може бути будь-якій складності, він може містити будь-які умови відбору й пропозиції групування.

Після створення представлення з ним можна працювати як із звичайною таблицею, що має ім'я, задане як ім'я представлення. Деякими виключеннями є представлення, що містять пропозиції групування. Для таких представлень немає ніяких обмежень по вибірці даних, але застосування до них операторів маніпулювання даними (підмножини команд DDL) неприпустимо.

Вилучення представлень

Видалення представлень виконується за допомогою оператора DROP VIEW, при виклику якого можуть указуватися параметри RESTRICT або CASCADE. Дані параметри визначають дії при вилученні представлення, на яке посилаються інші представлення і/або обмеження. При використанні варіанта RESTRICT в цьому випадку буде видано повідомлення про помилку, і вилучення не буде виконано. Якщо ж використовується режим CASCADE, то виконання оператора DROP VIEW приведе до видалення всіх базових представлень і обмежень. Типовий синтаксис оператора DROP VIEW має вигляд:

DROP VIEW ім'я_представлення [RESTRICT | CASCADE]

Зауваження: Вилучення представлення (на відміну від вилучення таблиці) не приводить до видалення даних, на які це представлення посилається. Вилучається лише запит на вибірку даних, на основі якого було створено представлення.

5.4.6. Збережені процедури

Збережені процедури (Stored Procedure) є групами пов'язаних операторів SQL. Використання збережених процедур забезпечує додаткову гнучкість при роботі з базою даних, оскільки виконати збережену процедуру зазвичай набагато простіше, ніж послідовність окремих операторів SQL.

Збережені процедури зберігаються в базі даних у відкомпільованому вигляді, що забезпечує вищу швидкість їх виконання.

Збережені процедури, можуть отримувати вхідні параметри, повертати значення додатку й можуть бути викликані

явно з додатку або підстановкою замість імені таблиці в інструкції SELECT.

Основні переваги, які дає використання збережених процедур, полягають у тому, що:

- Збережені процедури, дозволяють винести частину логіки на сервер бази даних. Це послаблює залежність бази даних інформаційної системи від клієнтської частини;
- Збережені процедури забезпечують модульність проекту: вони можуть бути спільними для клієнтських додатків, які звертаються до однієї й тієї ж бази даних, що дозволяє уникати коду, який повторюється, і зменшується розмір додатків;
- Збережені процедури спрощують супровід додатків: при оновленні процедур зміни автоматично відбиваються в усіх додатках, які їх використовують, без необхідності повторної компіляції й збирання;
- Збережені процедури підвищують ефективність роботи інформаційної системи: вони виконуються сервером, а не клієнтом, що знижує мережевий трафік;
- швидкість виконання збережених процедур вища, ніж для послідовності окремих операторів SQL. Це пов'язано з тим, що процедури зберігаються на сервері у відкомпільованому вигляді.

Розрізняють два види збережених процедур:

- *процедури вибору*, які додатки можуть використовувати замість таблиць або представлень в операторові вибірки даних. Процедура вибору повинна повертати одне або декілька значень, інакше результатом виконання процедури буде помилка;
- *виконувані процедури*, які викликаються явно з використанням спеціального оператора. Виконувана процедура може не повертати результату програмі, що викликає.

Створення збережених процедур

Для створення збереженої процедури використовується оператор CREATE PROCEDURE. Синтаксис цього оператора сильно залежить від використовуваної реалізації SQL, тому ми не будемо його детально розглядати.

Оператор CREATE PROCEDURE визначає нову збережену процедуру в базі даних. Мова процедур сильно залежить від реалізації SQL, але, як правило, містить усі інструкції SQL для маніпулювання даними і ряд розширень, які охоплюють:

- умовні оператори;
- різні види операторів циклу;
- можливості обробки виняткових ситуацій.

Збережені процедури, складаються із заголовка і тіла. Заголовок процедури містить:

- ім'я процедури, яке повинно бути унікальним серед імен процедур і таблиць у базі даних;
- список вхідних параметрів та їх типів даних, які процедура приймає з програми, що викликає (може бути відсутнім);
- список вихідних параметрів і їх типів даних, якщо процедура повертає значення у програму, що викликає.

Тіло процедури містить:

- список локальних змінних і їх типів даних (якщо вони використовуються в коді процедури);
- блок інструкцій на мові процедур і тригерів, що міститься між ключовими словами BEGIN і END. Блок може містити інші блоки, реалізовуючи декілька рівнів вкладеності.

Виконання збережених процедур

Оператор, що запускає збережену процедуру на виконання залежить від типу процедури. Процедури вибору виконуються при зверненні до них за допомогою оператора вибірки даних SELECT.

Для виклику виконуваної процедури треба використовувати спеціальний оператор EXECUTE. Синтаксис цього оператора залежить від використовуваної реалізації SQL.

Вилучення збережених процедур

Для вилучення збережених процедур, використовується оператор DROP PROCEDURE. Синтаксис цього оператора є достатньо загальним для різних реалізацій SQL і має вигляд:

DROP PROCEDURE ім’я_збереженої_процедури

5.4.7. Тригери

Тригери є різновидом збережених процедур. Проте на відміну від збережених процедур, виконання тригера відбувається не в результаті явного виклику деякого оператора SQL, а при виконанні одного з операторів маніпулюванням даними, які вносять зміни в базу даних. При цьому тригери можуть виконуватися як до, так і після виконання оператора маніпулювання даними.

Зауваження: У певному значенні тригери є аналогами оброблювачів подій мови Object Pascal (і ряду інших мов).

Тригери використовуються для забезпечення посилальної цілісності даних у базі.

Вони надають такі можливості:

- можливість контролю даних, що уводяться, щоб гарантувати, що користувач увів у поля таблиці тільки допустимі значення;
- спрощення супроводу додатків, оскільки зміна в тригері автоматично відбивається у всіх додатках, які використовують таблиці з пов'язаними з ними тригерами;
- автоматичне документування змін таблиці. Додаток може керувати журналом змін за допомогою тригерів, які виконуються всякий раз, коли відбувається зміна таблиці.

Створення тригера

Для створення тригера використовується оператор CREATE TRIGGER. Синтаксис цього оператора істотно залежить від використовуваної реалізації SQL, тому ми не розглядатимемо його детально й поговоримо лише про загальні особливості створення тригерів.

Так само, як і збережені процедури, тригери складаються із заголовка й тіла. Заголовок тригера містить:

- ім'я тригера, унікальне в середині бази даних;
- ім'я таблиці, з якою пов'язаний тригер;
- інструкції, які визначають, коли тригер виконуватиметься (при виконанні якого оператора маніпулювання даними й в який момент часу - до або після виконання оператора).

Тіло тригера містить:

- список локальних змінних та їх типів даних (якщо вони використовуються в коді тригера);
- блок інструкцій мовою процедур і тригерів, що знаходиться між ключовими словами BEGIN і END. Блок може містити в собі інший блок, реалізовуючи декілька рівнів вкладеності.

Отже, відмітність тригера від збереженої процедури, полягає тільки в заголовку.

Тригер пов'язаний з таблицею. Власник таблиці й будь-який користувач, наділений привілеями на таблицю, автоматично мають права виконувати пов'язані з нею тригери.

Зауваження: Після створення тригера до нього не можна внести зміни. Щоб внести зміни до вже створеного тригера, необхідно вилучити його й створити наново. Деякі реалізації SQL також допускають заміну тригера (у тому випадку, якщо тригер з указаним ім'ям уже існує) при виконанні оператора CREATE TRIGGER (при цьому немає необхідності явно вилучати раніше створений тригер).

Вилучення тригера

Для вилучення тригера використовується оператор DROP TRIGGER. Синтаксис цього оператора достатньо спільний для різних реалізацій SQL і має такий вигляд:

DROP TRIGGER ім'я_тригера

5.5. Маніпулювання даними

Для маніпулювання даними, які зберігаються в базі даних, використовується група операторів SQL, що виділяється як окремий тип команд, званих мовою маніпулювання даними (DML - Data Manipulation Language). За допомогою операторів DML користувач може завантажувати в таблиці нові дані, модифікувати і вилучати існуючі дані.

У мові SQL визначені тільки три основні оператори DML:

1. INSERT;
2. UPDATE;
3. DELETE.

5.5.1. Додавання в таблицю нової інформації

Процес уведення в таблицю бази даних нової інформації зазвичай називається *завантаженням даних*. Для завантаження даних використовується оператор INSERT.

Додавання до таблиці нового запису

Для додавання до таблиці нового запису використовується така синтаксична форма оператора INSERT:

```
INSERT INTO ім'я_таблиці
VALUES (значення_1, значення_2, ....., значення_N)
```

При використанні даної форми оператора INSERT список VALUES повинен містити кількість значень, що дорівнює кількості полів таблиці. Причому тип даних кожного з указаних значень у списку VALUES повинен збігатися з типом даних поля, що відповідає цьому значенню.

Зауваження: Послідовність полів визначається послідовністю їх опису в операторі CREATE TABLE, за допомогою якого таблиця була створена.

Значення, що відносяться до символьних типів і дат, повинні бути поміщені в апострофи. У списку значень може також використовуватися значення NULL.

Розглянемо приклад. Таблиця ПОСАДИ була створена з використанням такого оператора:

```
CREATE TABLE Посади (
Код_посади INTEGER NOT NULL PRIMARY KEY,
Посада VARCHAR(50) NOT NULL UNIQUE,
Розряд INTEGER NOT NULL,
Зарплатня DECIMAL(7,2) NOT NULL)
```

Для додавання нового запису в цю таблицю треба використовувати такий оператор INSERT:

```
INSERT INTO Посади
VALUES (12, 'Провідний програміст', 12, 2000.00)
```

Уведення даних в окремі поля таблиці

При додаванні даних у таблицю можна заповнювати не всі поля, а лише деякі з них. У цьому випадку використовується така синтаксична форма оператора INSERT:

INSERT INTO ім'я_таблиці (ім'я_поля_1, ім'я_поля_2, ..., ім'я_поля_N)
VALUES (значення_1, значення_2, ..., значення_N)

Наприклад, при додаванні інформації про нового співробітника в таблицю ФІЗИЧНІ ОСОБИ необхідно вказати тільки інформацію про повне ім'я співробітника. У цьому випадку можна використовувати такий оператор:

INSERT INTO Фізичні_особи (Код_фізичної_особи, Ім'я, Прізвище, По батькові)
VALUES (234, 'Петренко', 'Федір', 'Михайлович')

Зауваження: Список полів в операторові INSERT може мати довільний порядок, не залежний від порядку задавання полів при створенні таблиці. Проте список значень повинен відповідати порядку, згідно з яким указані поля пов'язані з цими значеннями.

При виконанні даного оператора в усі решта поля буде занесене значення NULL. Природно, що поля, які не вказуються в круглих дужках після імені таблиці, не повинні мати обмеження NOT NULL, інакше спроба виконання оператора INSERT виявиться невдалою.

Занесення до таблиці даних, що містяться в іншій таблиці

Іноді потрібно перенести частину інформації з однієї таблиці в іншу. Такого роду операцію можна виконати за допомогою комбінації оператора INSERT з оператором вибірки даних SELECT.

Зауваження: За допомогою оператора вибірки даних SELECT формується запит - звернення до бази даних з метою отримання певної інформації.

Об'єднуючи оператори INSERT і SELECT, можна додати в таблицю дані, отримані в результаті виконання запиту з іншої таблиці (таблиць). Синтаксис оператора INSERT в цьому випадку матиме вигляд:

INSERT INTO ім'я_таблиці (ім'я_поля_1, ім'я_поля_2, ..., ім'я_поля_N)
SELECT [* | ім'я_поля_1, им'я_поля_2, ..., ім'я_поля_N]
FROM ім'я_таблиці
WHERE умова

У даному операторові замість пропозиції VALUES використовується оператор SELECT (див. параграф 5.5.3). Стисло пояснимо синтаксис цього оператора. Після слова SELECT вказується список полів, значення яких включаються у вибірку (якщо після SELECT вказати символ *, то у вибірку будуть включені всі поля). Пропозиція FROM використовується для вказівки імені таблиці, з якої проводиться вибірка даних. Пропозиція WHERE є необов'язковою і використовується для накладення обмежень на дані, що включаються у вибірку.

Кількість полів, що вказуються в круглих дужках після імені таблиці в операторові INSERT, повинна дорівнювати кількості полів, що включаються у вибірку. Відповідність полів визначається порядком їх слідування: першому полю в списку оператора INSERT відповідає перше поле в списку оператора SELECT і т.д.

5.5.2. Зміна даних, що зберігаються в таблиці

Для зміни даних, уже занесених до таблиці, використовується оператор UPDATE. Даний оператор не додає нових записів у таблицю, а замінює існуючі дані на нові. Оператор UPDATE може бути застосований як до одного поля таблиці (найбільш частий випадок), так і до декількох полів. Кількість змінюваних записів залежить від потреб користувача - за допомогою UPDATE можна

змінити як один, так і декілька записів (аж до зміни значення всіх записів, що містяться в таблиці).

Модифікація даних в одному полі таблиці

Для зміни даних тільки в одному з полів таблиці використовується найбільш проста форма оператора UPDATE, яка має наступний вигляд:

```
UPDATE ім'я_таблиці
SET ім'я_поля = значення, ...
[WHERE умова]
```

Сенс окремих синтаксичних елементів оператора UPDATE достатньо зрозумілий: після ключового слова UPDATE вказується ім'я таблиці, в якій модифікуються дані, після ключового слова SET виконується присвоєння полю із заданим ім'ям нового значення. Умова, що задається за допомогою необов'язкової пропозиції WHERE, визначає кількість записів, які будуть змодифіковані.

Зауваження: Умова, що вказується в пропозиції WHERE оператора UPDATE, формується за тими самими правилами, що й умова, яка задається в пропозиції WHERE оператора SELECT.

Розглянемо приклад. Припустимо, потрібно змінити номер телефону співробітника організації, що зберігається в таблиці ФІЗИЧНІ ОСОБИ (така необхідність може виникнути або при зміні номера телефону, або в разі коректування помилково занесених даних). У цьому випадку оператор UPDATE повинен змінити значення тільки одного поля і лише в одному записі. Тому в пропозиції WHERE необхідно вказати таку умову, яка б вибирала необхідний нам запис. Найбільш простим рішенням для відбору потрібного запису буде поле первинного ключа «Код_фізичної_особи». Значення, що зберігаються в цьому полі, унікальні й однозначно визначають співробітника. Тоді оператор UPDATE, що виконує зміну номера телефону, матиме вигляд:

```
UPDATE Фізичні_особи
SET Телефон = '(095) 2347890'
WHERE Код_фізичної_особи = 16
```

Даний оператор змінить значення номера телефону тільки для запису, що відповідає співробітникові, зареєстрованому в базі даних під номером 16. Якби ми не задали обмежувальної умови в наведеному вище операторові, то значення номера телефону було б змінене для всіх записів у таблиці

Зауваження: При використанні оператора UPDATE необхідно бути дуже уважним і правильно формулювати обмежувальні умови. Інакше виконання оператора UPDATE може привести до втрати інформації, що зберігається в базі даних.

Зміна значень у декількох полях таблиці

За допомогою оператора UPDATE можна одночасно змінювати значення в декількох полях таблиці. Для цього треба указати після ключового слова SET не одне, а декілька полів:

```
UPDATE ім'я_таблиці
SET ім'я_поля_1 = значення_1,
ім'я_поля_2 = значення_2,
.....,
ім'я_поля_N = значення_N
[WHERE умова]
```

Використання оператора в даній формі нічим не відрізняється від розглянутого раніше. Тут так само потрібно бути дуже обережним при формуванні умови.

Вилучення даних з таблиці

Вилучення даних з таблиці виконується за допомогою оператора DELETE. Даний оператор повністю вилучає весь запис, а не дані з окремих полів. Синтаксис оператора DELETE має вигляд:

```
DELETE FROM ім'я_таблиці
[WHERE умова]
```

Записи, що вилучаються, визначаються відповідно до умови, заданої за допомогою необов'язкової пропозиції WHERE. За відсутності пропозиції WHERE в операторі DELETE дані будуть вилучені з усієї таблиці.

5.5.3. Вибірка даних з таблиць

Для добування записів з таблиць в SQL визначений оператор SELECT. За допомогою цієї команди реалізовується не лише операція реляційної алгебри "вибірка" (горизонтальна підмножина), але й попереднє з'єднання (join, див. Додаток 2) двох або більше таблиць. Це скомплектований та могутній засіб SQL, повний синтаксис оператора SELECT має вигляд:

```
SELECT [ALL | DISTINCT]  список_вибору
FROM  ім'я_таблиці, ...
  [ WHERE  умова ]
  [ GROUP BY ім'я_стовпця,... ]
  [ HAVING  умова ]
  [ORDER BY  ім'я_стовпця [ASC | DESC],... ]
```

Послідовність команд в операторі SELECT повинна бути строго дотримана (наприклад, GROUP BY має завжди іти попереду ORDER BY), інак це приведе до появи помилок.

Почнемо розгляд SELECT з найбільш простих його форм.

Цей оператор завжди починається із ключового слова SELECT. У конструкції «список_вибору» визначається стовпець або стовпці, що включаються до результату. Він може утворюватися з імен одного або кількох стовпців або з одного символу «*» (зірочка), що визначає всі стовпці. Елементи списку відокремлюються комами.

Для початку створимо базу даних PUBLICATION:

```
CREATE DATABASE publications;
CREATE TABLE authors (au_id INT PRIMARY KEY,
            author VARCHAR(25) NOT NULL);
CREATE TABLE publishers (pub_id INT PRIMARY KEY,
                    publisher VARCHAR(255) NOT NULL, url
                                        VARCHAR(255));
CREATE TABLE titles (title_id INT PRIMARY KEY,
            title VARCHAR(255) NOT NULL, yearpub INT,
            pub_id INT REFERENCES publishers (pub_id));
CREATE TABLE titleautors (au_id INT REFERENCES authors
(au_id),
                title_id INT REFERENCES titles (title_id));
```

```
CREATE TABLE wwwsites (site_id INT PRIMARY KEY,
        site VARCHAR(255) NOT NULL, url
VARCHAR(255));
CREATE TABLE wwwsiteauthors (au_id INT REFERENCES
authors(au_id), site_id INT REFERENCES wwwsites (site_id));
```

Приклад: одержати список усіх авторів (для прикладу розглянемо базу даних певного видавництва).

```
SELECT author FROM authors;
```

одержати список усіх полів таблиці «authors»:

```
SELECT * FROM authors;
```

У випадку, коли нас інтерисують не всі записи, а тільки ті, що забезпечують певну умову, цю умову можна ввести після ключового слова WHERE. Наприклад, знайдемо всі книги, видані після 1996 року:

```
SELECT title FROM titles WHERE yearpub > 1996;
```

Припустімо тепер, що нам треба знайти усі публікації за період 1995 - 1997 р. Цю умову варто записати наступним чином:

```
SELECT title FROM titles WHERE yearpub >=
1995 AND yearpub <= 1997;
```

Інший варіант цієї команди можна одержати з використанням логічної операції перевірки на входження в інтервал:

```
SELECT title FROM titles WHERE yearpub
BETWEEN 1995 AND 1997;
```

При застосуванні конструкції NOT BETWEEN будуть знайдені всі рядки, які не входять до вказаного діапазону.

Ще одину варіацію цієї команди можна створити за допомогою логічної операції перевіряння на наявність у списку:

```
SELECT title FROM titles WHERE yearpub IN
(1995,1996,1997);
```

Тут ми навели в явному вигляді список значень, що нас цікавлять. Конструкція NOT IN дозволяє знайти рядки, які не задовольняють умови, перераховані у списку.

Найбільш явно переваги ключового слова IN проявляються у вкладених запитах, що також мають назву підзапити. Припустимо, нам потрібно відшукати всі видання, надруковані компанією "Oracle Press". Найменування видавничих компаній внесені до таблиці «Видавці», назви книг – до таблиці «titles». Ключове слово NOT IN дає змогу поєднати обидві таблиці (без одержання загального відношення) і внаслідок цього отримати необхідну інформацію:

```
SELECT title FROM titles WHERE pub_id IN
(SELECT pub_id FROM titles WHERE
publisher='Oracle Press');
```

При виконанні цієї команди СКБД спершу обробляє вкладений запит по таблиці «publishers», а далі його результат передає на вхід основного запиту по таблиці «titles».

Певні завдання не можна вирішити, використовуючи тільки операторів порівняння. Наприклад, ми хочемо знайти web-site видавництва "Wiley", але не знаємо його точного найменування. Для вирішення цього завдання встановлене ключове слово LIKE, його синтаксис має такий вигляд:

```
WHERE ім'я_стовпця LIKE зразок [ ESCAPE
<ключовий_символ> ]
```

Зразок міститься в лапках і повинен містити шаблон для пошуку. Звичайно, у шаблонах застосовуються два символи:

- % (знак відсотка) - замінює будь-яку кількість символів;
- _ (підкреслення) - замінює одиночний символ.

Спробуємо знайти необхідний web-site:

```
SELECT publisher, url FROM publishers WHERE publisher LIKE '%Wiley%';
```

Відповідно до шаблона СКБД знайде всі рядки, що містять у собі підрядок "Wiley". Інший приклад: віднайти всі книги, назва котрих починається зі слова "SQL":

```
SELECT title FROM titles WHERE title LIKE 'SQL%';
```

У випадку, коли необхідно знайти значення, що саме містить один із символів шаблона, використовують ключове слово ESCAPE й «ключовий_символ». Літерал, що йде в шаблоні після ключового символу, розглядається як звичайний символ, усі наступні символи мають звичайне значення. Для прикладу, нам треба знайти посилання на web-сторінку, про яку відомо, що в її url є підрядок "my_works":

```
SELECT site, url FROM wwwsites WHERE url LIKE '%my@_works%' ESCAPE '@';
```

На закінчення зауважимо, що при застосуванні оператора SELECT результуюче відношення може містити декілька записів з однаковими значеннями всіх полів. Задля виключення повторюваних записів з вибірки використовується ключове слово DISTINCT. Ключове слово ALL указує, що в результат необхідно включати всі рядки.

5.5.4. Вибірка даних з декількох таблиць

Доволі часто виникає ситуація, у якій вибірку даних потрібно робити з відношення, що є результатом злиття (join, див. Додаток 2) двох інших відношень. Наприклад, нам потрібно одержати з бази даних PUBLICATIONS інформацію про всі друковані видання у вигляді такої таблиці:

назва_книги	рік_видання	видавництво

Для цього СКБД попередньо повинна виконати злиття таблиць TITLES й PUBLISHERS, а тільки потім зробити вибірку з отриманого відношення.

Для здійснення операції такого характеру в операторі SELECT після ключового слова FROM указується список таблиць, по яких проводиться пошук даних. Після ключового слова WHERE вказується умова, за якою здійснюється злиття. Для того, щоб виконати даний запит, необхідно дати команду:

```
SELECT titles.title, titles.yearpub, publishers.publisher
  FROM titles, publishers
  WHERE titles.pub_id=publishers.pub_id;
```

Ще один приклад, де водночас задаються умови і злиття, і вибірки (результат попереднього запиту обмежено виданнями після 1996 року):

```
SELECT titles.title, titles.yearpub, publishers.publisher
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id AND
      titles.yearpub>1996;
```

Варто зауважити, що коли в різних таблицях присутні однойменні поля, то задля усунення неоднозначності перед ім'ям поля вказується ім'я таблиці та знак "." (крапка). (Хороше правило: *завжди вказувати ім'я таблиці!*)

Закономірно, є можливість робити злиття й більш ніж двох таблиць. Скажімо, щоб доповнити описану вище вибірку іменами авторів книг, треба скласти оператор такого виду:

```
SELECT authors.author, titles.title, titles.yearpub,
publishers.publisher
FROM titles, publishers, titleauthors
```

```
WHERE titleauthors.au_id=authors.au_id AND
      titleauthors.title_id=titles.title_id AND
      titles.pub_id=publishers.pub_id AND
      titles.yearpub > 1996;
```

5.5.5. Обчислення всередині SELECT

SQL дає змогу виконувати різні арифметичні операції над стовпцями результуючого відношення. У конструкції «список_вибору» можна застосувати константи, функції та їх комбінації з арифметичними операціями та дужками. Наприклад, щоб довідатися скільки років минуло з 1992 року (рік прийняття стандарту SQL-92) до публікації тієї або іншої книги треба виконати команду:

```
SELECT title, yearpub-1992 FROM titles WHERE
yearpub > 1992;
```

В арифметичних виразах допускаються операції додавання (+), віднімання (-), ділення (/), множення (*), а також інші функції (COS, SIN, ABS - абсолютне значення й т.д.). Також у запит можна додати рядкову константу:

```
SELECT 'the title of the book is', title, yearpub-
1992
FROM titles WHERE yearpub > 1992;
```

У SQL теж визначені так звані агрегатні функції, котрі роблять дії над сукупністю однакових полів у групі записів. Серед них:

- **AVG** (ім'я поля) - середнє за всіма значеннями даного поля;
- **COUNT** (ім'я поля) або COUNT (*) - кількість записів;
- **MAX** (ім'я поля) - максимальне із усіх значень даного поля;
- **MIN** (ім'я поля) - мінімальне зо всіх значень даного поля;
- **SUM** (ім'я поля) - сума всіх значень даного поля.

Варто враховувати, що кожна агрегуюча функція повертає єдине значення. Для прикладу: визначити дату публікації "найстарішої" книги в наявній базі даних

```
SELECT MIN(yearpub) FROM titles;
```

Виявити кількість книг у нашій базі даних:

```
SELECT COUNT(*) FROM titles;
```

Область дії дані функції можна обмежити за допомогою логічної умови. Наприклад, кількість книг, у назві котрих є слово "SQL":

```
SELECT COUNT(*) FROM titles WHERE title LIKE
'%SQL%';
```

5.5.6. Групування даних

Групування даних в операторі SELECT здійснюються за використання ключового слова GROUP BY і ключового слова HAVING, за допомогою котрого задаються умови розбивки записів на групи.

GROUP BY нерозривно пов'язане з агрегуючими функціями, без них воно практично не застосовується. GROUP BY розподіляє таблицю на групи, а агрегуюча функція розраховує для кожної із них підсумкове значення. Визначимо, скажімо, кількість книг кожного видавництва в наявній базі даних:

```
SELECT publishers.publisher, count (titles.title)
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher;
```

Ключове слово HAVING працює наступним способом: спочатку GROUP BY розбиває рядки на групи, потім на отримані набори накладаються умови HAVING. Наприклад, усунемо з попереднього запиту ті видавництва, що мають лише одну книгу:

```
SELECT publishers.publisher, count (titles.title)
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher
HAVING COUNT(*)>1;
```

Інший варіант застосування HAVING - віднести до результату тільки ті видавництва, назва яких закінчується на підрядок "Press":

```
SELECT publishers.publisher, count (titles.title)
FROM titles, publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher
HAVING publisher LIKE '%Press';
```

У чому відмінність між цими двома варіантами застосування HAVING? У другому варіанті умову відбору записів можна помістити в розділ ключового слова WHERE, у першому ж варіанті цього зробити не можна, оскільки WHERE не допускає використання агрегуючих функцій.

5.5.7. Сортування даних

Для сортування даних, що отримуються за допомогою оператора SELECT, призначене ключове слово ORDER BY. За його допомогою можна сортувати результати за будь-яким стовпцем або виразом, зазначеному в «списку_вибору». Дані можуть бути впорядковані як за зростанням, так і за убуванням. Приклад: сортувати список авторів за алфавітом:

```
SELECT author FROM authors ORDER BY author;
```

Складніший приклад: одержати список авторів, відсортований за алфавітом, і список їх публікацій, причому для кожного автора список книг сортується за часом видання у зворотному порядку (тобто спочатку "свіжіші" книги, потім "старіші"):

```
SELECT authors.author, titles.title, titles.yearpub,
publishers.publisher
FROM authors, titles, publishers, titleauthors
WHERE titleauthors.au_id=authors.au_id AND
      titleauthors.title_id=titles.title_id AND
      titles.pub_id=publishers.pub_id
ORDER BY authors.author ASC, titles.yearpub
DESC;
```

Тут ключове слово DESC задає зворотний порядок сортування за полем «yearpub», ключове слово ASC (його можна опускати) - прямий порядок сортування за полем «author».

5.5.8. Операція об'єднання

У SQL передбачена змога виконання операції реляційної алгебри "ОБ'ЄДНАННЯ" (UNION, див. Додаток 2) над відношеннями, що є результатом оператора SELECT. Природно, ці відношення повинні бути визначені за однією схемою. Приклад: одержати всі Інтернет-посилання, збережені в базі даних «PUBLICATIONS». Ці посилання зберігаються в таблицях «publishers» й «wwwsites». Для того, щоб одержати їх в одній таблиці, ми повинні побудувати такий запит:

```
SELECT publisher, url FROM publishers
UNION
SELECT site, url FROM wwwsites;
```

5.6. Керування безпекою бази даних

Однією з найважливіших завдань керування базами даних є забезпечення безпеки даних, тобто захисту даних від їхнього несанкціонованого використання.

Зауваження: Під несанкціонованим використанням зазвичай розуміється доступ до даним з боку користувачів, що не мають на це має прав.

Забезпечення безпеки даних є дуже серйозним питанням, детальний розгляд, якого вимагає окремої об'ємної книги. Тому тут ми обговоримо лише один з аспектів забезпечення безпеки, а саме - керування доступом до бази даних.

5.6.1. Привілеї користувачів

Привілеями називаються рівні повноважень користувачів. Розмежування доступу до інформації, що зберігається в базі даних, регулюється за допомогою привілеїв. Розрізняють привілеї двох типів:

- системні привілеї;
- об'єктні привілеї.

Розглянемо кожен з типів детальніше.

Системні привілеї

Системні привілеї дають користувачам бази даних можливість виконувати дії, пов'язані з її адмініструванням: створювати, вилучати і змінювати структуру як самої бази даних, так і окремих її об'єктів. Крім того, системні привілеї дають право на зміну стану бази даних і її окремих об'єктів.

Можливі системні привілеї істотно залежать від використовуваної СКБД. Але у будь-якому випадку вони включають такі привілеї, як право (повноваження) на:

- створення таблиці;
- створення представлення;
- створення збереженої процедури;
- вилучення таблиці;
- вилучення представлення;
- вилучення збереженої процедури.

Цей список може бути розширений. Крім того, кожен з привілеїв має свої особливості в різних СКБД.

Об'єктні привілеї

Об'єктні привілеї є рівнями повноважень користувачів, що поширюються на об'єкти бази даних. Це означає, що для того, щоб виконати певні дії над об'єктами бази даних, користувач повинен мати відповідні права (повноваження).

Стандартом ANSI передбачені такі об'єктні привілеї:

- SELECT - дозволяє проводити вибірку даних з указаної таблиці (представлення);
- INSERT (ім’я_поля) - дозволяє виконувати додавання даних у певне поле вказаної таблиці (представлення);

- INSERT - дозволяє додавання даних у всі поля вказаної таблиці;
- UPDATE (ім'я_поля) - дозволяє модифікувати дані в заданому полі вказаної таблиці (представлення);
- UPDATE - дозволяє модифікувати дані у всіх полях указаної таблиці (представлення);
- REFERENCE (ім'я_поля) - дозволяє посилатися па задане поле вказаної таблиці (цей привілей потрібний при встановці будь-яких обмежень цілісності);
- REFERENCE - дозволяє посилатися на все поля вказаної таблиці.

Зауваження: Окрім указаних, існує чимало об'єктних привілеїв, доступних у різних СКБД.

5.6.2. Керування доступом до бази даних

Для керування доступом користувачів до бази даних у мові SQL існують два оператори:

GRANT;
REVOKE.

Як правило, ці оператори використовуються адміністратором бази даних або його помічником з безпеки.

Оператор GRANT

Оператор GRANT використовується для надання користувачеві як системних, так і об'єктних привілеїв. Синтаксис даного оператора має вигляд:

```
GRANT привілей_1 [, привілей_2]
ON ім'я_об'єкту
TO ім'я_користувача [WITH GRANT OPTION]
```

Надання користувачеві з ім'ям USER права на вибір даних з таблиці СПІВРОБІТНИКИ виконується за допомогою такого оператора:

```
GRANT SELECT
ON Співробітники
TO USER
```

За допомогою одного оператора GRANT можна задавати відразу декілька привілеїв. Наприклад, наступний оператор надасть користувачеві USER право як проглядати, так і додавати дані в таблицю СПІВРОБІТНИКИ:

```
GRANT SELECT, INSERT
ON Співробітники
TO USER
```

При виклику оператора GRANT може використовуватися необов'язкова пропозиція WITH GRANT OPTION. Дана пропозиція означає, що користувач, для якого надаються привілеї, також отримує право надавати привілеї на даний об'єкт. Наприклад, якщо викликати розглянутий вище оператор з пропозицією WITH GRANT OPTION, то користувач з ім'ям USER, окрім права переглядати і додавати дані в таблицю СПІВРОБІТНИКИ, отримає також право надавати ці привілеї іншим користувачам:

```
GRANT SELECT, INSERT
ON Співробітники
TO USER
WITH GRANT OPTION
```

Оператор REVOKE

Оператор REVOKE використовується для відміни наданих користувачеві привілеїв. Даний оператор може викликатися з одним із двох параметрів - RESTRICT або CASCADE. При використанні варіанта RESTRICT оператор REVOKE буде успішно виконаний тільки в тому випадку, якщо його виконання не призведе до появи так званих *залишених* привілеїв

Зауваження: Залишеними називаються привілеї, що залишилися в користувача, якому вони були надані за допомогою пропозиції WITH GRANT OPTION оператора GRANT.

При використанні режиму CASCADE вилучаються всі привілеї, які могли б залишитися в інших користувачів. Це означає, що якщо користувачеві USER1 були надані привілеї за допомогою параметра WITH GRANT OPTION, а він надав ці

привілеї користувачеві USER2, то скасування привілеїв користувачеві USER1 в режимі CASCADE приведе до скасування привілеїв і для користувача USER2.

Синтаксис оператор REVOKE має такий вигляд:

REVOKE привілей_1 [, привілей_2]
ON ім'я_об'єкту
FROM ім'я_користувача [RESTRICT | CASCADE]

Наприклад, для скасування права додавання даних в таблицю СПІВРОБІТНИКИ для користувача USER треба використати наступний оператор:

REVOKE INSERT
ON Співробітники
FROM USER

Список літератури

1. Месарович М., Тахакара Я. Общая теория систем: математические основы. – М.: Мир, 1978. – 312 с.
2. Бусленко Н.П. Моделирование сложных систем. – М.: Наука, 1978. – 400 с.
3. Острейковский В.А. Теория систем: Учеб. для вузов. – М.: Высшая школа, 1997. – 240 с.
4. Хомоненко А.Д., Цыганкова В.М., Мальцева М.Г. Базы данных: Уч. для высш.уч.завед. – Спб.: Корона принт, 2000. – 416 с.
5. К. Дж. Кейт. Введение в системы баз данных. – М.: Вильямс, 2002. – 1072 с.
6. Гарсиа-Молина Г., Ульман Д., Уидом Дж. Системы баз данных. Полный курс. – М.:Вильямс, 2004. – 1088 с.
7. Олифер В.Г., Олифер Н.А. Компьютерные сети: принципы, технологии, протоколы. – Санкт-Петербург.: Питер, 2001. – 650с.
8. Миловзоров В.П. Элементы информационных систем: Учеб. для вузов. – М.: Высшая школа, 1989. – 440 с.
9. Якубайтис Э.А. Открытые информационные сети. – М.: Финансы и статистика, 1996. – 135 с.
10. Ложе И.В. Информационные системы: методы и средства. – М.: Мир, 1979. – 640 с.
11. Э. Таненбаум. Компьютерные сети. – М.: Питер, 2002. – 487 с.
12. Хезер Остерлох. TCP/IP. Семейство протоколов передачи данных в сетях компьютеров. – СПб.: ООО «ДиаСофтЮП», 2002. – 520с.
13. Э.Озкарахан "Машины баз данных и управление базами данных" -М: Мир, 1989. – 345с.

Запитання та завдання для самоконтролю

Питання до модуля 1

1. Поняття інформаційної системи.
2. Поняття корпоративної інформаційної системи.
3. Які типові програмні компоненти входять до складу інформаційної системи?
4. Які фактори впливають на розвиток корпоративної інформаційної системи?
5. Назвіть основні складові корпоративної інформаційної системи.
6. Як між собою співвідносяться основні складові корпоративної інформаційної системи?
7. Як класифікуються інформаційні системи?
8. Як класифікуються інформаційні системи за масштабом?
9. Як класифікуються інформаційні системи за сферою застосування?
10. Як класифікуються інформаційні системи за способом організації (архітектурою)?
11. Назвіть функціональні компоненти інформаційної системи.
12. Особливості побудови архітектури файл-сервер.
13. Від чого залежить продуктивність архітектури файл-сервер?
14. Які недоліки архітектури файл-сервер ви знаєте?
15. Особливості побудови архітектури клієнт-сервер.
16. Відмітна риса серверів баз даних.
17. Поняття “збереженої процедури”.
18. Особливості побудови дворівневої архітектури клієнт-сервер.
19. Особливості побудови багаторівневої архітектури.
20. Призначення трирівневої архітектури.
21. Особливості побудови інформаційних систем на основі технології Інтернет/інтранет.
22. Яку структуру має інформаційний додаток при використанні технології Інтернет/інтранет?
23. Поняття проекту.
24. Назвіть основні характерні ознаки проекту, як об’єкта управління?
25. Якими властивостями має володіти проект?
26. Укажіть техніко-економічні показники проекту?
27. Класифікація проектів.
28. Виділіть основні типи проектів.
29. Як поділяються проекти за масштабом?

30. Виділіть основні фази розвитку інформаційної системи.
31. Що містить концептуальна фаза розвитку інформаційної системи?
32. Що містить фаза розробки технічної пропозиції?
33. Що містить фаза проектування інформаційної системи?
34. Що містить фаза розробки інформаційної системи?
35. Що містить фаза уведення системи в експлуатації?
36. Які основні помилки трапляються у початкових фазах?
37. Поняття життєвого циклу інформаційної системи.
38. На яких групах процесів заснована структура життєвого циклу інформаційних систем?
39. Який стандарт регламентує життєвий цикл інформаційних систем?
40. Що відноситься до основних процесів життєвого циклу інформаційних систем?
41. Які роботи передбачає розробка інформаційних систем?
42. Які роботи передбачає експлуатація інформаційних систем?
43. Які роботи передбачає супровід інформаційних систем?
44. Що передбачає технічне й організаційне забезпечення проекту?
45. Поняття верифікації розробки.
46. Поняття перевірки розробки.
47. Поділ життєвого циклу інформаційних систем згідно методології Rational Software.
48. Які моделі життєвого циклу інформаційних систем ви знаєте?
49. Яку організацію робіт передбачає каскадна модель?
50. Основні етапи розробки каскадної моделі.
51. Основні переваги каскадної моделі.
52. Основні недоліки каскадної моделі.
53. Яку організацію робіт передбачає спіральна модель?
54. Основні переваги спіральної моделі.
55. Проблеми, що виникають при використанні спіральної моделі.
56. Які завдання повинна забезпечувати методологія створення корпоративних інформаційних систем?
57. На чому заснована методологія RAD?
58. На яких елементах заснована методологія RAD?
59. Основні принципи методології RAD.
60. Об’єктно-орієнтований підхід у методології RAD.
61. Візуальне програмування в методології RAD.
62. Які види корпоративних стандартів ви знаєте?

63. Поняття профілю інформаційної системи.
64. Які групи профілів ви знаєте?
65. Які завдання дозволяє вирішити використання профілів інформаційних систем?
66. Що включають у себе профілі інформаційної системи з ієрархічною структурою?
67. Що повинні містити описи профілів для їх коректного застосування?
68. Які основні функціональні профілі ви знаєте?

Питання до модуля 2

1. Укажіть основні напрямки використання обчислювальної техніки?
2. Які вимоги ставляться до обчислювальної техніки в інформаційних системах?
3. Коли з'явилися перші системи керування базами даних?
4. Якими властивостями володіють системи керування базами даних?
5. Наведіть основні функції, якими мають володіти систем керування базами даних.
6. Поняття транзакції.
7. Поняття серіалізації транзакцій.
8. Укажіть види апаратних збоїв?
9. Що таке журнал бази даних?
10. Який протокол має підтримуватися СКБД для відновлення системи після будь-якого збою?
11. Що потрібно для відновлення бази даних після "жорсткого" збою?
12. Стандартна мова реляційних СКБД.
13. Що дозволяє мова SQL?
14. Які моделі даних ви знаєте?
15. Основні характеристики ієрархічної моделі даних.
16. Переваги ієрархічної моделі даних.
17. Основні недоліки ієрархічної моделі даних.
18. Основні характеристики мережевої моделі даних.
19. Переваги ієрархічної моделі даних.
20. Основні недоліки мережевої моделі даних.
21. Основні характеристики реляційної моделі даних.
22. Основні характеристики постреляційної моделі даних.
23. Основні характеристики багатомірної моделі даних.
24. Основні характеристики СКБД першого покоління.
25. Недоліки СКБД першого покоління.
26. Що було основою для початку створення реляційних СКБД?
27. Що запропонував Кодд для обробки даних?
28. Принципи, запропоновані Коддом.
29. Характеристики об'єктно-орієнтованих СКБД.
30. Головне завдання реляційної моделі даних.
31. Основні переваги реляційного підходу.

32. Недоліки реляційних баз даних.
33. Основні поняття реляційної моделі даних.
34. Які типи даних ви знаєте?
35. Поняття домену.
36. Поняття атрибуту.
37. Що таке схема відношення?
38. Що таке степінь відношення?
39. Що таке схема бази даних?
40. Поняття кортежу.
41. Що таке кардинальне число або потужність відношення?
42. Поняття порожнього значення.
43. Поняття первинного ключа.
44. Строге визначення первинного ключа.
45. Поняття простого ключа.
46. Поняття складного, або складеного ключа.
47. Поняття суперключа.
48. Поняття штучного, або сурогатного ключа.
49. Поняття природного ключа.
50. Переваги природних ключів.
51. Недоліки природних ключів.
52. Поняття потенційного, або альтернативного ключів.
53. Поняття вторинного ключа.
54. Поняття зовнішнього ключа.
55. Поняття зовнішнього рекурсивного ключа.
56. Які обмеження цілісності ви знаєте?
57. Поняття категорійної цілісності.
58. Поняття посилальної цілісності.
59. Назвіть три підходи для вилучення кортежу з відношення?
60. Поняття головної таблиці.
61. Поняття підлеглої таблиці.
62. Назвіть типи зв’язків між таблицями реляційної бази даних?
63. Укажіть основні властивості відношень?
64. Визначення реляційної бази даних.
65. Поняття індексу.
66. Призначення індексів.
67. Наведіть типи індексів?
68. Поняття простого індексу.
69. Поняття складеного індексу.

70. Умови оптимальності слідування стовпців у складеному індексі.
71. Поняття унікального індексу.
72. Поняття нормалізації даних.
73. Проблеми, що виникають при використанні ненормалізованих таблиць.
74. Надмірність даних.
75. Аномалії оновлення.
76. Аномалії вилучення.
77. Аномалії уведення.
78. Мета нормалізації.
79. Назвіть нормальні форми, які ви знаєте?
80. Основні властивості нормальних форм.
81. Поняття першої нормальної форми.
82. Що потрібно зробити, щоб перейти від першої нормальної форми до другої?
83. Що потрібно зробити, щоб перейти від другої нормальної форми до третьої?
84. Типи команд SQL.
85. Типи даних SQL/92.
86. Створення, модифікація й видалення таблиць.
87. Задавання обмежень.
88. Задавання значень за замовчанням.
89. Поняття тригерів.
90. Поняття індексів.
91. Поняття збережених процедур.
92. Поняття представлень.
93. Керування доступом до бази даних.

Список індивідуальних завдань (рефератів)

1. СКБД ORACLE: характеристики, переваги, недоліки, особливості застосування.
2. СКБД DB2: характеристики, переваги, недоліки, особливості застосування.
3. СКБД Visual FOXPRO: характеристики, переваги, недоліки, особливості застосування.
4. СКБД SYBASE: характеристики, переваги, недоліки, особливості застосування.
5. СКБД MICROSOFT ACCESS: характеристики, переваги, недоліки, особливості застосування.
6. Застосування вищих нормальних форм (нормальна форма Бойса-Кодда, 4, 5).
7. Методологія функціонального моделювання. Нотація Йордона – Де Марко.
8. Методологія SADT (IDEF0).
9. Огляд CASE-системи Power Designer.
10. Огляд CASE-системи BPWin й ERWin.
11. Огляд CASE-системи Designer/2000.
12. Огляд CASE-системи Rational Rose.
13. Постреляційні СКБД.
14. Об’єктно-зорієнтовані СКБД.
15. Стандарт ODMG.
16. Об’єктно-реляційні СКБД.
17. Нечислова обробка й асоціативні процесори.
18. Інтерфейс ODBC.
19. Інтерфейс JDBC.

Лабораторний практикум

Лабораторна робота № 1
СТВОРЕННЯ КОНЦЕПТУАЛЬНИХ СХЕМ

Мета роботи: Знайомство з поняттям баз даних. Створення концептуальної схеми.

Завдання до роботи:

1. Ознайомитись з поняттям бази даних і методами роботи з нею.
2. Створити концептуальну схему.

ТЕОРЕТИЧНА ЧАСТИНА

Моделі даних. Концептуальна схема й підсхеми

Обчислювальна техніка з кожним роком усе ширше застосовується в різних сферах людської діяльності. Різко збільшений потік інформації приводить до необхідності розробки нових прийомів її осмислення й обробки. Накопичений досвід та розвиток програмного забезпечення обчислювальної техніки дає змогу переосмислити таку традиційну область обробки інформації, як зберігання й керування даними. Новий підхід до організації процесу обробки інформації приводить до поняття *база даних* і методів роботи з нею. У загальному розумінні *база даних* – це набір записів і файлів, організованих особливим чином.

Зупинимося на певних визначеннях, що застосовуються при проектуванні й експлуатації баз даних. Перше базове поняття – це інформація й дані. Під *інформацією* розуміють будь-які відомості про якусь подію, процес, явище, необхідні при їхньому описі й дослідженні. Інформація, подана в певному стандартному вигляді, називається *даними.* Так, опис завдань, що виникають при роботі з потоком інформації, можна здійснювати із двох перспектив: з позиції інформації – тоді ми приходимо до концептуальної схеми, або з позиції даних – тоді отримаємо відображення концептуальної схеми в даталогічне середовище, тобто фізичну схему завдання.

При постановці інформаційних завдань, насамперед, варто виявити *предметну область,* тобто область, для якої будуть визна-

чені об'єкти дослідження та їх взаємодія між собою. Ця область може бути визначена досить широко, наприклад сфера керування підприємством, транспортом, сфера наукових досліджень тощо. Однак досить широко визначена певна предметна область може бути звужена, тобто може бути виділена деяка підобласть для розв'язання деяких більш вузьких підзавдань вихідної проблеми. Скажімо, якщо предметну область визначити завданням керування аеропортом, то при роботі касира важливі наявність або відсутність квитків на задані рейси й наявність сприятливих погодніх умов, а для організації перевезень – наявність необхідних літаків, а також допущених до перевезень пілотів.

Як приклад опису предметної області розберемо діяльність торговельної фірми, що володіє мережею магазинів. У загальному вигляді її функціонування можливо зобразити у вигляді наступної схеми (рис.1).

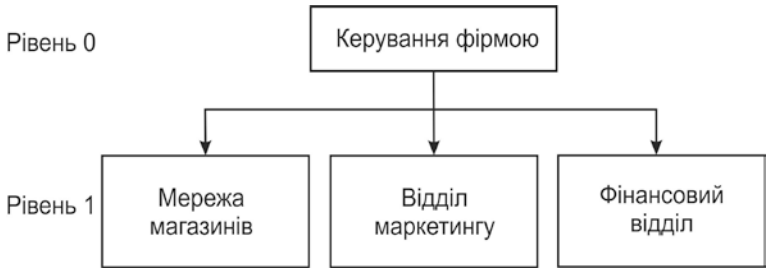


Рис. 1. Схема предметної області

Розглянемо приклад побудови концептуальної схеми роботи бібліотеки. Виділимо підсхеми роботи зі списком читачів, бібліотечним фондом, відділом зовнішніх зв'язків і бухгалтерським відділом.

Як підсхему можна розглянути будь-який із блоків першого рівня. У свою чергу, кожний із блоків також може бути зображений у вигляді аналогічної схеми. Наприклад, мережа магазинів може бути подана у вигляді двох блоків: опис діяльності конкретного магазину й система зв'язку й обміну між ними. У свою чергу, функціонування магазину можна зобразити у вигляді блоків, що описують доставку товару, наявність товару

на складі магазину й оптовому складі фірми, фінансовий відділ магазину й відділ роботи з покупцями, що також містить у собі відділ продажів і доставки.

Таблиця 1.
Об’єкти та їх атрибути

Об’єкт	Атрибут	Первинний ключ
Бібліотека	Код бібліотеки Назва бібліотеки Адреса Графік роботи	Код бібліотеки
Список читачів	Код читача ПІБ Замовив книгу	Код читача
Бібліотечний фонд	Код книги Кількість книжок Дата випуску	Код книги
Відділ зовнішніх зв’язків	Код зв’язку Назва закладу Адреса Телефон	Код зв’язку
Бухгалтерський відділ	Код працівника Зарплатня Стаж роботи Дотації на нові книги	Код працівника

Таблиця 2.
Типи зв’язків

Зв’язок	Об’єкти	Тип зв’язку
Складає	Бібліотека Список читачів	1:M
Має	Бібліотека Бібліотечний фонд Бухгалтерський відділ	1:M
Має зв’язки	Бібліотека Відділ зовнішніх зв’язків	1:M

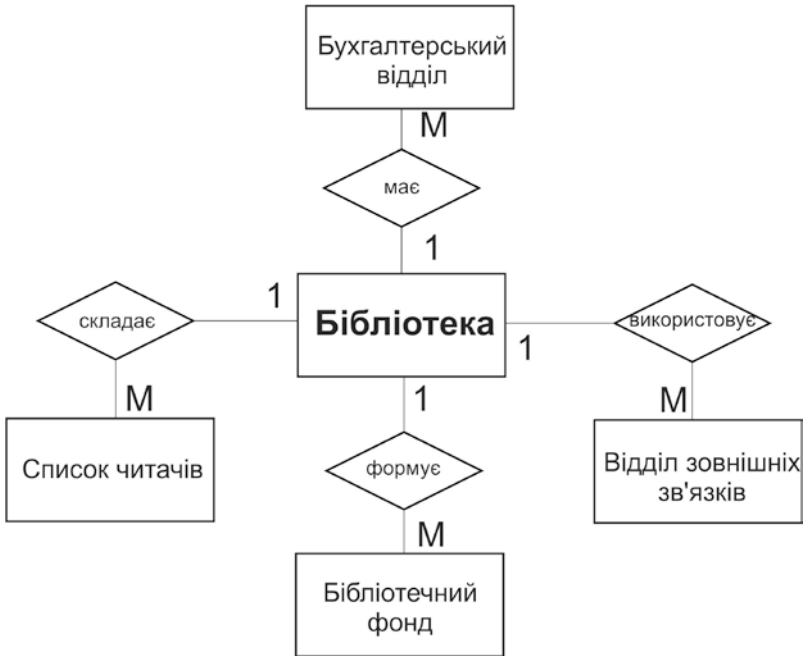


Рис. 2. Концептуальна схема

Завдання

Завдання 1. Опишіть повністю кожний із блоків першого рівня схеми (рис. 1), виділяючи в кожному з них основні завдання. Наприклад відділ маркетингу.

Завдання 2. Опишіть предметну область та розробіть концептуальну схему функціонування якогось факультету вищого навчального закладу.

Завдання 3. Опишіть предметну область й розробіть концептуальну схему функціонування якогось установи культури (театр, художня галерея, філармонія тощо) з погляду репертуарного плану.

Завдання 4. Опишіть предметну область і розробіть концептуальну схему функціонування авторемонтної майстерні з погляду формування замовлень на ремонт автомобіля.

Завдання 5. Опишіть предметну область і розробіть концептуальну схему функціонування фермерського господарства з погляду поставки вироблених товарів на ринок.

Завдання 6. Опишіть предметну область і розробіть концептуальну схему проведення спортивної олімпіади студентів України для ведення протоколів проведених змагань.

Завдання 7. Опишіть предметну область та розробіть концептуальну схему функціонування оптової товарної бази для ведення документації обробки замовлень.

Завдання 8. Опишіть предметну область й розробіть концептуальну схему функціонування середньої школи для ведення звітності проведених занять.

Завдання 9. Опишіть предметну область і розробіть концептуальну схему роботи з фізичними особами філії банку Ощадбанку України.

Завдання 10. Опишіть предметну область і розробіть концептуальну схему функціонування страхової компанії.

Контрольні запитання

1. Що таке база даних?
2. Як визначається предметна область?
3. Розтлумачте поняття “інформація”.

Лабораторна робота № 2
ЕТАПИ РОЗРОБКИ БАЗИ ДАНИХ

Мета роботи: Навчитися визначати сутності та їх набори, а також установлювати зв’язки між ними.

Завдання до роботи:

1. Ознайомитись з поняттям “сутність” і методами роботи з нею.
2. Описати визначену сутність.

ТЕОРЕТИЧНА ЧАСТИНА

При описі предметної області з погляду концептуальної моделі насамперед варто визначити сутності, що належать цій області, і зв'язки між ними. Під *сутністю* у такому підході, розуміється те, про що повинна бути накопичена й оброблена інформація. Наприклад, при розробці схеми функціонування факультету сутностями можуть виступати студенти факультету, викладачі, що викладають дисципліни, методичний і науково-дослідний матеріал, що розробляється факультетом, семінари й конференції, проведені на даному факультеті тощо. Кожна сутність характеризується за допомогою обмеженого набору властивостей та зв'язків з іншими сутностями. Група сутностей, що характеризується тим самим набором властивостей, утворить *набір сутностей.* Так, наприклад, список студентів утворить набір сутностей, що ми назвемо СТУДЕНТ, і він буде характеризуватися наступними властивостями: прізвище, ім'я та по батькові; номер студентського квитка; група; місце проживання; рік народження; наявність чи відсутність стипендії тощо.

Властивості набору сутностей звуться атрибутами, а множину допустимих значень атрибутів називають доменом. З погляду даталогічної моделі, при описі атрибутів кожного з набору сутностей варто вказати не тільки ім'я атрибута, але й тип даних, що описують певний атрибут. Тип даних, який використовується при описі атрибута, залежить від того змісту, що вкладається в цей атрибут при проектуванні моделі об'єктної області. Для прикладу, якщо в наборі об'єктів СТУДЕНТ атрибут “стипендія” характеризує тільки її наявність або відсутність, тобто домен цього атрибута складається всього лише із двох значень, то для

його опису варто використати логічний тип. Якщо ж цей атрибут описує дійсне значення стипендії, то тоді його значення повинно бути числовим або ж грошовим. Якщо ж цей атрибут характеризує тип стипендії, до прикладу: звичайна, підвищена, іменна тощо, то тип даних, що відповідають такому атрибуту, варто задати літерним.

Як приклад розглянемо низку об'єктів, що характеризує працівників деякої фабрики. Як атрибути можна вказати наступне:

ПРАЦІВНИКИ

Назва атрибута	**Тип даних**	**Домен**
Прізвище	Літерний	Сполучення символів-букв
Ім'я	Літерний	Сполучення символів-букв
По батькові	Літерний	Сполучення символів-букв
Номер відділу	Числовий	Будь-яка позитивна ціла цифра
Посада	Літерний	Сполучення символів-букв
Дата народження	Тип дата	Припустимі значення при описі дати
Стаж	Числовий	Будь-яка позитивна ціла цифра
Характеристика	Текст	Будь-який текст
Табельний номер	Числовий	Будь-яка позитивна ціла цифра

Слід зазначити, що в наборі сутностей повинна бути можливість виділити конкретну сутність із набору. Для однозначної ідентифікації конкретної сутності вводиться поняття *ключа*. Ключем може слугувати або конкретний атрибут (простий ключ), або ж певна сукупність атрибутів (складний або складений ключ).

При проектуванні концептуальної моделі першим важливим питанням є вибір набору сутностей, за допомогою якого цілком охоплюється частина предметної області, що нас ціка-

вить. Друге питання – це вибір атрибутів, що підходять для опису цих наборів сутностей. Третє важливе питання – це встановлення зв'язків між наборами сутностей і їхній опис.

Зв'язок між наборами об'єктів буває трьох типів. Перший тип – зв'язок один до одного (позначення 1:1), коли між записами двох наборів сутностей установлюється зв'язок, що характеризується взаємно однозначною відповідністю між сутностями, що входять до кожного з наборів. Скажімо, якщо один із наборів сутностей – це номера проданих на даний рейс квитків, а інший – це список пасажирів, то зв'язок між ними буде один до одного. При порушенні цього принципу повинен видаватися сигнал помилки, тому що на одне й те саме місце буде продано декілька квитків.

Другий тип зв'язку – це один до багатьох (1:M) або зворотний варіант – багато до одного (M:1). Для прикладу, якщо один набір сутностей – це клієнти деякого банку, а інший – рахунки банку, то якщо у клієнта в банку допускається декілька рахунків, то буде встановлений зв'язок один до багатьох. У випадку, коли первинним розглядається рахунок, то зв'язок буде трактуватися як багато до одного.

Третій вид зв'язку – це багато до багатьох (M:N), коли декільком записам одного набору сутностей відповідає кілька записів іншого набору. Як приклад можна розглянути список студентів певного факультету й список предметів, що викладають на цьому факультеті. Зв'язок поміж цими наборами сутностей буде визначатися саме як багато до багатьох, причому він ускладниться, якщо для студентів на факультеті допускається певний вибір предметів, що ними вивчаються.

У реляційній моделі даних сутність, набір сутностей можна інтерпретувати у вигляді таблиць, що називаються *відношеннями* або *реляціями.* У відношеннях стовпці являють собою *атрибути*, і їм привласнюються імена, за якими потім відбувається звертання. *Кортеж,* що відповідає даній схемі відношення, являє собою множину пар {ім'я атрибута, значення}, що містить одне входження кожного імені атрибута, що належить даному відношенню.

В Access відношення відображабться у вигляді таблиць, а зв'язки поміж ними встановлюються за допомогою схеми

відношень. Для встановлення зв'язків між таблицями використовуються зовнішні ключі відношень (*зовнішні ключі* - атрибут або множина атрибутів одного відношення, що є ключем іншого відношення). Зв'язок між двома таблицями виконується шляхом присвоювання значень зовнішнього ключа однієї таблиці значенням ключа іншої таблиці. Так само, як й інші ключі, зовнішні ключі можуть бути як простими, так і складеними.

Завдання

Завдання 1. Опишіть набір сутностей, що задає сукупність студентів заданого факультету. Укажіть перелік атрибутів з указівкою типу даних, що відповідають кожному з них. Для кожного з атрибутів указати домен.

Завдання 2. Опишіть набір сутностей, що задає навчальний план певного факультету. Вважати, що на факультеті можлива спеціалізація за декількома спеціальностями. Наведіть перелік атрибутів із зазначенням типу даних, що відповідатимуть кожному з них. Необхідно вказати також властивість об'єкта, що описується кожним з атрибутів. Для кожного з атрибутів вкажіть домен.

Завдання 3. Опишіть набір сутностей, що описують деякий оптовий склад торговельної фірми. Вважати, що фірма одержує товар від різних постачальників. Укажіть перелік атрибутів із вказівкою типу даних, що відповідає кожному з них. Необхідно вказати також властивість об'єкта, описаного кожним з атрибутів. Для кожного з атрибутів указати домен.

Завдання 4. Опишіть набір сутностей, що описують сукупність товарів певного приватного магазину. Вважати, що магазин одержує товар із різних оптових складів та різних фірм-постачальників. Укажіть перелік атрибутів із вказівкою типу даних, що відповідають кожному з них. Необхідно вказати також властивість об'єкта, описа ного кожним з атрибутів. Для кожного з атрибутів указати домен.

Завдання 5. Опишіть набір сутностей, що задає книжковий фонд певної бібліотеки. Укажіть перелік атрибутів із вказівкою типу даних, що відповідають кожному з них. Не-

обхідно вказати також властивість об'єкта, що описується кожним з атрибутів. Для кожного з атрибутів указати домен.

Завдання 6. Опишіть набір сутностей, що задає список читачів деякої бібліотеки. Укажіть перелік атрибутів із вказівкою типу даних, які відповідатимуть кожному з них. Необхідно вказати також властивість об'єкта, описаного кожним з атрибутів. Для кожного з атрибутів указати домен.

Контрольні запитання

1. Дайте визначення сутності, набору сутностей.
2. Дайте визначення атрибутів.
3. Розтлумачте поняття “ключ”.
4. Які різновиди ключів ви знаєте?
5. Які типи зв’язків ви знаєте?
6. Дайте визначення атрибутів.
7. Дайте визначення відношення, кортежу відношення.
8. Розкрийте зміст поняття “домен”.

Лабораторна робота № 3
ПРОЕКТУВАННЯ БАЗ ДАНИХ

Мета роботи: Знайомство з основними формами баз даних. Створення бази даних.

Завдання до роботи:

1. Ознайомитись з основними умовами функціонування баз даних.
2. Дослідити основні форми баз даних.
3. Створити базу даних.

ТЕОРЕТИЧНА ЧАСТИНА

Умови цілісності даних. Нормалізація даних

Однією з умов нормального функціонування бази даних є поняття цілісності даних, де можна виділити два аспекти:

- категорійна цілісність;
- посилальна цілісність.

Категорійна цілісність пов'язана із тим, що значення атрибутів, які являються ключем відношення, повинні бути унікальними й не можуть бути невизначеними. Тому рядок, що є кортежем, не може бути занесений у відношення доти, допоки не будуть визначені всі атрибути його ключа.

Що стосується *посилальної цілісності*, то якщо в одному відношенні видалити запис, то у відношеннях, пов'язаних з ним, не повинно бути кортежів, які б мали значення зовнішнього ключа запису, що видаляється.

При використанні Access для побудови бази даних відношення трансформуються в таблиці, а при встановленні зв'язків за допомогою схеми даних необхідно подбати про цілісність й каскадне відновлення й видалення даних.

Нормалізація являє собою процес реорганізації,що передбачає виключення повторення тих самих відомостей й інших суперечностей. Остаточна мета нормалізації полягає в одержанні такого проекту бази даних, у якій кожен факт з'являється один раз в одному місці (виключення надмірності інформації), і, крім того, виключення суперечностей у даних, що зберігаються.

Метою нормалізації є звільнення проекту від надмірності даних, аномалій відновлення, аномалій видалення, аномалій уведення.

Основні проблеми, що виникають при роботі з ненормалізованими таблицями:

- надлишковість даних;
- аномалія оновлення;
- аномалія видалення:
- аномалія уведення.

Щоб проілюструвати проблеми, що можуть виникати, розглянемо стіл замовлень певного книжкового складу. Якщо уявити собі таблицю замовлень у вигляді {Код замовлення, прізвище, ім'я, по батькові, адреса, телефон, назви книг, автори, кількість замовлених екземплярів, вартість, дата замовлення}, то легко бачити, що при такій її побудові існує маса недоліків. Для прикладу, якщо замовлено декілька книг, то атрибут назви книг не може бути єдиним (атомарним), що спричинить неатомарність полів автори, кількість замовлений екземплярів та вартість. Якщо ж спробувати зробити його атомарним, то інформація про замовника буде повторюватися стільки ж разів, скільки замовлень він може виконати. Окрім того, інформація про книги може з'явитися вийнятково тоді, коли буде зроблене замовлення. Якщо замовник видаляється з таблиці, то може бути загублена інформація про книги, наявні на складі. Точно в такий самий спосіб може бути загублена інформація про клієнта, якщо деякі назви книг будуть вилучені.

Визначають п'ять видів нормальних форм, однак для успішної роботи цілком достатньо перших трьох, на яких ми й зупинимося.

Перша нормальна форма накладає обмеження на значення полів таблиці – вони повинні бути атомарними. Дана вимога є базовою вимогою класичної реляційної моделі даних. Наприклад, у таблиці зібрана інформація про працівників певної установи.

ПРАЦІВНИКИ

Код	Прізвище	Ім'я	По батькові	Дата народження	Діти	Дата народження дітей
1	Іванов	Іван	Петрович	12.09.63	Ганна Петро	24.05.93 14.02.96
2	Іванова	Ганна	Сергіївна	30.11.69	Ганна Петро	24.05.93 14.02.96
3	Петрова	Інна	Петрівна	23.06.74	Іван	24.05.94
4	Рощин	Сергій	Олегович	20.12.60	Сергій Павло Ірина	12.12.85 18.03.91 24.09.96

Як бачимо, такий опис неодмінно приводить до багатозначності, якщо дітей більше ніж один. Крім того, якщо батьки працюють в одній установі, то інформація про дітей буде повторена двічі. Щоб звільниться від цієї суперечності, таку таблицю варто розбити на дві: таблиця, що містить інформацію про працівників, і таблиця, що містить інформацію про дітей.

ПРАЦІВНИКИ

Код	Прізвища	Ім'я	По батькові	Дата народження
1	Іванов	Іван	Петрович	12.09.63
2	Іванова	Ганна	Сергіївна	30.11.69
3	Петрова	Інна	Петрівна	23.06.74
4	Рощин	Сергій	Олегович	20.12.60

ДІТИ

Код	Прізвище	Ім'я	По батькові	Дата народження
1	Іванов	Петро	Іванович	12.09.63
2	Іванова	Ганна	Іванівна	24.05.93
3	Петров	Іван	Сергійович	24.05.94
4	Рощин	Сергій	Сергійович	12.12.85
5	Рощин	Павло	Сергійович	18.03.91
6	Рощина	Ірина	Сергіївна	24.09.96

Код матері й батька не слід уводити в таблицю «Діти», тому що не обов'язково обидва батьки працюють на даному підприємстві. Для зв'язку цих таблиць краще побудувати окреме відношення, тому що в цьому випадку зазначені суперечності будуть зняті. Дане відношення буде мати два поля: код дітей і код батьків. Воно встановлює зв'язок між батьками й дітьми. Варто врахувати, що при побудові концептуальної схеми необхідно відзначити, що зв'язок між таблицями «Працівники» й «Діти» - «багато до багатьох», тому що працівники можуть мати й більше дітей, і якщо батьки працюють разом, то дитина має двоє батьків у таблиці «Працівники».

Код Працівника	**Код дитини**
1	1
1	2
2	1
2	2
3	3
4	4
4	5
4	6

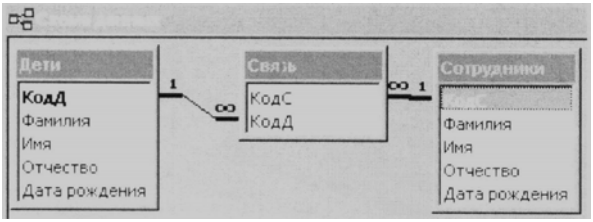


Рис. 1. Схема даних

Access не підтримує типи зв'язків «багато до багатьох», тому введення такого відношення знімає цю проблему. У вікні «Схема даних» цей зв'язок виглядає, як зображено на рис. 1.

Для підтримки цілісності даних при встановленні зв'язків необхідно забезпечити цілісність даних, яка підтримується Access. Вікно, що встановлює забезпечення цілісності даних, зображено на рис. 2.

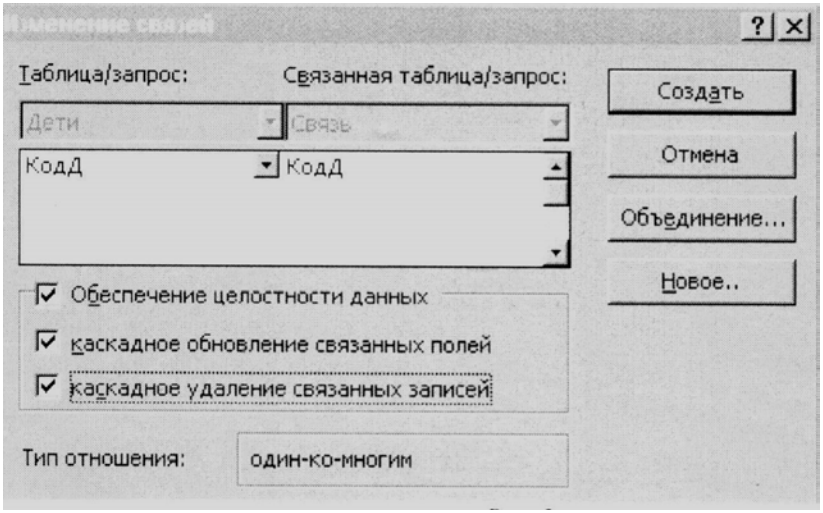


Рис. 2. Зміна зв'язків

Друга нормальна форма. Відношення перебуває в другій нормальній формі в тому й тільки в тому випадку, коли це відношення перебуває в першій нормальній формі, і кожний неключовий атрибут повністю залежить від первинного ключа. Розглянемо репертуар драматичного театру на певний місяць. У таблиці визначені наступні поля {код актора; ПІБ актора; звання; дата народження; назва п'єси; автор; роль; дата постановки}. Ключем є (код актора)+(назва п'єси)+роль.

Задля того, аби перейти від першої нормальної форми до другої, необхідно виконати такі дії:

- Визначити, на які частини можна розбити первинний ключ так, щоб неключові поля залежали тільки від однієї із цих частин.
- Створити нові таблиці для кожної із частин ключа й групи, що залежать від його частин, перемістити в нову таблицю.
- Видалити з вихідної таблиці переміщені поля.

Для прикладу, таблицю можна змінити в такий спосіб: одна таблиця буде визначати особистість актора {код актора; ПІБ актора; звання; дата народження), а друга буде пов'язана з виконуваною роллю, {код ролі; код актора; роль; назва п'єси; автор; дата постановки).

Третя нормальна форма. Розглянемо другу з отриманих вище таблиць. Між полями (назва п'єси)+автор і роль існує функціональна залежність. Дійсно, ми не зможемо внести в таблицю назву п'єси, якщо не відбувся розподіл ролей. Відношення перебуває в третій нормальній формі акурат в тому випадку, коли воно перебуває в другій нормальній формі, і кожен неключовий атрибут нетранзитивно залежить від первинного ключа. Щоб перейти від другої нормальної форми до третьої, необхідно виконати такі дії.

- Визначити всі поля (або групи полів), від котрих залежать інші поля.
- Створити нову таблицю для кожного такого поля (або групи полів) і групи залежних від нього полів та перемістити їх у створену таблицю.
- Видалити переміщені поля з вихідної таблиці.

У розглянутому випадку перетворення другої таблиці можна представити так (рис. 4).

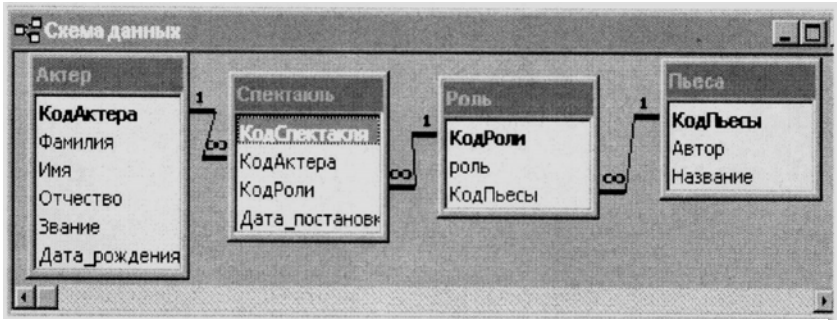


Рис. 3. Схема даних, що перебувають у третій нормальній формі

Завдання

Завдання 1. Побудувати базу даних для ведення замовлень авторемонтної майстерні. Інформація обов'язково має містити відомості про клієнта (ПІП, адреса), тип роботи, оплату та інформацію про виконавця (ПІП, кваліфікація).

Завдання 2. Побудувати базу даних, що описувала б результати сесії. Інформація має вміщувати номер семестру,

відомості про студента (ПІП, група, спеціальність), відомості про предмет, що складали студенти (назва, семестр), дату складання іспиту, оцінку й ПІП екзаменатора.

Завдання 3. Побудувати базу даних, що описує роботу бібліотеки із читачами. Інформація повинна містити відомості про читача (ПІП, адреса, телефон), дані про видану книгу (назва, автор, видавництво) й дату видачі книги.

Завдання 4. Побудувати базу даних, що описує обіг хворих у поліклініці. Інформація має містити відомості про хворих (ПІП, адресу, дату народження), лікаря (ПІП, спеціальність), дату огляду й висновок лікаря.

Завдання 5. Побудувати базу даних, яка описує роботу із замовленнями певної оптової бази. Інформація повинна містити відомості про замовника (Назва фірми, адреса, телефон), відомості про замовлений товар, (найменування, фірма-виробник, рік випуску, вартість одиниці продукції), а також кількість замовленого товару.

Завдання 6. Побудувати базу даних, що описує формування фонду мережі магазинів певної фірми. Інформація повинна містити відомості про магазин (назва, адреса, телефон), та відомості про постачальника (найменування, адреса, телефон), відомості про товар (найменування, кількість) і дату постачання.

Завдання 7. Побудувати базу даних, що описує роботу із клієнтами фірми з технічного обслуговування торговельного устаткування. Інформація повинна збиратися про майстрів, що виконують ремонтні роботи (ПІП, кваліфікація, телефон), про магазини, що подає заявки на ремонт устаткування (найменування встаткування, магазин, адреса, телефон) і про виконання замовлення із вказівкою дати виконання й оплату.

Завдання 8. Побудувати базу даних, що описує репертуарну політику театру. Інформація збирається про акторів (ПІП, звання, дата народження, адреса, телефон), про п'єсу (автори, назва, список ролей із вказівкою їхньої характеристики, тобто вік, амплуа тощо) і про репертуар на наступний місяць із вказівкою дати спектаклю.

Завдання 9. Побудувати базу даних, що описує репертуарну політику філармонії. Інформація збирається про виконавців (ПІП або назва колективу, адреса, телефон, додаткові відомості), про твори,

що виконуються, концертну площадку (назва, характеристика, обсяг), контактний телефон, дату концерту й час його початку.

Завдання 10. Побудувати базу даних, що описує проведення чемпіонату вищої ліги з футболу. Інформація повинна містити відомості про клуб (назва, головний тренер, місце дислокації), футболістів (ПІП, дата народження, номер гравця, спеціалізація), місце проведення матчу (місто, площадка), дату проведення матчу й рахунок.

Завдання 11. Побудувати базу даних, що описує роботу страхової компанії. Інформація повинна містити відомості про компанію (назва, номер реєстрації, ПІП агента, телефон зв'язку), про види страхування, про клієнта (ПІП, адреса, телефон), дату укладання угоди, страхову суму й комісійні.

Завдання 12. Побудувати базу даних, що описує діяльність ремонтної бригади ЖРЕП. Інформація повинна містити відомості про працівників бригади (ПІП, кваліфікація, спеціальність), відомості про замовника (ПІП, адреса, телефон), контактний телефон ЖРЕП, вид ремонту й дату виконання замовлення.

Контрольні запитання

1. Що таке нормалізація даних?
2. Які ви знаєте нормальні форми баз даних?
3. Розтлумачте поняття "цілісність даних".

Лабораторна робота № 4
РОБОТА З БАЗАМИ ДАНИХ ACCESS

Мета роботи: Знайомство з Access. Створення бази даних (БД), створення таблиць, заповнення таблиць інформацією, коригування і перегляд даних.

Завдання до роботи:

1. Ознайомитись із графічним інтерфейсом та основними можливостями програми Access.
2. Створити базу даних (БД).
3. Створити таблицю.
4. Заповнити таблицю інформацією.
5. Відкоригувати та переглянути дані.

ТЕОРЕТИЧНА ЧАСТИНА

У діловій сфері часто доводиться працювати з даними з різноманітних джерел, кожне з яких пов'язане із певним видом діяльності. Для координації всіх цих даних необхідні знання й організаційні навички.

Електронною базою даних (БД) є послідовність даних заданої структури, записана на магнітний диск комп'ютера.

Системи керування базами даних (СКБД) є набором програмних засобів, необхідних для створення, використання й підтримки баз даних.

База даних – це набір даних із наступними властивостями:

- дані логічно пов'язані між собою й несуть відповідну інформацію;
- структура баз даних звичайно відповідає тому специфічному набору даних, які вона містить;
- бази даних відображають тільки окремі аспекти реального світу, що дає змогу визначити їх як "мікросвіт".

СКБД поєднує відомості із різних джерел в одній реляційній базі даних. Форми, запити й звіти, що створюються, дозволяють швидко й ефективно обновлювати дані, отримувати відповіді на запитання, здійснювати пошук потрібних даних, аналізувати дані, друкувати звіти, діаграми та поштові наклейки.

Системи керування даними першого покоління

СКБД першого покоління характерні тим, що кожна група користувачів розробляла своє власне програмне забезпечення для керування даними. Наслідками такого відокремлення стало надмірне дублювання програмних кодів і даних.

Системи керування даними другого покоління

Файли взаємопов'язаних даних об'єднуються в бази даних. СКБД створюються для таких досвідчених користувачів, як програмісти.

Системи керування даними третього покоління

Можливості СКБД розширились. Створені розвинуті інтерфейси, що забезпечують інтерактивний доступ звичайним користувачам.

Переваги СКБД :

- зменшення надлишковості даних;
- без баз даних неможливо уникнути зберігання надлишкових даних;
- при наявності центрального контролю баз даних деякі надлишкові дані можна усунути;
- надлишкові дані не можуть бути повністю усунені, оскільки велику роль у СКБД відіграють питання часу й достовірності.

У світі існує безліч СКБД. Незважаючи на те, що вони можуть по-різному працювати з різними об'єктами і надають користувачу різні функції й засоби, більшість СКБД спираються на єдиний комплекс основних понять. Це дає нам можливість розглянути одну систему й узагальнити її поняття, прийоми й методи на весь клас СКБД. Як такий навчальний об'єкт розглянемо СКБД Microsoft Access, що входить до пакета Microsoft Office. Вона дозволяє розв’язувати широке коло завдань користувачів без програмування і доступна для широкого кола непрофесійних користувачів персональних комп’ютерів.

СКБД Access розроблена для експлуатації в комп’ютерних мережах у середовищі Windows. Одна з основних переваг СКБД Access полягає в тому, що вона має прості та зручні засоби обробки кількох таблиць в одній базі даних. Таблиця є основним

об’єктом бази даних. В одній базі даних зберігається кілька таблиць та засоби зв’язування таблиць.

У системі Access є різні способи керування даними, а саме:

- система меню;
- панелі інструментів;
- контекстне меню;
- укажчик миші;
- комбінації клавіш.

СКБД Access має значну кількість спеціальних програм – майстрів. Є майстер таблиць, майстер кнопок, майстер форм та ін. Майстри здійснюють діалог із користувачем, у процесі якого визначаються дані, необхідні для розв’язування відповідної задачі. Для зручності роботи кожен майстер має певні етапи (кроки). Будь-який етап можна пропустити або звернутись до попередніх.

Формою зображення даних на екрані користувач може керувати. Важливо правильно конструювати форми, оскільки саме з ними працює користувач при введенні й редагуванні записів бази даних. Крім того, форми можна використовувати для збирання та виведення інформації.

ХІД ВИКОНАННЯ РОБОТИ

Створення бази даних

Запустіть програму Microsoft Access. У вікні, що з'явилося, Microsoft Access виберіть *Новая база* даних і клацніть на кнопці ОК. Створити нову базу даних можна також, обравши пункт меню *Файл/Создать базу* даних. У вікні *Файл новой базы данных* виберіть папку, в якій будете поміщати БД, а в нижній частині вікна дайте ім'я файлу **Бібліотека.mdb** (розширення mdb система додасть автоматично). Клацніть на кнопці *Создать*.

Відкривається вікно бази даних, в якому відображені всі компоненти БД:

- **таблиці** - об'єкти, в яких зберігається інформація про якусь предметну область (наприклад роботи бібліотеки, складального цеху ЗТЗ і т.д.). У таблицях інформація подана в стовпчиках, що називаються полями, і в рядках, що називаються

записами. Кожне поле має ім'я, тип, розмір, заголовок, що задаються користувачем при створенні таблиць;

- **запити** - вибірки з декількох таблиць, що відповідають деяким умовам;
- **звіти** - інформація з таблиць, підготовлена для друку;
- **форми** - зображення даних із таблиць на екрані у формі зручної для запровадження, перегляду й коригування інформації;
- **макроси і модулі** - програми обробки даних, що зберігаються в БД на мові VBA.

Створення таблиць

Створимо три таблиці, що містять інформацію про роботу бібліотеки університету:

- **Книги** - містить інформацію про книги, що зберігаються в бібліотеці;
- **ЧитКниги** - містить інформацію про книги, що одержані читачами;
- **Читачі** - містить інформацію про читачів бібліотеки.

Для створення таблиці клацніть на кнопці *Создать* у вікні бази даних на вкладці *Таблицы*. У вікні *Новая таблица* виберіть *Конструктор* і клацніть *ОК*. У вікні конструктора таблиць введіть інформацію, подану на рис. 1. Вам необхідно задати для кожного поля ім'я поля, тип поля, розмір поля (за домовленістю система встановлює розмір поля, що дорівнює 50 позиціям, проте у більшості випадків це занадто великий розмір і його необхідно зменшити). Тип даних вибирається зі списку, що розкривається, якщо натиснути кнопку зі стрілкою. Опис дозволяє більш докладно зазначити призначення поля й особливості інформації, що зберігається в ньому. Заповнювати графу *Описание* не обов'язково. У графі *Формат поля* можна вказувати формат для даних, що вводяться, графа *Подпись* дозволяє задати заголовок поля при виведенні таблиці на екран, якщо заголовок не заданий, то виводиться ім'я поля. У графі *Условие на значение* записують логічні вирази для значень, що вводяться в поле (наприклад, для поля **Вартість** можна поставити **<100**, якщо вартість книги не повинна перевищувати 100 гривень). Графа *Сообщение об ошибке* містить повідомлення користувачу

при введенні помилкових значень. Одне з полів таблиці звичайно призначається ключовим. Значення в цьому полі однозначно визначають запис. Це поле повинно бути призначено *Обязательным* і необхідно зазначити, що це поле є *Индексированным (без повторений).* Таким полем у таблиці **Книги** є поле **Інв№.** Щоб призначити це поле ключовим, позначте поле і клацніть на інструменті *Ключ* (він виділений на рисунку). Закрийте вікно Конструктора таблиць для зберігання структури таблиці та надайте їй ім'я у вікні запиту. Клацніть на кнопці *Открыть* і введіть інформацію про 10 книг. Наприклад, про такі:

Інв№ 1
Шифр 681.3.06 / Г32
Автор книги Гарнаев А.Ю.
Назва книги Самоучитель VBA
Місто Санкт-Петербург
Видавництво БХВ
Рік вид. 1999
Вартість 19
Анотація Цей посібник з мови програмування, що використовується в пакеті Microsoft Office.

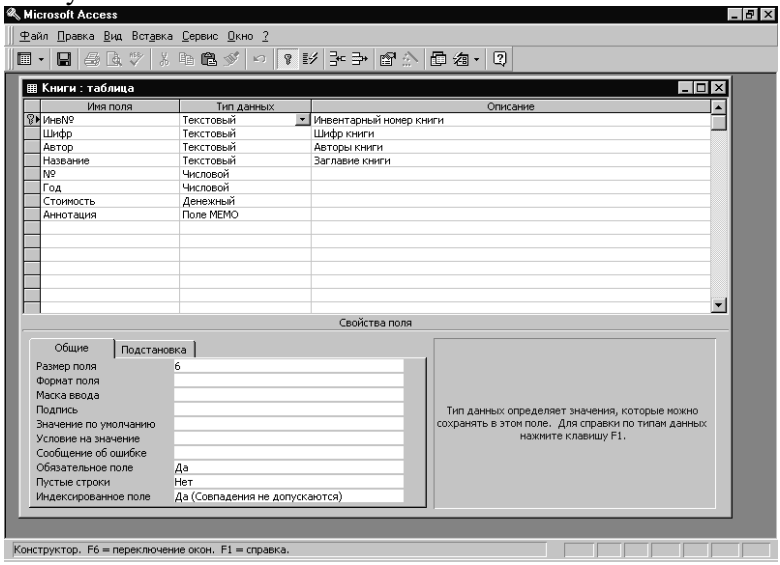


Рис. 1. Створення структури таблиці **Книги**

Інв. № 2
Шифр 618. 3/ К54
Автор книги Камарада, Билл
Назва книги Использование Microsoft Word 97: Пер. с англ.
Місто Москва
Видавництво Издательский дом «Вильямс»
Рік 1998
Вартість 36
Анотація Мета цієї книги – зробити вас кваліфікованим користувачем.

Інв. № 3
Шифр 618. 3/ Х21
Автор книги Дж. Хоникатт и др.
Назва книги Использование Internet: Пер. с англ.
Місто Москва
Видавництво Издательский дом «Вильямс»
Рік 1998
Вартість 24
Анотація У даній книзі наведені найбільш повні дані про Internet.

Створіть нову таблицю **ЧитКниги** з полями:

- **Інв№** - інвентарний номер книги, виданої читачу;
- **NB** - номер читацького квитка читача;
- **Дата видачі** - дата видачі книги читачу;
- **Дата повернення** - дата, коли читач повинен повернути книгу в бібліотеку.

Виберіть відповідні до змісту типи полів і їхні розміри. Поля **Інв№ і NB** повинні бути індексованими й обов'язковими. У цій таблиці можна не призначати ключового поля.

Заповніть таблицю даними про видані книги. Стежте за тим, щоб дані у всіх трьох таблицях були узгоджені, тобто не видавайте книг, котрих немає в бібліотеці.

Створіть нову таблицю **Читачі** з полями:

- **NB** - номер читацького квитка читача;
- **Прізвище** - прізвище читача;
- **Кафедра** - кафедра, на якій працює читач, або група, в якій навчається читач;

- **Телефон** - робочий телефон читача.

Ключовим полем в останній таблиці є поле **NB**, тому що саме воно однозначно визначає кожний запис. Це поле повинно бути індексованим і обов'язковим.

Заповніть і цю таблицю даними. Стежте за тим, щоб не видавати книги неіснуючим читачам, тому що в цьому випадку ви не зможете встановити необхідні зв'язки між таблицями.

Створення зв'язків між таблицями

Проектування нашої бази даних можна вважати завершеним. Залишилося тільки встановити постійні зв'язки між таблицями для того, щоб можна було вибирати дані з кількох таблиць в відповідно до значень збіжних полів. Для цього клацніть на інструменті *Схема данных* та додайте до вікна схеми даних три створені таблиці. На екрані з'явилося схематичне зображення трьох таблиць. Зв'язки між ними встановлюються за допомогою миші за методом «зачепити й перетягнути». Зачепіть поле **Інв№** у таблиці **Книги** й протягніть до такого ж поля в таблиці **ЧитКниги.** На схемі з'явиться лінія, що з'єднує ці поля. Аналогічно встановіть зв'язок двох інших таблиць по полю **NB**.

Створення форм для перегляду і введення даних

Перейдіть на вкладку *Форми* й створіть прості форми для кожної з таблиць. Спробуйте кілька різноманітних типів форм *(простая, ленточная, в столбец)* і виберіть для себе найбільш зручну. Введіть нові дані в таблиці, використовуючи форми.

Створення запитів для відбору даних з однієї або кількох таблиць

Перейдіть на вкладку *Запросы* й створіть такі запити:

- виберіть із таблиці **Книги** книги видавництва «Діалектика», ціна котрих менше 20 гривень;
- виведіть список читачів, що працюють на кафедрі економіки;
- виберіть читачів і книги, що отримані ними в бібліотеці (запит до трьох таблиць);
- створіть запит з ім’ям **Список1**. Цей запит буде використовуватись як джерело даних при створенні підпорядкованої

форми. Для створення запиту **Список1**, з якого буде будуватися підпорядкована форма Список1, перейдіть на вкладку *Запросы* і виберіть кнопку *Создать*. Додайте в запит таблиці **Книги** і **ЧитКниги**. У вікні схеми даних повинен бути показаний зв'язок між таблицями по полю **Інв№** - інвентарний номер книги. Перетягніть мишею в нижню половину вікна поле, що необхідно включити до запиту: **Автор, Назва, Вартість, Інв№, Дата видачі, Дата повернення, NB**-номер читацького квитка. Останнє поле буде потрібно для зв'язку запиту з таблицею Читачі. У першій вільній колонці нижньої частини вікна створіть поле, що обчислюється, з ім'ям **Пеня**. Для цього наберіть у верхньому рядку (де розташовується ім'я поля) такий текст:

Пеня:iif([Дата повернення]<Date(); DateDiff("d";[Дата повернення]; Date())*0,01*[Вартість]; 0)

Імена полів записуються у квадратних скобках.

Побудова запиту завершена. Збережіть його під ім'ям **Список1**. Переконайтеся, що все зроблено правильно, перегляньте запит, клацнувши на кнопці *Открыть*.

Створення форм із підпорядкованою формою

Створимо форму, що містить дані з кількох таблиць. Створимо форму, що для кожного читача виводить список книг, що в нього на руках, з указівкою автора книги, назви книги, дати видачі і дати повернення. Крім того, створимо поле, що обчислюється, - **Пеня**. Пеня нараховується в розмірі одного відсотка від вартості книги за кожний прострочений день. Підрахуємо також загальну суму пені для кожного читача. Форма буде мати підпорядковану форму зі списком книг читача. Дані в підпорядковану форму будуть братися з запиту **Список1**, який вже створено в попередньому пункті.

Початковий макет форми створимо за допомогою *Мастера форм*, а потім поліпшимо його за допомогою *Конструктора*.

Виконайте такі операції для створення форми майстром форм:

1) У вікні бази даних виберіть вкладку *«Формы»* і клацніть на кнопці *Создать*.

2) У вікні *Новая форма* в списку: *«Выберите в качестве источника данних таблицу или запрос»* розкрийте список таб-

лиць і виберіть таблицю **Читачі**. Потім в іншому полі виберіть *Мастер форм* і клацніть на кнопці ОК.

3) З'явилося вікно *Создание форм*. Зі списку *Доступные поля* перенесіть у список *Выбранные поля* поля **NB, Прізвище, Кафедра й Телефон**. Потім в іншому полі розкрийте список таблиць та запитів, виберіть запит **Список1**. Його поля з'являться в списку *Доступные поля*, перенесіть у список *Выбранные поля* всі поля запиту.

4) У наступному вікні необхідно вибрати тип зображення даних. Виберіть *«Читачі»*, тому що головною формою буде форма, що показує дані про читачів. Позначте перемикач *Подчиненные формы*, щоб інші дані були вставлені в підпорядковану форму, і клацніть на кнопці *Далее.*

5) У наступному вікні виберіть вид підпорядкованої форми. Тому що зручніше було б бачити дані про книги, що читаються, подані у вигляді таблиці, позначте перемикач *Ленточний* або *Табличный* і клацніть на кнопці *Далее*.

6) Виберіть стиль для головної форми. Стиль показується у вікні вибору відразу ж, як тільки ви відзначите один із них. Клацніть на кнопці *Далее***.**

7) У наступному вікні необхідно задати імена форм - головної й підпорядкованої. Access створив дві форми, пов'язані одна з одною. Але ви можете коректувати їх у режимі конструктора незалежно один від одного, а також користуватися підпорядкованою формою незалежно від головної. Дайте головній формі ім'я **Читачі**, а підпорядкованій – **Список1**. Клацніть на кнопці *Готово* – побачите на екрані створену форму.

На наступному кроці поліпшимо створену форму за допомогою *Конструктора форм.*

Створимо тепер форму, де не тільки нараховується пеня за прострочені книги, але й обчислюється загальна сума пені для кожного читача:

а) Відкрийте підпорядковану форму **Список1** у режимі конструктора. В області *Примечание формы* створіть нове поле, що обчислюється. Дайте йому ім'я **"Усього пені**" та у вікні *Свойства* для графи *Данные* задайте формулу: =Sum([Пеня]), за якою буде підсумовуватися пеня для кожного читача. Збережіть зроблені зміни й відкрийте форму для перегляду. Створене поле не видно,

тому що підпорядкована форма **Список1** подана в табличному вигляді, що не передбачає зображення яких-небудь інших полів, крім табличних. Проте поле у формі є і обчислення здійснюється.

б) Нарешті, створимо поле, що обчислюється, у формі **Читачі,** що буде показувати нам дані, обчислені в полі **Усього пені** форми **Список1**.

Для цього відкрийте форму **Читачі** в режимі конструктора й додайте елемент керування *Текстове поле* (рис. 2).

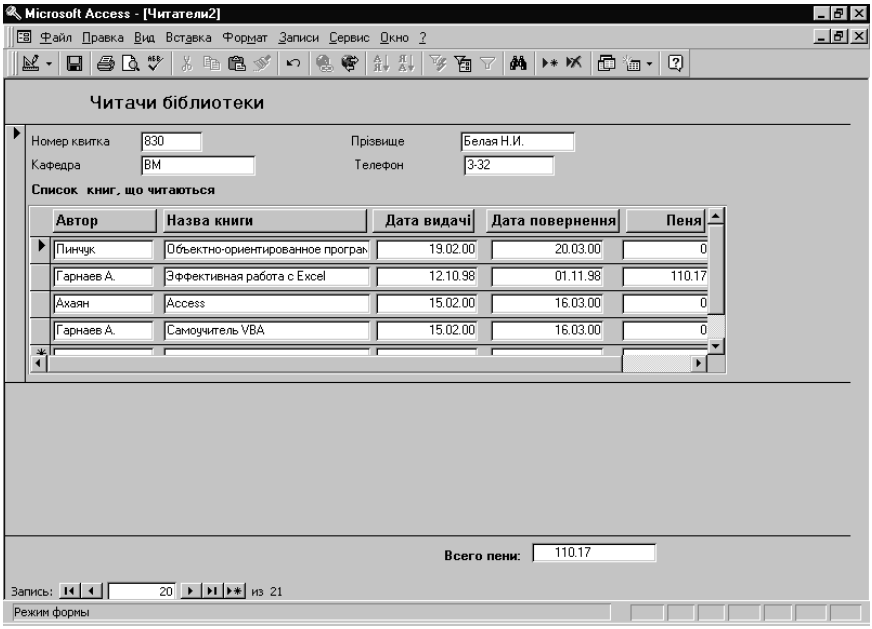


Рис. 2. Створена форма **Читачі**

Позначте створене текстове поле, викличте правою кнопкою миші контекстне меню, виберіть *Свойства* та в полі *Данные* впишіть такий текст:

=[Forms]! [Читачі]![Список1].[Form]! [Усього пені]

Змініть текст перед цим полем, і ваша нова форма готова. Збережіть їх і відкрийте для перегляду.

Створення звітів для виведення даних на принтер. Перейдіть на вкладку *Отчеты* й створіть звіт про наявні в бібліотеці книги, упорядкований за роком видання книг.

Створимо простий звіт, що виводить на принтер список книг, згрупованих по видавництвах, і підраховує загальну вартість книг кожного видавництва.

1. Відкрийте у вікні бази даних закладку *Отчеты* й клацніть на кнопці *Создать*.

2. У наступному вікні виберіть як джерело даних таблицю **Книги** й зазначте, що будете створювати звіт *Мастером отчетов*. Для переходу до наступного кроку клацніть на кнопці *Далее***.**

3. Виберіть поля, що будуть відображені у звіті. Виберіть усі поля таблиці «**Книги**». Зазначимо, що на цьому кроці можна вибирати поля не тільки із зазначеної вище таблиці, але з будь-яких таблиць і запитів поточної бази даних. Для цього у вікні *Таблицы/запросы* відкрийте список таблиць і запитів і виберіть потрібний об'єкт. Список полів обраної таблиці з'явиться у вікні *Доступные поля* та у вас з'явиться можливість перенести потрібні поля у вікно *Отобранные поля*.

4. На цьому кроці необхідно визначити, чи хочете ви групувати дані у звіті за значенням якогось поля. Access часто самий пропонує поле, за якомим виконувати групування. Виберіть поле «Видавництво».

5. Наступний екран пропонує вам вибрати порядок сортування й обчислення, що необхідно виконати для запису. Сортування можна виконувати по чотирьох полях. Виберіть у першому вікні поле **«Шифр»,** а в другому **«Рік видання**». Це означає, що для кожного видавництва книги будуть упорядковані за шифром, а для кожного шифру - за роком видання. Клацніть на кнопці *Результати…*, щоб організувати обчислення підсумкових значень для потрібних полів. Access запропонує вам усі числові поля серед відібраних у звіт. У нашому випадку будуть запропоновані поля **“Рік” і “Вартість**”. Для поля “**Вартість**” позначте прапорці під написами *Sum* і *Avg*, щоб прорахувати сумарну й середню вартості книг для кожного видавництва і по бібліотеці в цілому. Позначте перемикач

Показать данные и результаты й прапорець *Вычислить проценты*, якщо це необхідно.

6. Виберіть вид макета для звіту. Макет показується на екрані й ви можете вибрати його за своїм смаком.

7. Виберіть стиль звіту серед тих, що пропонуються у вікні.

8. Назвіть ваш звіт **«Список книг**» і клацніть на кнопці *Готово***.**

Перегляньте створений звіт і переконайтеся, що він задовольняє всі вимоги.

Створимо звіт, що виводить список читачів бібліотеки, згрупованих по кафедрах, список книг кожного читача, рахується пеня та підводиться по ній результат для кожного читача й по всіх читачах. Для створення такого звіту використовується таблиця **«Читачі»** і запит **«Список1**», створений у попередній роботі, в якому є поле **«Пеня»**, що обчислюється. Звіт буде мати аналогічний формі вигляд, але його можна друкувати. Приклад такого звіту наведений на рисунку.

Для створення звіту виконайте такі дії:

1. Виберіть *Мастер отчетов* і таблицю **Читачі.**
2. Виберіть із таблиці **Читачі** поля **Прізвище, Кафедра й Телефон**. З запиту «**Список1»** усі його поля.
3. Тип зображення даних - *«Читачі*».
4. Додати рівні групування - по полю **Кафедра.**
5. Сортувати по полю **«Дата видачі**», а результати підводити по полю **«Пеня**».
6. Виберіть вид макета й стиль.
7. Дайте звіту ім'я «**Список читачів**» і Готово.
8. Уважно перегляньте створений звіт (Рис. 3).

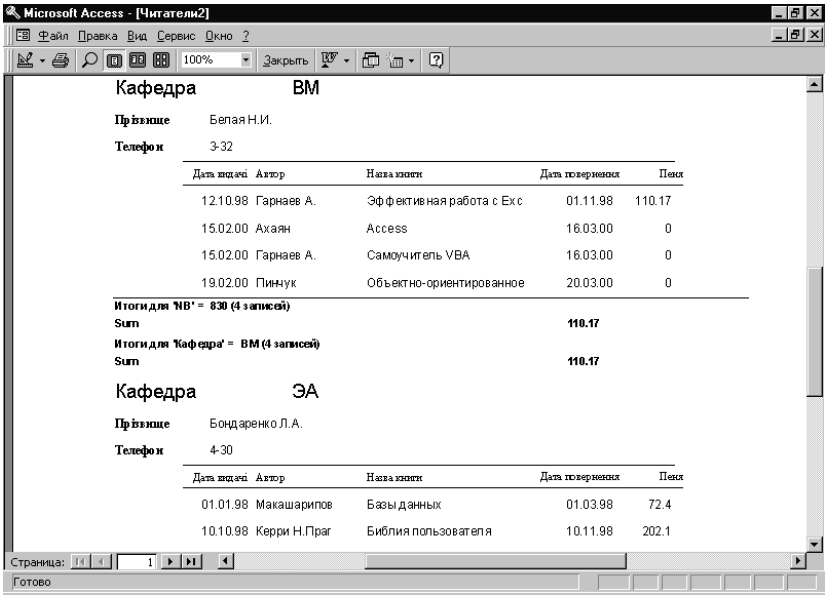


Рис. 3. Звіт про книги, що на руках у читача

Контрольні запитання

1. Яке призначення СКБД?
2. Які ви знаєте можливості графічного інтерфейсу?
3. Поясніть різницю між формою та звітом.

Лабораторна робота № 5
ВИКОРИСТАННЯ БАЗ ДАНИХ MYSQL ПРИ СТВОРЕННІ ВЕБ-РЕСУРСІВ

Мета роботи: Вивчити мову звертання до реляційних баз даних SQL на прикладі бази даних MySQL. Реалізувати керування базою даних MySQL.

Завдання до роботи:

1. Ознайомитись з основними можливостями програми MySQL.
2. Створити базу даних (БД).
3. Створити таблицю.
4. Заповнити таблицю інформацією.
5. Коригувати та переглянути дані.

ТЕОРЕТИЧНА ЧАСТИНА

1. Особливості БД MySQL

MySQL - це популярна система керування базами даних (СКБД), яка дуже часто застосовується разом із PHP.

База даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими: від простого списку майбутніх покупок до переліку експонатів картинної галереї або величезної кількості інформації в корпоративній мережі. Для запису, вибірки й обробки даних, що зберігаються в комп'ютерній базі даних, необхідна система керування базою даних, якою і є програмне забезпечення MySQL. Оскільки комп'ютери чудово справляються з обробкою великих обсягів даних, керування базами даних відіграє центральну роль в обчисленнях. Реалізація подібного керування може бути здійснена по-різному - як у вигляді окремих утилітів, так й у вигляді коду, що входить до складу інших додатків.

У реляційній базі даних дані зберігаються не колективно, а в окремих таблицях, завдяки чому досягається виграш у швидкості й гнучкості. Таблиці зв'язуються між собою за допомогою залежностей, завдяки чому забезпечується можливість поєднувати при виконанні запиту дані з кількох таблиць. SQL як частину системи MySQL можна охарактеризувати як мову

структурованих запитів плюс як найпоширенішу стандартну мову для доступу до баз даних.

Припустимо, що необхідно оформити адресну книгу у вигляді таблиці з рядками й колонками. Кожен рядок (називається також записом) буде відповідати певній особистості; кожна колонка буде містити значення для кожного типу даних: ПІП, телефонні номери й адреси, що є в кожному рядку.

Адресна книга могла б виглядати так:

Ім'я	**Телефон**	**Адреса**
Іванко І.І.	(044)365-8775	03056 Київ
Іванов П.П.	(055)874-3553	02056 Севастополь
Іванчук Б.Б.	(066)976-3665	01056 Львів

Даний двовимірний масив даних є основою реляційної бази даних. Однак реляційні бази даних рідко складаються з однієї таблиці. Часто для БД малого розміру однієї таблиці достатньо, але якщо кількість записів починає сягати сотень тисяч, то для мінімізації ресурсів, необхідних для збереження всіх даних, для зменшення кількості помилок у записах, завжди доцільно створювати кілька таблиць із зовнішніми зв'язками.

2. Встановлення і початок роботи з БД MySQL

БД MySQL не є комерційним продуктом і розповсюджується безкоштовно. Свіжу версію дистрибутиву MySQL для різних операційних систем завжди можна завантажити із сайту розробника www.mysql.org. Є два шляхи встановлення програмного продукту:

- Користувачам із правами адміністратора можна запустити встановлювач setup.exe;
- Завантажити архів із файлами з наступним виконанням з командного рядка команди для встановлення MySQL як системного процесу: *mysqld-max-nt --install*

Зупинка сервісу*: mysql-max-nt --remove*

Універсальним способом запуску серверу БД MySQL є мануальний з командного рядка: *mysqld-max-nt --standalone*

Після чого в новому вікні командного рядка запускаємо діалоговий режим роботи з БД mySQL, як це показано в лістингу 1.

Лістинг 1

```
C:\mysql-5.0.51a-win32\bin>mysql --user=root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.51a-community MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

1.3. Основні команди керування БД MySQL за допомогою мови SQL

Керування БД відбувається в діалоговому режимі: введення команди – повернення повідомлення або про її успішне виконання або про помилки синтаксису мови SQL.

У додатка А наведені основні команди і ключові слова для керування БД MySQL з прикладами. Загалом мова є простою, всі команди мають інтуїтивне значення, наприклад якщо треба створити БД, то команда буде виглядати так:

CREATE DATABASE [ім'я_БД]

У лістингу 2 наведено приклад створення нової БД у діалоговому режимі.

У лістингу 3 наводиться приклад створення таблиці в БД з описом полів, які мають два атрибути - ім'я й тип. Опис типів наводиться в додатку Б. Команда *DESCRIBE* дозволяє проконтролювати правильність створеної таблиці. Внесення записів у таблицю здійснюється командою *INSERT INTO*, як це показано у лістингу 4.

Лістинг 2

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.00 sec)

mysql> CREATE DATABASE firstDB;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| firstDB            |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> USE firstDB;
Database changed

mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Лістинг 3

```
mysql> INSERT INTO students (autoID,name,comment,birth)  VALUES
(NULL,'Ivanov', 'the best', 1985);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM students;
+--------+--------+----------+-------+
| autoID | name   | comment  | birth |
+--------+--------+----------+-------+
|      1 | Ivanov | the best |  1985 |
+--------+--------+----------+-------+
1 row in set (0.01 sec)

  .....

mysql> SELECT * FROM students;
+--------+---------+-------------------+-------+
| autoID | name    | comment           | birth |
+--------+---------+-------------------+-------+
|      1 | Ivanov  | the best          |  1985 |
|      2 | Ivanko  | ivanko@ivanko.ua  |  1985 |
|      3 | Ivanyuk | ivanyuk@gmail.com |  1984 |
+--------+---------+-------------------+-------+
3 rows in set (0.00 sec)
```

Лістинг 4

```
 mysql> CREATE TABLE students (autoID INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT, name VARCHAR(15),comment TEXT,birth SMALLINT);
 Query OK, 0 rows affected (0.09 sec)


 mysql> DESCRIBE students;
 +---------+------------------+------+-----+---------+----------------+
 | Field   | Type             | Null | Key | Default | Extra          |
 +---------+------------------+------+-----+---------+----------------+
 | autoID  | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
 | name    | varchar(15)      | YES  |     | NULL    |                |
 | comment | text             | YES  |     | NULL    |                |
 | birth   | smallint(6)      | YES  |     | NULL    |                |
 +---------+------------------+------+-----+---------+----------------+
 4 rows in set (0.05 sec)
```

ХІД ВИКОНАННЯ РОБОТИ

Створити базу даних за допомогою засобів MySQL згідно з завданням наведеним нижче (номер завдання вибирається згідно з порядкованим номером студента в групі). У межах створеної бази даних реалізувати:

- Створення необхідних таблиць.
- Додавання записів у таблицю.
- Модифікацію записів у таблиці.
- Вибірку даних із таблиці БД.
- Вибірку всіх даних із таблиці.
- Вибірку всіх даних із таблиці, що задовольняють умову.
- Вибірку певних стовпців із даними з таблиці.
- Вибірку унікальних записів із таблиці.
- Сортування.
- Сортування у зворотному порядку.
- Пошук інформації за заданим критерієм.
- Пошук інформації за заданим критерієм із регулярними виразами.
- Лічильник кількості однакових записів у таблиці.
- Вибірку записів одночасно з кількох таблиць.
- Вибірку максимального значення.

- Додавання стовпця до раніше створеної таблиці.
- Видалення стовпця з таблиці.

Завдання

Завдання 1. Побудувати базу даних, що здійснює ведення замовлень авторемонтної майстерні. Інформація повинна містити відомості про клієнта(ПІП, адреса), тип роботи, оплату й інформацію про виконавця (ПІП, кваліфікація).

Завдання 2. Побудувати базу даних, що описує результати сесії. Інформація повинна містити номер семестру, відомості про студента (ПІП, група, спеціальність), відомості про здаваний предмет (назва, семестр), дату складання іспиту, оцінку й ПІП екзаменатора.

Завдання 3. Побудувати базу даних, що описує роботу бібліотеки із читачем. Інформація повинна містити відомості про читача (ПІП, адреса, телефон), інформацію про видану книгу (назва, автор, видавництво) і дату видачі книги.

Завдання 4. Побудувати базу даних, що описує обіг хворих у поліклініці. Інформація повинна містити відомості про хворих (ПІБ, адресу, дату народження), лікаря (ПІП, спеціальність), дату огляду й висновок лікаря.

Завдання 5. Побудувати базу даних, що описує роботу із замовленнями деякої оптової бази. Інформація повинна містити відомості про замовника (Назва фірми, адреса, телефон), відомості про замовлений товар, (найменування, фірма-виробник, рік випуску, вартість одиниці продукції), а також кількість замовленого товару.

Завдання 6. Побудувати базу даних, що описує формування фонду мережі магазинів деякої фірми. Інформація повинна містити відомості про магазин (назва, адреса, телефон), відомості про постачальника (найменування, адреса, телефон) відомості про товар (найменування, кількість) і дату поставки.

Завдання 7. Побудувати базу даних, що описує роботу із клієнтами фірми з технічного обслуговування торговельного устаткування. Інформація повинна збиратися про майстрів, що виконують ремонтні роботи (ПІП, кваліфікація, телефон), про магазини, що подає заявки на ремонт устаткування (найменуван-

ня устаткування, магазин, адреса, телефон) і про виконання замовлення із вказівкою дати виконання й оплату.

Завдання 8. Побудувати базу даних, що описує репертуарну політику театру. Інформація збирається про акторів (ПІП, звання, дата народження, адреса, телефон), про п'єсу (автори, назва, список ролей із вказівкою їхньої характеристики, тобто вік, амплуа тощо) і про репертуар на наступний місяць із вказівкою дати спектаклю.

Завдання 9. Побудувати базу даних, що описує репертуарну політику філармонії. Інформація збирається про виконавців (ПІП або назва колективу, адреса, телефон, додаткові відомості), про твори, що виконуються, концертну площадку (назва, характеристика, обсяг), контактний телефон, дату концерту й час його початку.

Завдання 10. Побудувати базу даних, що описує проведення чемпіонату вищої ліги з футболу. Інформація повинна містити відомості про клуб (назва, головний тренер, місце дислокації), футболістів (ПІП, дата народження, номер гравця, спеціалізація), місце проведення матчу (місто, площадка), дату проведення матчу й рахунок.

Завдання 11. Побудувати базу даних, що описує роботу страхової компанії. Інформація повинна містити відомості про компанію (назва, номер реєстрації, ПІП агента, телефон зв'язку), про види страхування, про клієнта (ПІП, адреса, телефон), дату заключення угоди, страхову суму й комісійні.

Завдання 12. Побудувати базу даних, що описує діяльність ремонтної бригади ЖРЕП. Інформація повинна містити відомості про працівників бригади (ПІП, кваліфікація, спеціальність), відомості про замовника (ПІП, адреса, телефон). контактний телефон ЖРЕП, вид ремонту й дату виконання замовлення.

Контрольні запитання

1. Що таке MySQL?
2. Як створити базу даних в MySQL?
3. Які ви знаєте команди для керування базою даних в MySQL?

Додаток А. Основні функції mySQL

Команда довідки з синтаксису й опис команд і ключових слів SQL: help *команда*; або **?** *команда*; наприклад: mysql> **help** SELECT; або mysql>**?** CREATE;

Вибір бази даних: mysql> **USE** *database*;

Виведення списку раніше створених БД: mysql> **SHOW DATABASES**;

Виведення списку раніше створених таблиць у БД: mysql> **SHOW TABLES**;

Переглянути опис формату таблиці: mysql> **DESCRIBE** *table*;

Створення нової БД: mysql> **CREATE DATABASE** *db_name*;

Створення нової таблиці в БД: mysql> **CREATE TABLE** *table_name* (*field1_name* TYPE(SIZE), *field2_name* TYPE(SIZE)); Приклад: mysql> CREATE TABLE pet (name VARCHAR(20), sex CHAR(1), birth DATE);

Завантаження даних, розділених символами табуляції, в таблицю:

mysql> **LOAD DATA** LOCAL INFILE "infile.txt" INTO TABLE table_name; (Use \n for NULL)

Додавання записів у таблицю: mysql> **INSERT INTO** *table_name* (*column_name1* VARCHAR(20), *column_name2* TEXT, *column_name3* DATE) **VALUES** ('MyName', 'MyOwner', '2002-08-31'); (Use NULL for NULL)

Модифікація записів у таблиці: mysql> **UPDATE** *table* SET *column_name* = "*new_value*" WHERE *record_name* = "*value*";

Вибірка даних із таблиці БД : mysql> **SELECT** *from_columns* **FROM** *table* **WHERE** *умова*;

Вибірка всіх даних із таблиці: mysql> **SELECT * FROM** *table*;

Вибірка всіх даних із таблиці, що задовольняють умову: mysql> SELECT * FROM *table* **WHERE** *rec_name* = "*value*";

Вибірка всіх даних із таблиці, що задовольняють кілька умов: mysql> **SELECT * FROM** *table* **WHERE** *rec1* = "*value1*" **AND** *rec2* = "*value2*";

Вибірка певних стовпців із даними з таблиці: mysql> **SELECT** *column_name* **FROM** *table*;

Вибірка унікальних записів із таблиці: mysql> **SELECT DISTINCT** *column_name* **FROM** *table*;

Сортування: mysql> **SELECT** *col1, col2* **FROM** *table* **ORDER BY** *col2*;

Сортування у зворотному порядку: mysql> **SELECT** *col1, col2* **FROM** *table* **ORDER BY** *col2* **DESC**;

Пошук інформації за заданим критерієм: mysql> **SELECT * FROM** *table* **WHERE** *rec* **LIKE** "*blah%*"; (% - груповий символ, замінює довільне число символів);

Знайти всі 5-символьні записи: SELECT * FROM table **WHERE** rec like "_____"; (_ замінює будь-який один символ).

Пошук інформації за заданим критерієм із регулярними виразами: mysql> **SELECT * FROM** *table* **WHERE** *rec* **RLIKE** "*^b$*";

(. - замінює символ, [...] - замінює клас символів, * - для 0 або більше випадків, ^ - запис починається наступними символами, {n} -повторюється *n* разів, $ - запис закінчується наступними символами).

RLIKE можна замінити еквівалентним ключовим словом **REGEXP**). Для того, щоб у пошуку враховувався регістр символів, використовується ключове слово **BINARY,** наприклад "**REGEXP BINARY**".

Лічильник кількості однакових записів у таблиці: mysql> **SELECT COUNT(*) FROM** *table*;

Групування записів із визначенням кількості записів у кожній групі: mysql> **SELECT** *owner*, **COUNT**(*) **FROM** *table* **GROUP BY** *owner*; (GROUP BY групує разом усі записи для кожного обраного стовпця '*owner*')

Вибірка записів одночасно з кількох таблиць: mysql> **SELECT** pet.name, comment **FROM** pet, event **WHERE** pet.name = event.name;

(Ви можете об'єднувати таблиці, призначаючи нові назви стовпців за допомогою ключового слова '**AS**')

Показує обрану БД: mysql> **SELECT DATABASE()**;

Вибір максимального значення: mysql> **SELECT MAX(*col_name*) AS** *label* **FROM** *table*;

Колонки з автоматичним інкрементуванням чисел у кожному наступному записі: mysql> **CREATE TABLE** table (number INT NOT NULL **AUTO_INCREMENT**, name CHAR(10) NOT NULL);

mysql> INSERT INTO table (name) VALUES ("tom"), ("dick"), ("harry");

Додавання стовпця до раніше створеної таблиці: mysql> **ALTER TABLE** *table* **ADD COLUMN** [*column_create* syntax] **AFTER** *col_name*;

Видалення стовпця з таблиці: mysql> **ALTER TABLE** *table* **DROP** COLUMN *col*;

Створення резервної копії БД mysql: # mysqldump --opt -u username -p database > database_backup.sql (Use 'mysqldump --opt --all-databases > all_backup.sql' to backup everything.)

Більш детальну інформація по синтаксису SQL ви можете знайти на сайті ***mysql.com***.

Додаток Б. Типи даних mySQL:

Тип поля може бути:

- Цілочисловим;
- Дійсним;
- Строковим;
- Бінарним;
- Дата й час;
- Списки й множини.

Можливі типи даних, діапазони й описи подані в наступних таблицях:

Цілочислові типи даних:

Тип	Діапазон
TINYINT	-128... +127
SMALLINT	-32768...+32767
MEDIUMINT	-8 388 608...+8 388 607
INT	-2 147 483 648...+2 147 483 647
BIGINT	-9 223 372 036 854 775 808...+9 223 372 036 854 775 807

Дійсні (реальні) типи записуються у вигляді:

ТИП (ДОВЖИНА, ЗНАКИ) [UNSIGNED]

Довжина - це кількість знакомісць, в яких буде розміщене все число при його передачі, а ЗНАКИ - це кількість знаків після десяткової крапки, які будуть враховуватися. Якщо зазначено модифікатор UNSIGNED, знак числа враховуватися не буде.

Тип	Опис
FLOAT	Невелика точність
DOUBLE	Подвійна точність
REAL	Те ж, що й DOUBLE
DECIMAL	Дробове число, що зберігається у вигляді рядка
NUMERIC	Те ж, що й DECIMAL

Рядкові типи:

Тип	Опис
TINYTEXT	Максимальна довжина 255 символів
TEXT	Максимальна довжина 65535 символів (64 Кб)
CHAR(73);	Максимальна довжина до 255 символів
VARCHAR(30)	Максимальна довжина до 255 символів
MEDIUMTEXT	Максимальна довжина 16 777 215 символів
LONGTEXT	Максимальна довжина 4 294 967 295 символів

Бінарні типи даних:

Тип	Опис
TINYBLOB	Максимум 255 символів
BLOB	Максимум 65535 символів
MEDIUMBLOB	Максимум 16 777 215 символів
LONGBLOB	Максимум 4 294 967 295

Дата й час:

Тип	Опис
DATE	Дата у форматі РР-ММ-ДД
TIME	Час у форматі ГГ:ХХ:СС
TIMESTAMP	Дата й час у форматі timestamp, виводиться у вигляді РРРРММДДГГХХСС
DATETIME	Дата й час у форматі РРРР-ММ-ДД ГГ:ХХ:СС

Лабораторна робота № 6
КЕРУВАННЯ БАЗАМИ ДАНИХ MYSQL ЗА ДОПОМОГОЮ PHP

Мета роботи: Ознайомитись із мовою програмування PHP. Реалізувати віддалене керування БД MySQL засобами PHP скриптів.

Завдання до роботи:

1. Ознайомитись з основними можливостями програми PHP.
2. Створити з'єднання з базою даних.
3. Здійснити додавання даних у базу даних.
4. Відобрази дані з бази в браузері за допомогою мови PHP.
5. Створити SQL-запити до бази даних і обробити отримані відповіді.

ТЕОРЕТИЧНА ЧАСТИНА

1. Вступ у PHP

PHP - це скрипт-мова (scripting language), що вбудовується в HTML, яка інтерпретується та виконується на сервері. Простіше показати це на прикладі:

```
<html>
 <head>
 <title>Example</title>
 </head>
 <body>
 <?php echo "Hi, I'm a PHP script!"; ?>
 </body>
 </html>
```

Після виконання цього скрипту ми одержимо сторінку, в якій буде записано: Hi, I'm a PHP script!

Основна відмінність від CGI-скриптів, написаних на інших мовах, типу Perl або C - це те, що в CGI-програмах ви самі пишете HTML-код, а користуючись PHP - ви вбудовуєте свою програму в готову HTML-сторінку, використовуючи відкриваючий та закриваючий теги (у прикладі це <?php та ?>). Відмінність PHP від JavaScript полягає в тому, що PHP-скрипт

виконується на сервері, а клієнту передається результат роботи, тоді як JavaScript-код цілком передається на клієнтську машину та тільки там виконується.

Можливості PHP

На PHP можна зробити все, що можна зробити за допомогою CGI-програм. Наприклад: обробляти дані з форм, генерувати динамічні сторінки, одержувати та надсилати куки (cookies). Крім цього, в PHP включена підтримка багатьох баз даних (databases), що робить створення Web-аплікацій з використанням БД дуже простим. Ось неповний перелік БД, що підтримуться:

Adabas D InterBase Solid
dBase mSQL Sybase
Empress MySQL Velocis
FilePro Oracle Unix dbm
Informix PostgreSQL

На додаток до всього, PHP розуміє протоколи IMAP, SNMP, NNTP, POP3 та навіть HTTP, а також має можливість працювати із сокетами (sockets) та спілкуватися за іншими протоколами.

Коротка історія PHP

Початком PHP можна вважати осінь 1994р., коли Rasmus Lerdorf вирішив розширити можливості своєї Home-page та написати невеликий «двигун» для виконання найпростіших задач. Такий «двигун» був готовий до початку 1995 року та називався Personal Home Page Tools. Умів він не дуже багато - розумів найпростішу мову та усього кілька макросів.

До середини 1995р. з'явилася друга версія, що називалася PHP/FI Version 2. Приставка FI - приєдналася з іншого пакета Rasmusa, що вмів обробляти форми (Form Interpritator). PHP/FI компілювався всередину Apach'a та використовував стандартний API Apache. PHP-скрипти виявилися швидше аналогічних CGI-скриптів, тому що серверу не було необхідності породжувати новий процес. Мова PHP за можливостями наблизилась до Perl, найпопулярнішої мови для написання CGI-програм. Була додана підтримка багатьох відомих баз даних (наприклад MySQL та Oracle). Інтерфейс до GD-бібліотеки дозволяв генерувати картинки в процесі інтерпретації. З цього моменту почалося поширення PHP/FI.

Наприкінці 1997р. Zeev Suraski та Andi Gutmans вирішили переписати внутрішній двигун, з метою виправити помилки інтерпретатора та підвищити швидкість виконання скриптів. Через півроку, 6 червня 1998р. року вийшла нова версія, що була названа PHP3.

До літа 1999р. року PHP 3 був включений у кілька комерційних продуктів. За даними NetCraft, на листопад 1999р. PHP використовувався в більш ніж 1 млн. доменах. У грудні 1999р. випустили нову версію PHP 4, в якій внутрішній двигун був знову переписаний (він має назву Zend). Продуктивність нової версії стала в десятки разів вище.

Чому треба обирати PHP?

Розробникам Web-аплікацій немає необхідності говорити, що web-сторінки - це не тільки текст та картинки. Гідний уваги сайт повинен підтримувати деякий рівень інтерактивності з користувачем: пошук інформації, продаж продуктів, конференції тощо. Традиційно все це було реалізовано CGI-скриптами, написаними на Perl. Але CGI-крипти дуже погано масштабуються. Кожний новий виклик CGI вимагає від ядра породження нового процесу, а це забирає процесорний час та витрачає оперативну пам'ять. PHP пропонує інший варіант - він працює як частина Web-серверу й цим схожий на ASP від Microsoft.

Синтаксис PHP дуже схожий на синтаксис C або Perl. Люди, знайомі з програмуванням, дуже швидко зможуть почати писати програми на PHP. У цій мові немає строгої типізації даних та немає необхідності в діях із виділення/звільнення пам'яті. Написаний PHP-код легко прочитати та зрозуміти, на відміну від Perl-програм.

2. Взаємодія з базою даних MySQL

До дистрибутиву PHP входить розширення, що містить вбудовані функції для роботи з базою даних MySQL. У цій лабораторній роботі ми познайомимося з деякими основними функціями для роботи з MySQL, що будуть потрібні для розв’язання задач побудови web-інтерфейсів з метою відображення й наповнення бази даних. Такі інтерфейси потрібні для того, щоб вносити інформацію до бази даних і переглядати її вміст могли люди, не знайомі з мовою запитів SQL. При роботі з web-

інтерфейсом для додавання інформації в базу даних людині потрібно просто ввести ці дані в html-форму і відправити їх на сервер, а наш скрипт зробить все інше. А для перегляду вмісту таблиць досить просто клацнути по посиланню й зайти на потрібну сторінку.

Для прикладу розглянемо побудову такого інтерфейсу для таблиці Artifacts, в якій міститься інформація про експонати віртуального музею інформатики. Нехай кожен експонат у колекції Artifacts описується за допомогою наступних характеристик:

назва (title);
автор (author);
опис (description);
альтернативна назва (alternative);
зображення (photo).

Назва й **альтернативна назва** є рядками менш ніж 255 символів довжиною (тобто мають тип VARCHAR(255)), **опис** - текстове поле (має тип TEXT), а в полях **автор** і **зображення** містяться ідентифікатори автора з колекції Persons і зображення експоната з колекції Images відповідно.

Отже, у нас є якась таблиця в базі даних. Щоб побудувати інтерфейс для додавання інформації до цієї таблиці, потрібно її структуру (тобто набір її полів) відобразити в html-форму.

Розіб'ємо цю задачу на наступні підзадачі:

- встановлення з'єднання з БД;
- вибір робочої БД;
- одержання списку полів таблиці;
- відображення полів у html-форму.

Після цього дані, введені до форми, потрібно записати до бази даних. Розглянемо всі ці задачі поступово.

2.1. Встановлення з'єднання із БД

Отже, перше, що потрібно зробити, - це встановити з'єднання з базою даних. Скористаємося функцією mysql_connect. Синтаксис mysql_connect:

mysql_connect ([рядок server [, рядок username [, рядок password [, логічне new_link [, ціле client_flags]]]]])

Дана функція встановлює з'єднання із сервером MySQL і повертає вказівник на це з'єднання або FALSE у випадку невдачі. Для відсутніх параметрів встановлюються наступні значення за замовчуванням:

server = 'localhost:3306'
username = ім'я користувача власника процесу сервера
password = порожній пароль

Якщо функція викликається двічі з тими самими параметрами, то нове з'єднання не встановлюється, а повертається посилання на старе з'єднання. Щоб цього уникнути, використовують параметр new_link, що змушує в будь-якому випадку відкрити ще одне з'єднання.

Параметр client_flags - це комбінація наступних констант: MYSQL_CLIENT_COMPRESS (використовувати протокол стискання), MYSQL_CLIENT_IGNORE_SPACE (дозволяє вставляти пробіли після імен функцій), MYSQL_CLIENT_INTERACTIVE (чекати interactive_timeout секунд - замість wait_timeout - до закриття з'єднання).

Параметр new_link з'явився в PHP 4.2.0, а параметр client_flags - у PHP 4.3.0.

З'єднання із сервером закривається при завершенні виконання скрипта, якщо воно до цього не було закрито за допомогою функції mysql_close().

Отже, установлюємо з'єднання з базою даних на локальному сервері для користувача nina з паролем "123":

```
<?
$conn = mysql_connect("localhost", "nina","123")
or die("Неможливо встановити з'єднання: ". mysql_error());
echo "З'єднання встановлене";
mysql_close($conn);
?>
```

Дія mysql_connect рівносильна команді shell>mysql -u nina -p12314.2. Вибір бази даних.

Після встановлення з'єднання потрібно вибрати базу даних, з якою будемо працювати. Наші дані зберігаються в базі даних book. У MySQL вибір бази даних здійснюється за допомогою команди use:

mysql>use book;

У PHP для цього існує функція mysql_select_db.
Синтаксис mysql_select_db:

логічне mysql_select_db (рядок database_name [, ресурс link_identifier])

Ця функція повертає TRUE у випадку успішного вибору бази даних і FALSE - у протилежному випадку.
Зробимо базу даних book робочою:

```
<?
$conn = mysql_connect("localhost","nina","123")
or die("Неможливо встановити з'єднання: ". mysql_error());
echo "З'єднання встановлене";
mysql_select_db("book");
?>
```

2.2. Одержання списку полів таблиці

Тепер можна зайнятися власне розв’язанням задачі. Як одержати список полів таблиці? Дуже просто. У PHP і на цей випадок є своя команда - mysql_list_fields.
Синтаксис mysql_list_fields:

ресурс mysql_list_fields (рядок database_name, рядок table_name[, ресурс link_identifier])

Ця функція повертає список полів у таблиці table_name у базі даних database_name. Отже, вибирати базу даних нам було необов'язково, але це знадобиться пізніше. Як можна помітити, результат роботи цієї функції - змінна типу ресурс. Тобто це не

зовсім те, що ми хотіли отримати. Це посилання, яке можна використовувати для одержання інформації про поля таблиці, включаючи їх назви, типи і прапори.

Функція mysql_field_name повертає ім'я поля, отриманого в результаті виконання запиту. Функція mysql_field_len повертає довжину поля. Функція mysql_field_type повертає тип поля, а функція mysql_field_flags повертає список прапорів поля, записаних через пробіл. Типи полів можуть бути int, real, string, blob і т.д. Прапори можуть бути not_null, primary_key, unique_key, blob, auto_increment і т.д.

Синтаксис у всіх цих команд однаковий:

рядок mysql_field_name (ресурс result, ціле field_offset)
рядок mysql_field_type (ресурс result, ціле field_offset)
рядок mysql_field_flags (ресурс result, ціле field_offset)
рядок mysql_field_len (ресурс result, ціле field_offset)

Тут result - це ідентифікатор результату запиту (наприклад запиту, відправленого функціями mysql_list_fields або mysql_query (про неї буде розказано пізніше)), а field_offset - порядковий номер поля в результаті.

Функції mysql_list_fields або mysql_query, являють собою таблицю, а точніше, вказівник на неї. Щоб одержати з цієї таблиці конкретні значення, потрібно задіяти спеціальні функції, що порядково читають цю таблицю. До таких функцій і належать mysql_field_name і т.п. Щоб перебрати всі рядки в таблиці результату виконання запиту, потрібно знати кількість рядків у цій таблиці. Команда mysql_num_rows(ресурс result) повертає кількість рядків у безліч результатів result.

А тепер спробуємо одержати список полів таблиці Artifacts (колекція експонатів).

```
<?
$conn = mysql_connect("localhost","nina","123")
or die("Неможливо встановити з'єднання: ".mysql_error());
echo "З'єднання встановлене";
mysql_select_db("book");
```

```
$list_f = mysql_list_fields ("book","Artifacts",$conn);
$n = mysql_num_fields($list_f);
for($i=0; $i<$n; $i++){
  $type = mysql_field_type($list_f, $i);
  $name_f = mysql_field_name($list_f,$i);
  $len = mysql_field_len($list_f, $i);
  $flags_str =  mysql_field_flags ($list_f, $i);
echo "<br>Ім'я поля: ".$name_f;
echo "<br>Тип поля: ".$type;
echo "<br>Довжина поля: ".$len;
echo "<br>Рядок прапорів поля: ".$flags_str . "<hr>"; }
?>
```

У результаті повинно вийти приблизно наступне (якщо в таблиці всього два поля, звичайно):

Ім'я поля: id
Тип поля: int
Довжина поля: 11
Рядок прапорів поля: not_null primary_key auto_increment
Ім'я поля: title
Тип поля: string
Довжина поля: 255
Рядок прапорів поля:

2.3. Відображення списку полів у html-формі

Тепер дещо підкоректуємо попередній приклад. Будемо не просто виводити інформацію про поле, а відображати його як потрібний елемент html-форми. Так, елементи типу BLOB переведемо в textarea (помітимо, що поле description, що ми створювали з типом TEXT, відображається як таке, що має тип BLOB), числа й рядки відобразимо як текстові рядки введення <input type=text>, а елемент, що має позначку автоінкремента, взагалі не будемо відображати, оскільки його значення встановлюється автоматично.

Усе це розв’язується досить просто, за винятком виділення зі списку прапорів прапора auto_increment. Для цього потрібно скористатися функцією explode.

Синтаксис explode:

масив explode(рядок separator, рядок string [, int limit])

Ця функція розбиває рядок string на частини за допомогою роздільника separator і повертає масив отриманих рядків.

У нашому випадку як роздільник потрібно взяти пробіл " ", а як заданий рядок для поділу - рядок прапорів поля.

Отже, створимо форму для введення даних у таблицю Artifacts:

```
<?
$conn=mysql_connect("localhost","nina","123"); // встановлюємо з'єднання
$database = "book";
$table_name = "Artifacts";
mysql_select_db($database); // обираємо базу даних для роботи
$list_f = mysql_list_fields($database,$table_name);
                // отримуємо список полів у базі
$n = mysql_num_fields($list_f);
                // кількість рядків  у результаті попереднього запиту
                //тобто скільки всього полів в таблиці Artifacts)
echo "<form method=post action=insert.php>";
                  // створюємо форму для введення даних
echo "<TABLE BORDER=0 CELLSPACING=0 width=50%><tr>
   <TD  BGCOLOR='#005533' align=center>
    <font color='#FFFFFF'>
      <b> Add new row in $table_name</b>
   </font></td></tr>
   <tr><td></td></tr></TABLE>";
echo "<table border=0 CELLSPACING=1 cellpadding=0 width=50% >";
// для кожного поля отримуємо його ім'я, тип, довжину й прапори
for($i=0;$i<$n; $i++){
   $type = mysql_field_type($list_f, $i);
   $name_f = mysql_field_name ($list_f,$i);
   $len = mysql_field_len($list_f, $i);
   $flags_str = mysql_field_flags ($list_f, $i);
```

```
    // із рядка прапорів робимо масив, де кожен елемент масиву – прапор поля,
        $flags = explode(" ", $flags_str);
        foreach ($flags as $f){
            if ($f == 'auto_increment') $key = $name_f;
                // запам'ятовуємо ім'я інкремента
        }
    /* для кожного поля, що не є автоінкрементом, у
    залежності від його типу виводимо потрібний елемент форми */
    if ($key <> $name_f){
    echo "<tr><td align=right bgcolor='#C2E3B6'>
        <font size=2><b> ". $name_f ."</b></font>
        </td>";
    switch ($type){
          case "string":
              $w = $len/5;
              echo "<td><input type=text name=\"$name_f\" size = $w ></td>";
           break;
           case "int":
              $w = $len/4;
              echo "<td><input type=text name=\"$name_f\" size = $w ></td>";
           break;
           case "blob":
            echo "<td><textarea rows=6 cols=60 name=\"$name_f\"></textarea></td>";
           break;
           }
       }
       echo "</tr>";
    echo "</table>";
    echo "<input type=submit name='add' value='Add'>";
    echo "</form>";
    ?>
```

2.4. Запис даних у базу даних

Отже, форма створена. Тепер потрібно зробити найголовніше - відправити дані з цієї форми до нашої бази даних. Як ви вже знаєте, для того, щоб записати дані в таблицю, використовується команда INSERT мови SQL. Наприклад:

mysql> INSERT INTO Artifacts SET title='Петров';

Скористатись будь-якою командою SQL у PHP-скрипті можна використовуючи функцію mysql_query().

Синтаксис mysql_query:

ресурс mysql_query (рядок query [, ресурс link_identifier])

mysql_query() посилає SQL-запит активній базі даних MySQL серверу, що визначається за допомогою вказівника link_identifier (це посилання на якесь з'єднання із сервером MySQL). Якщо параметр link_identifier опущений, використовується останнє відкрите з'єднання. Якщо відкриті з'єднання відсутні, функція намагається з'єднатися з СКБД аналогічно функції mysql_connect() без параметрів. Результат запиту буферизується.

Тільки для запитів SELECT, SHOW, EXPLAIN, DESCRIBE, mysql_query() повертає вказівник на результат запиту або FALSE, якщо запит не був виконаний. В інших випадках mysql_query() повертає TRUE, якщо запит виконаний успішно, і FALSE - у випадку помилки. Значення, що недорівнює FALSE, говорить про те, що запит був виконаний успішно. Воно не говорить про кількість порушених або повернутих рядків. Цілком можлива ситуація, коли успішний запит не торкнеться жодного рядка mysql_query() також вважається помилковим і поверне FALSE, якщо в користувача недостатньо прав для роботи із зазначеною в запиті таблицею.

Отже, нам відомо, як відправити запит на вставку рядків у базу даних. Зазначимо, що в попередньому прикладі елементи форми ми назвали іменами полів таблиці. Тому вони будуть доступні в скрипті insert.php, що обробляє дані форми як змінні виду $_POST['ім'я_поля'].

```
<?
$conn=mysql_connect("localhost","nina","123");// встановлюємо з'єднання
$database = "book";
$table_name = "Artifacts";
mysql_select_db($database); // обираємо базу даних
```

```
$list_f = mysql_list_fields($database, $table_name);
        // отримуємо список полів у базі
$n = mysql_num_fields($list_f);
          // кількість рядоку результаті попереднього запиту
          // створимо один запит відразу для всіх полів таблиці
$sql = "INSERT INTO $table_name SET ";
   // починаємо створювати запит, перебираємо всі поля таблиці
for($i=0;$i<$n; $i++){
   $name_f = mysql_field_name ($list_f,$i); // визначаємо ім'я поля
   $value = $_POST[$name_f]; // обчислюємо значення поля
   $j = $i + 1;
   $sql = $sql . $name_f." = '$value'";
             // дописуємо в рядок $sql пару ім'я=значення
   if ($j <> $n) $sql = $sql . ", ";
             // якщо поле не останнє в списку, то ставимо кому
}
// перед тим, як записувати що- небудь у базу,  можна подивитися,
//який запит отримається
//echo $sql;
$result = mysql_query($sql, $conn); // відправляємо запит
// виводимо повідомлення, чи успішно виконано запит
if (!result) echo " Can't add ($table_name) ";
   else echo "Success!<br>";
?>
```

Отже, ми навчились додавати дані за допомогою web-інтерфейсу. Тепер спробуємо отримати дані, що зберігаються в базі даних СКБД MySQL.

2.5. Відображення даних, що зберігаються в MySQL

Щоб відобразити якісь дані в браузер за допомогою PHP, потрібно спочатку одержати ці дані у вигляді змінних PHP. При роботі з MySQL без посередника (такого як PHP) вибірка даних робиться за допомогою команди SELECT мови SQL:

```
mysql> SELECT * FROM Artifacts;
```

У попередньому пункті ми бачили, що будь-який запит, у тому числі й на вибірку, можна відправити на сервер за допомогою функції mysql_query(). Там у нас була дещо інша задача - одержати дані з форми й відправити їх за допомогою запиту на вставку в базу даних. Результатом роботи mysql_query() там міг бути лише один із виразів – TRUE або FALSE. Тепер же

потрібно відправити запит на вибірку всіх полів, а результат відобразити в браузері. І тут результат - це ціла таблиця значень, а точніше, вказівник на цю таблицю. Отже, потрібні якісь аналоги функції mysql_field_name(), тільки щоб вони одержували з результату запиту не ім'я, а значення поля. Таких функцій у PHP кілька. Найбільш популярні mysql_result() і mysql_fetch_array().

Синтаксис mysql_result:

змішане mysql_result (ресурс result, ціле row [, змішане field])

mysql_result() повертає значення однієї комірки результату запиту. Аргумент field може бути порядковим номером поля в результаті, ім'ям поля або ім'ям поля з ім'ям таблиці через крапку tablename.fieldname. Якщо для імені поля в запиті застосовувався аліас ('select foo as bar from...'), використовуйте його замість реального імені поля.

Працюючи з великими результатами запитів, варто задіяти одну з функцій, що обробляє відразу цілий ряд результатів (наприклад mysql_fetch_row(), mysql_fetch_array() і т.д.). Тому що ці функції повертають значення кількох комірок відразу, вони НАБАГАТО швидші mysql_result(). Крім того, потрібно врахувати, що вказівка чисельного зсуву (номера поля) працює набагато швидше, ніж вказівка стовпчика або стовпчиків і таблиці через крапку.

Виклики функції mysql_result() не повинні змішуватися з іншими функціями, що працюють із результатом запиту.

Синтаксис mysql_fetch_array:

масив mysql_fetch_array (ресурс result [, ціле result_type])

Ця функція обробляє ряд результатів запиту, повертаючи масив (асоціативний, чисельний або обидва) з обробленим рядом результатів запиту, або FALSE, якщо рядів більше немає.

mysql_fetch_array() - це розширена версія функції mysql_fetch_row(). Крім збереження значень у масиві з чисельними індексами, функція повертає значення в масиві з індексами за назвою стовпчиків.

Якщо декілька стовпчиків у результаті будуть мати однакові назви, буде повернутий останній стовпчик. Щоб одержати доступ до перших, варто використовувати чисельні індекси

масиву або аліаси в запиті. У випадку аліасів ви не зможете використовувати дійсні імена стовпчиків, як, наприклад, не зможете використовувати "photo" в описаному нижче прикладі.

select Artifacts.photo as art_image, Persons.photo as pers_image from Artifacts, Persons

Важливо зазначити, що mysql_fetch_array() працює НЕ повільніше, ніж mysql_fetch_row(), і надає більш зручний доступ до даних.

Другий опційний аргумент result_type у функції mysql_fetch_array() є константою й може набувати наступних значень: MYSQL_ASSOC, MYSQL_NUM і MYSQL_BOTH. Ця можливість додана в PHP 3.0.7. Значенням за замовчуванням є: MYSQL_BOTH.

Використовуючи MYSQL_BOTH, одержимо масив, що складається як з асоціативних індексів, так і з чисельних. MYSQL_ASSOC поверне тільки асоціативні відповідності, а MYSQL_NUM - тільки чисельні.

У теоретичні частині ми розглянули дві задачі: додавання даних до бази даних і їх відображення в браузері за допомогою мови PHP. Для цього ми розглянули ряд функцій, що дозволяють відправляти SQL-запити до бази даних і обробляти отримані відповіді. Використовуючи наведену технологію, можна розв’язати цілий ряд схожих задач, таких як задачі зміни і видалення даних, задачі маніпулювання таблицями бази даних (тобто їхнє створення, зміна і знищення) і т.п. Усе це типові задачі, що виникають при розробці систем керування даними, і вміння їх розв’язувати, як і вміння працювати з базами даних у цілому, дуже важливі для web-програміста.

ХІД ВИКОНАННЯ РОБОТИ

Створити базу даних MySQ за допомогою PHP згідно із завданням, наведеним нижче (номер завдання вибирається згідно з порядкованим номером студента групи). У межах створеної бази даних реалізувати:

- з’єднання з базою данихЖ;
- додавання даних до бази даних;

- відображення даних у браузері за допомогою мови PHP;
- SQL-запити до бази даних і обробку отриманих відповідей.

Завдання

Завдання 1. Побудувати базу даних, що обслуговує ведення замовлень авторемонтної майстерні. Інформація повинна містити відомості про клієнта(ПІП, адреса), тип роботи, оплату й інформацію про виконавця (ПІП, кваліфікація).

Завдання 2. Побудувати базу даних, що описує результати сесії. Інформація повинна містити номер семестру, відомості про студента (ПІП, група, спеціальність), відомості про здаваний предмет (назва, семестр), дату складання іспиту, оцінку й ПІП екзаменатора.

Завдання 3. Побудувати базу даних, що описує роботу бібліотеки із читачем. Інформація повинна містити відомості про читача (ПІП, адреса, телефон), інформацію про видану книгу (назва, автор, видавництво) і дату видачі книги.

Завдання 4. Побудувати базу даних, що описує обіг хворих у поліклініку. Інформація повинна містити відомості про хворих (ПІП, адресу, дату народження), лікаря (ПІП, спеціальність), дату огляду й висновок лікаря.

Завдання 5. Побудувати базу даних, що описує роботу із замовленнями деякої оптової бази. Інформація повинна містити відомості про замовника (Назва фірми, адреса, телефон), відомості про замовлений товар, (найменування, фірма- виробник, рік випуску, вартість одиниці продукції), а також кількість замовленого товару.

Завдання 6. Побудувати базу даних, що описує формування фонду мережі магазинів деякої фірми. Інформація повинна містити відомості про магазин (назва, адреса, телефон), відомості про постачальника (найменування, адреса, телефон) відомості про товар (найменування, кількість) і дату поставки.

Завдання 7. Побудувати базу даних, що описує роботу із клієнтами фірми з технічного обслуговування торговельного устаткування. Інформація повинна збиратися про майстрів, що виконують ремонтні роботи (ПІП, кваліфікація, телефон), про

магазини, що подає заявки на ремонт устаткування (найменування устаткування, магазин, адреса, телефон) і про виконання замовлення із вказівкою дати виконання й оплату.

Завдання 8. Побудувати базу даних, що описує репертуарну політику театру. Інформація збирається про акторів (ПІП, звання, дата народження, адреса, телефон), про п'єсу (автори, назва, список ролей із вказівкою їхньої характеристики, тобто вік, амплуа тощо) і про репертуар на наступний місяць із вказівкою дати спектаклю.

Завдання 9. Побудувати базу даних, що описує репертуарну політику філармонії. Інформація збирається про виконавців (ПІП або назва колективу, адреса, телефон, додаткові відомості), про твори, що виконуються, концертну площадку (назва, характеристика, обсяг), контактний телефон, дату концерту й час його початку.

Завдання 10. Побудувати базу даних, що описує проведення чемпіонату вищої ліги з футболу. Інформація повинна містити відомості про клуб (назва, головний тренер, місце дислокації), футболістів (ПІП, дата народження, номер гравця, спеціалізація), місце проведення матчу (місто, площадка), дату проведення матчу й рахунок.

Завдання 11. Побудувати базу даних, що описує роботу страхової компанії. Інформація повинна містити відомості про компанію (назва, номер реєстрації, ПІП агента, телефон зв'язку), про види страхування, про клієнта (ПІП, адреса, телефон), дату укладання угоди, страхову суму й комісійні.

Завдання 12. Побудувати базу даних, що описує діяльність ремонтної бригади ЖРЕП. Інформація повинна містити відомості про працівників бригади (ПІП, кваліфікація, спеціальність), відомості про замовника (ПІП, адреса, телефон), контактний телефон ЖРЕП, вид ремонту й дату виконання замовлення.

Контрольні запитання

1. Що таке PHP?
2. Як створити базу даних в MySQL засобами PHP?
3. Як здійснюється керування базою даних у PHP?
4. Як створюються SQL-запити до бази даних і обробка отриманих відповідей?

ДОДАТОК 1
«Модель сутність-зв’язок»

Представлення даних за допомогою моделі "сутність-зв'язок"

1. Призначення моделі

Перш, ніж приступати до створення системи автоматизованої обробки інформації, розроблювач повинен сформувати поняття про предмети, факти й події, якими буде оперувати дана система. Для того, щоб привести ці поняття до тієї або іншої моделі даних, необхідно замінити їх інформаційними представленнями. Одним з найбільш зручних інструментів уніфікованого представлення даних, незалежно від реалізуючого їх програмного забезпечення, є модель "сутність-зв'язок" (entity - relationship model, ER - model).

Модель "сутність-зв'язок" ґрунтується на певній важливій семантичній інформації про реальний світ і призначена для *логічного* представлення даних. Вона визначає значення даних у контексті їх взаємозв'язку з іншими даними. Важливим для нас є той факт, що з *моделі "сутність-зв'язок"* можуть бути породжені всі існуючі *моделі даних* (ієрархічна, мережева, реляційна, об'єктна), тому вона є найбільш загальною.

Зауваження: Відзначимо, що модель "сутність-зв'язок" не є моделлю даних у тому розумінні, як це було визначено, оскільки не визначає операцій над даними й обмежується описом тільки їх логічної структури.

Модель "сутність-зв'язок" запропонована в 1976 р. Пітером Пін-Шен Ченом, російський переклад його статті 'Модель "сущность-связь" - шаг к единому представлению данных” опублікований у журналі "СКБД" N 3 за 1995 р.

2.Елементи моделі

Будь-який фрагмент предметної області може бути представлений як *множина сутностей*, між якими існує деяка *множина зв'язків*. Дамо визначення:

Сутність (entity) - це об'єкт, що може бути ідентифікований певним способом, що відрізняє його від інших об'єктів. Приклади: *конкретна людина, підприємство, подія й т.ін.*

Набір сутностей (entity set) - множина сутностей одного типу (тих, що володіють однаковими властивостями). Приклади: *усі люди, підприємства, свята й т. і.* Набори сутностей не обов'язково повинні бути непересічними. Наприклад, сутність, що належить до набору ЧОЛОВІКИ, також належить набору ЛЮДИ.

Сутність фактично являє собою множину *атрибутів*, які описують властивості всіх членів даного набору сутностей.

Приклад:

Розглянемо множину працівників певного підприємства. Кожного з них можна описати за допомогою характеристик *табельний номер, ім'я, вік*. Тому, сутність СПІВРОБІТНИК має атрибути ТАБЕЛЬНИЙ_НОМЕР, ІМ'Я, ВІК. Використовуючи нотацію мови Pascal, цей факт можна представити як:

```
type employe = record
        number : string[6];
        name   : string[50];
        age    : integer;
end;
```

Надалі для визначення сутності та її атрибутів будемо використовувати позначення виду

СПІВРОБІТНИК (ТАБЕЛЬНИЙ_НОМЕР, ІМ'Я, ВІК).

Наприклад, відділи з яких складається підприємство, і в яких працюють співробітники, можна описати як ВІДДІЛ (НОМЕР_ВІДДІЛУ, НАЙМЕНУВАННЯ).

Множина значень (область визначення) атрибута називається *доменом*. Наприклад, для атрибута ВІК домен (назвемо його КІЛЬКІСТЬ_РОКІВ) задається інтервалом цілих чисел, більших від нуля, оскільки людей з негативним віком не буває.

У згаданій статті П. Чена атрибут визначається як *функція, що відображає набір сутностей у набір значень або в декартовий добуток наборів значень*. Так, атрибут ВІК здійснює відображення в набір значень (домен) КІЛЬКІСТЬ_РОКІВ. Атрибут ІМ'Я здійснює відображення в декартовий добуток наборів значень ІМ'Я, ПРІЗВИЩЕ й ПО БАТЬКОВІ.

Звідси визначається *ключ сутності* - група атрибутів, така, що відображення набору сутностей у відповідну групу наборів значень, є взаємооднозначним відображенням. Іншими словами: ключ сутності - це один або більше атрибутів, що унікально визначають дану сутність. У нашому прикладі ключем сутності СПІВРОБІТНИК є атрибут ТАБЕЛЬНИЙ_НОМЕР (звичайно, тільки в тому випадку, якщо всі табельні номери на підприємстві унікальні).

Зв'язок (relationship) - це асоціація, встановлена між декількома сутностями. Приклади:

- оскільки кожен співробітник працює в якому-небудь відділі, між сутностями СПІВРОБІТНИК і ВІДДІЛ існує зв'язок "працює в" або ВІДДІЛ-ПРАЦІВНИК;
- оскільки один із працівників відділу є його керівником, то між сутностями СПІВРОБІТНИК і ВІДДІЛ є зв'язок "керує" або ВІДДІЛ-КЕРІВНИК;
- можуть існувати й зв'язки між сутностями одного типу, наприклад, зв'язок БАТЬКО-НАЩАДОК між двома сутностями ЛЮДИНА;

Але в методиці проектування даних є своєрідне правило доброго тону, відповідно до якого сутності позначаються за допомогою іменників, а зв'язки - дієслівними формами. Дане правило, однак, необов'язкове.

На жаль, не існує загальних правил визначення, що вважати сутністю, а що зв'язком. У розглянутому вище прикладі ми поклали, що "керує" - це зв'язок. Але, можна розглядати

сутність "керівник", що має зв'язки, "керує", із сутністю "відділ" і "що є" із сутністю "співробітник".

Зв'язок також може мати атрибути. Наприклад, для зв'язку ВІДДІЛ-ПРАЦІВНИК можна задати атрибут СТАЖ_РОБОТИ_У_ВІДДІЛІ.

Роль сутності у зв'язку - функція, що виконує сутність у даному зв'язку. Наприклад, у зв'язку БАТЬКО-НАЩАДОК сутності ЛЮДИНА можуть мати ролі "батько" й "нащадок". Указування ролей у моделі "сутність-зв'язок" не є обов'язковим і слугує для уточнення семантики зв'язку.

Набір зв'язків (relationship set) - це відношення між *n* (причому *n* не менше 2) сутностями, кожна з яких відноситься до деякого набору сутностей.

Приклад:

сутності		набори сутностей
e1	належить	E1
e2	належить	E2
	. . .	
en	належить	En

тоді [e1,e2,...,en] - набір зв'язків R

Хоча, строго кажучи, поняття "зв'язок" й "набір зв'язків" різні (перше є елементом другого), їх, проте, дуже часто змішують. Тому, ми, не претендуючи на академічну строгість, надалі також будемо часто користуватися термінами "зв'язок" маючи на увазі "набір зв'язків" й "сутність" маючи на увазі "набір сутностей".

У випадку *n=2*, тобто коли зв'язок поєднує дві сутності, він називається бінарний. Доведено, що *n*-арний набір зв'язків (*n>2)*

завжди можна замінити множиною бінарних, оскільки перші ліпше відображають семантику предметної області.

Та кількість сутностей, що може бути асоційована через набір зв'язків з іншою сутністю, називають *ступенем зв'язку*. Розгляд ступенів особливо корисний для бінарних зв'язків. Можуть існувати такі ступені бінарних зв'язків:

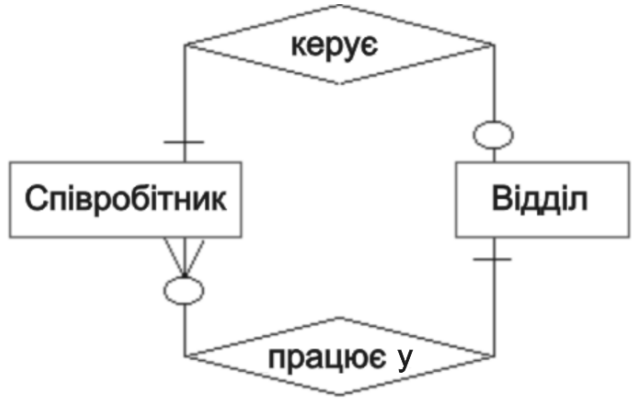
- *один до одного* (позначається *1:1*). Це означає, що в такому зв'язку сутності з однією роллю завжди відповідає не більше однієї сутності з іншою роллю. У розглянутому нами прикладі це зв'язок "керує", оскільки в кожному відділі може бути тільки один начальник, а співробітник може керувати тільки в одному відділі. Даний факт, представлений на рисунку, де прямокутники позначають сутності, а ромб - зв'язок. Оскільки ступінь зв'язку для кожної сутності дорівнює 1, то вони з'єднуються однією лінією.

Іншою важливою характеристикою зв'язку, крім його ступеня, є *клас приналежності* вхідних у нього сутностей або *кардинальність* зв'язку. Оскільки в кожному відділі обов'язково повинен бути керівник, то кожній сутності "ВІДДІЛ" неодмінно повинна відповідати сутність "СПІВРОБІТНИК". Однак не кожен співробітник є керівником відділу, отже, в даному зв'язку не кожна сутність "СПІВРОБІТНИК" має асоційовану з нею сутність "ВІДДІЛ".

Отже, кажуть, що сутність "СПІВРОБІТНИК" має *обов'язковий клас приналежності* (цей факт позначається також указівкою інтервалу числа можливих входжень сутності у зв'язок, у цьому випадку це 1,1), а сутність "ВІДДІЛ" має *необов'язковий клас приналежності* (0,1). Тепер даний зв'язок ми можемо описати як

0,1:1,1. Надалі кардинальність бінарних зв'язків ступеня 1 будемо позначати в такий спосіб:

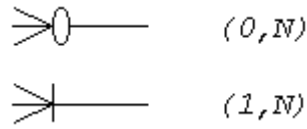
- *один до багатьох* (*1:n*). У цьому випадку сутності з однією роллю може відповідати будь-яка кількість сутностей з іншою роллю. Прикладом слугує зв'язок ВІДДІЛ-СПІВРОБІТНИК. У кожному відділі може працювати довільна кількість співробітників, але співробітник може працювати тільки в одному відділі. Графічно ступінь зв'язку *n* відображається "деревоподібною" лінією, так, як це зроблено на наступному рисунку.



Даний рисунок додатково ілюструє той факт, що між двома сутностями може бути визначено кілька наборів зв'язків.

Тут також необхідно враховувати клас належності сутностей. Кожен співробітник повинен працювати в якомусь відділі, але не кожен відділ (наприклад, знову сформований) повинен включати хоча б одного спів-

робітника. Тому сутність "ВІДДІЛ" має обов'язковий, а сутність "СПІВРОБІТНИК" необов'язковий класи належності. Кардинальність бінарних зв'язків ступеня *n* будемо позначати так:



- *багато до одного* (*n:1*). Цей зв'язок аналогічний відображенню *1:n*. Припустимо, що розглянуте нами підприємство будує свою діяльність на підставі контрактів, які укладаються із замовниками. Цей факт відображається в моделі "сутність-зв'язок" за допомогою зв'язку КОНТРАКТ-ЗАМОВНИК, що поєднує сутності КОНТРАКТ (НОМЕР, СТРОК_ВИКОНАННЯ, СУМА) і ЗАМОВНИК (НАЙМЕНУВАННЯ, АДРЕСА). Оскільки з одним замовником може бути укладене більше одного контракту, то зв'язок КОНТРАКТ-ЗАМОВНИК між цими сутностями буде мати ступінь *n:1*.

У цьому випадку, з очевидних міркувань (кожен контракт, укладений з конкретним замовником, а кожен замовник має хоча б один контракт, інакше він не був би таким), кожна сутність має обов'язковий клас приналежності.

- *багато до багатьох* (*n:n*). У цьому випадку кожна з асоційованих сутностей може бути представлена будь-якою кількістю екземплярів. Нехай на розглянутому нами підприємстві для виконання кожного контракту створюється робоча група, у яку входять співробітники

різних відділів. Оскільки кожен співробітник може входити в декілька (у тому числі й у жодну) робочих груп, а кожна група повинна включати не менше одного співробітника, то зв'язок між сутностями СПІВРОБІТНИК і РОБОЧА_ГРУПА має ступінь *n:n*.

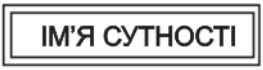
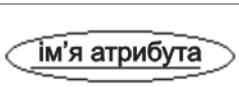
Якщо існування сутності *x* залежить від існування сутності *y*, то *x* називається *залежною сутністю* (іноді сутність *x* називають *"слабкою"*, а сутність *y* – *"сильною"*). Як приклад розглянемо зв'язок між раніше описаними сутностями РОБОЧА_ГРУПА й КОНТРАКТ. Робоча група створюється тільки після того, як буде підписаний контракт із замовником, і припиняє своє існування по виконанню контракту. Отже, сутність РОБОЧА_ГРУПА залежна від сутності КОНТРАКТ. Залежну сутність будемо позначати подвійним прямокутником, а її зв'язок із сильною сутністю – лінією зі стрілкою:

Відзначимо, що кардинальність зв'язку для сильної сутності завжди буде (1,1). Клас приналежності й ступінь зв'язку для залежної сутності можуть бути будь-якими. Припустимо, наприклад, що розглянуте нами підприємство користується декількома банківськими кредитами, які представляються набором сутностей КРЕДИТ (НОМЕР_ДОГОВОРУ, СУМА, СТРОК_ПОГАШЕННЯ, БАНК). За кожним кредитом повинні здійснюватися виплати відсотків і платежі задля його погашення. Цей факт представляється набором сутностей ПЛАТІЖ (ДАТА, СУМА) і набором зв'язків "здійснюється за". У тому випадку, коли одержання запланованого кредиту скасовується, інформація про нього повинна бути вилучена з бази даних.

Відповідно, повинні бути вилучені й усієї відомості про планові платежі за цим кредитом. Отже, сутність ПЛАТІЖ залежить від сутності КРЕДИТ.

3. Діаграма "сутність-зв'язок"

Дуже важливою властивістю моделі "сутність-зв'язок" є те, що вона може бути представлена у вигляді графічної схеми. Це значно полегшує аналіз предметної області. Існує кілька варіантів позначення елементів діаграми "сутність-зв'язок", кожний з яких має свої позитивні риси. Тут будемо використовувати певний гібрид нотацій Чена (позначення сутностей, зв'язків й атрибутів) і Мартіна (позначення ступенів і кардинальностей зв'язків). Наведемо список позначень.

Позначення	**Значення**
	Набір незалежних сутностей
	Набір залежних сутностей
	Атрибут
	Ключовий атрибут
	Набір зв'язків

Атрибути із сутностями й сутності зі зв'язками з'єднуються прямими лініями. При цьому для вказівки кардинальностей зв'язків використовуються позначення, уведені напередодні.

У процесі побудови діаграми можна виділити кілька очевидних етапів:

1. Ідентифікація сутностей і зв’язків, що представляють інтерес;
2. Ідентифікація семантичної інформації в наборах зв'язків (наприклад, чи є деякий набір зв'язків відображенням *1:n*);
3. Визначення кардинальностей зв'язків;
4. Визначення атрибутів і наборів їх значень (доменів);
5. Організація даних у вигляді відношень "сутність-зв'язок";

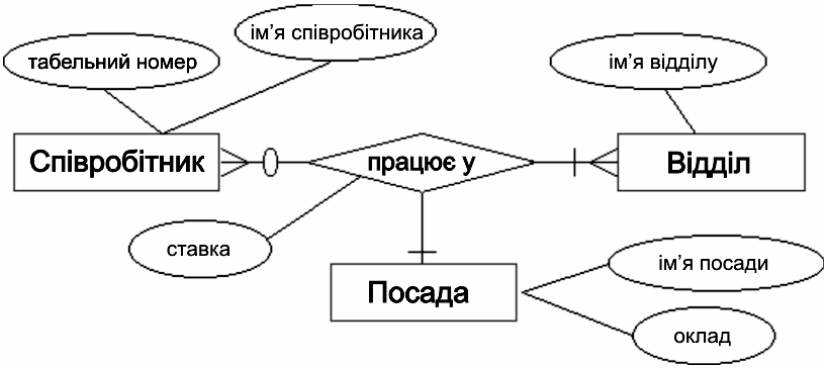
У якості прикладу побудуємо діаграму, що відображає зв'язок даних для підсистеми обліку персоналу підприємства.

Виділимо сутності та зв’язки, що нас цікавлять:

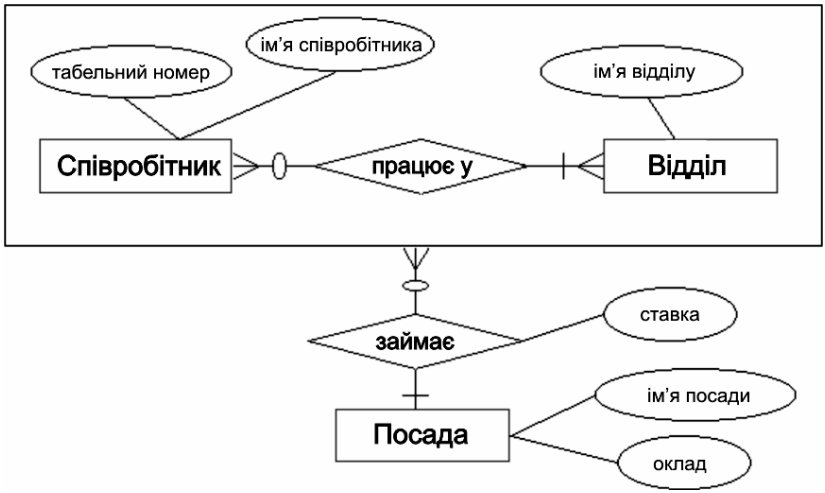
1. Насамперед підприємство складається з відділів, у яких працюють співробітники. Оклад кожного співробітника залежить від посади (інженер, провідний інженер, бухгалтер, прибиральник і т. і.). Далі припустимо, що на нашому підприємстві допускається сумісництво посад, тобто кожний співробітник може мати більш ніж одну посаду (і працювати більш ніж в одному відділі), причому може займати неповну ставку. У той самий час, ту саму посаду можуть займати одночасно декілька співробітників. У результаті цих міркувань ми повинні увести набори сутностей
   - ВІДДІЛ (ІМ'Я_ВІДДІЛУ),
   - СПІВРОБІТНИК (ТАБЕЛЬНИЙ_НОМЕР, ІМ'Я),
   - ПОСАДА (ІМ'Я_ПОСАДИ, ОКЛАД),

   і набір зв'язків ПРАЦЮЄ_У з атрибутом *ставка* між ними. Атрибут *ставка* може приймати значення з інтервалу (0,1] (більше нуля, але менше або дорівнює одиниці), він визначає яку частину посадового окладу одержує даний співробітник.

Як ми вже відзначали вище, кожен *n*-арний набір зв'язків можна замінити декількома бінарними наборами. Тепер саме представляється зручний випадок, щоб оцінити переваги кожного із цих способів представлення зв'язків.



- Тернарний зв'язок, показаний тут, безумовно, несе більш повну інформацію про предметну область. Справді, він однозначно відображає той факт, що оклад співробітника залежить від його посади, відділу, де він працює, і ставки. Однак у цьому випадку виникають деякі проблеми з визначенням ступеня зв'язку. Хоча, як було сказано, кожен працівник може займати кілька посад, а в штаті кожного відділу існують вакансії з різними посадами, проте клас належності сутності ПОСАДА на наведеному рисунку встановлений як (1,1). Це пояснюється тим, що ПОСАДА асоціюється фактично не із сутностями СПІВРОБІТНИК і ВІДДІЛ, а зі зв'язком між ними. Позначати цей факт пропонується так, як це показано на наступній діаграмі:

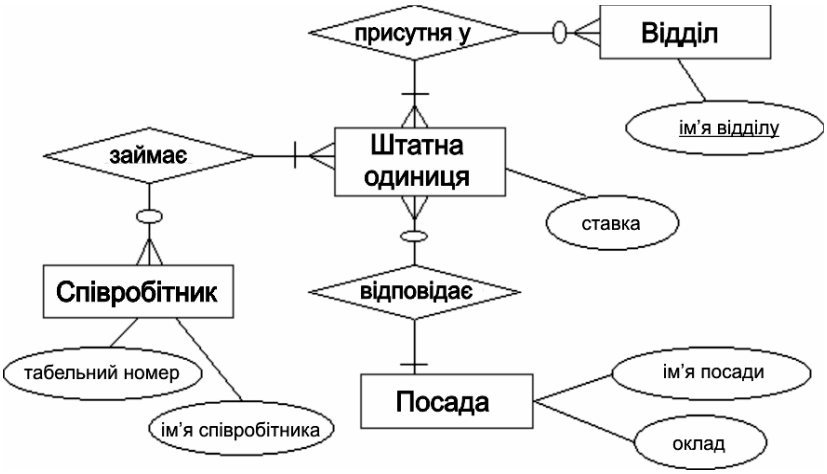


Тут сутності СПІВРОБІТНИК, ВІДДІЛ і зв'язок ПРАЦЮЄ_У агрегуються в якусь нову абстрактну сутність, що асоціюється із сутністю ПОСАДА за допомогою зв'язку ступеня *n:1*. (Це позначення запозичене із книги Silberschatz, "Korth and Sudarshan Database System Concepts", 1997).

- Спробуємо відобразити асоціації співробітників, відділів і посад за допомогою бінарних зв'язків.

У цьому випадку для адекватного опису семантики предметної області необхідно увести ще одну сутність ШТАТНА_ОДИНИЦЯ, що фактично заміняє собою зв'язок ПРАЦЮЄ_У в абстрактної сутності й, тому має атрибут *ставка*.

Перехід від *n*-арного зв'язку за допомогою агрегації сутностей до набору бінарних зв'язків можна розглядати як послідовні етапи одного процесу, що призводить до однозначного породження реляційної моделі даних. При побудові діаграми "сутність-зв'язок" можна використати будь-який із цих трьох способів подання даних.



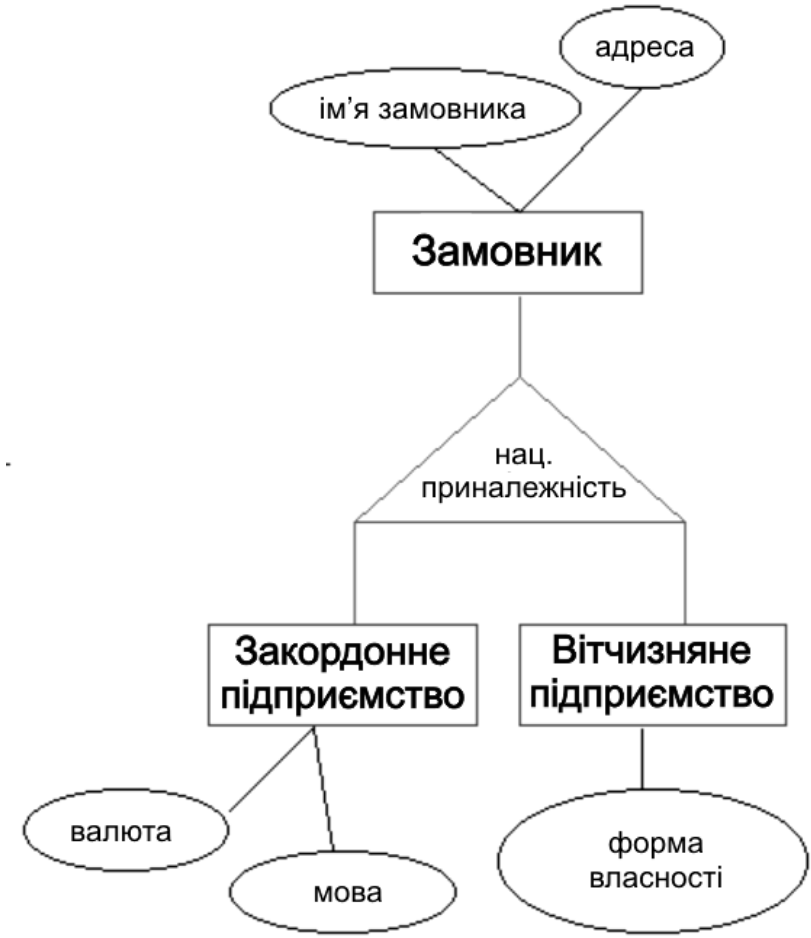
2. Перелічимо ряд об'єктів, які будуть корисні при моделюванні даних розглянутого підприємства. Їм відповідають такі сутності:

   - ЗАМОВНИК (ІМ'Я_ЗАМОВНИКА, АДРЕСА)
   - КОНТРАКТ (НОМЕР, СТРОК_ПОЧАТКУ, СТРОК_ЗАКІНЧЕННЯ, СУМА)
   - РОБОЧА ГРУПА (ВІДСОТОК_ВИНАГОРОДИ)

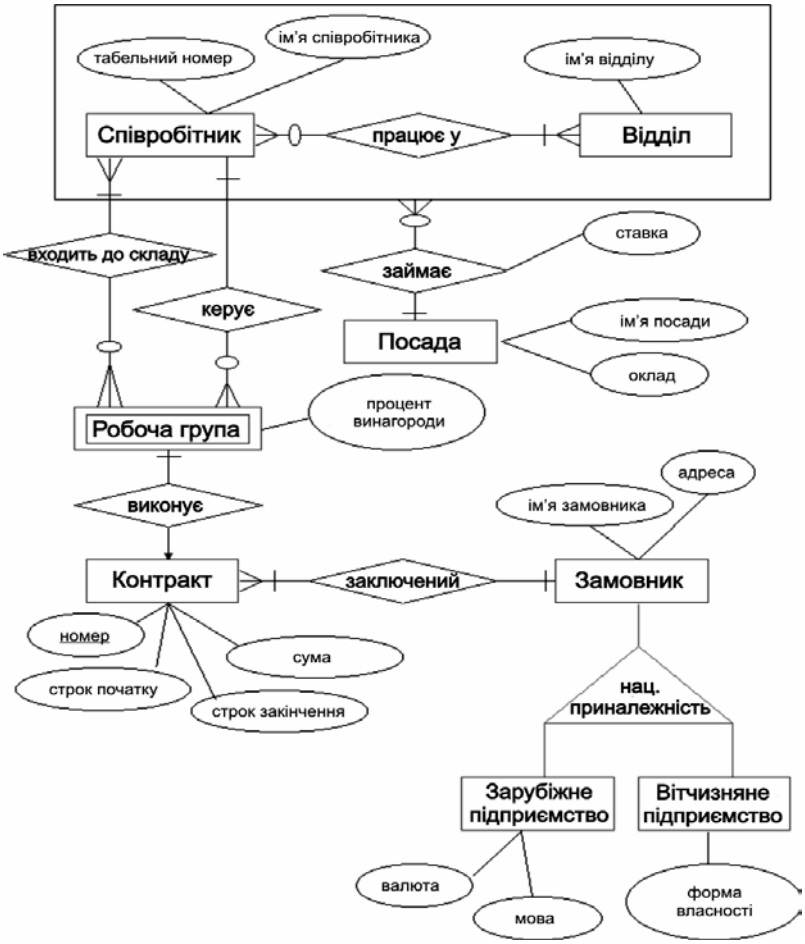
   Атрибут *"відсоток_винагороди"* відображає ту частку вартості контракту, що призначена для оплати праці членів відповідної робочої групи. Зміст інших атрибутів зрозумілий без додаткових пояснень. Зв'язки між перерахованими сутностями були встановлені раніше. Як правило, один із членів робочої групи є керівником по відношенню до інших співробітників, що входять до її складу. Для відображення цього факту ми повинні увести зв'язок "керує" з кардинальністю *1,1:0,n* між сутностями СПІВРОБІТНИК і РОБОЧА_ГРУПА (співробітник може

керувати в довільній кількості робочих груп, але кожна робоча група має одного й тільки одного керівника).

3. Розглянемо тепер більш уважно інформаційний об'єкт "замовник". На практиці дуже часто виникає необхідність розрізняти національну приналежність юридичних осіб, з якими підприємство вступає в договірні відносини. Це пов’язано з тим, що для закордонних фірм необхідно зберігати, наприклад, відомості про валюту, у якій здійснюються розрахунки, мову, якою підписаний контракт і т. і. А для вітчизняних компаній необхідно мати відомості про їхню форму власності (приватна або державна), оскільки від цього може залежати порядок оподатковування коштів, отриманих за виконання робіт за контрактом.
   Отже, ми дійдемо висновку, що необхідно увести в розгляд ще дві непересічні множини ЗАКОРДОННЕ_ПІДПРИЄМСТВО (ВАЛЮТА, МОВА) і ВІТЧИЗНЯНЕ_ПІДПРИЄМСТВО (ФОРМА_ВЛАСНОСТІ), об'єднання яких складає повну множину ЗАМОВНИК. Асоціацію між цими об'єктами називають *відношенням спадкування* або *ієрархічним зв'язком*, тому що сутності ЗАКОРДОННЕ_ПІДПРИЄМСТВО й ВІТЧИЗНЯНЕ_ПІДПРИЄМСТВО успадковують атрибути сутності ЗАМОВНИК (ІМ'Я_ЗАМОВНИКА, АДРЕСА). Щоб визначити до якої підмножини відноситься конкретна сутність з набору ЗАМОВНИК (і, відповідно, який набір атрибутів вона має), необхідно ввести атрибут "національна належність", що називається *дискримінантом*. Цей тип зв'язку пропонується відображати на діаграмі в такий спосіб:



Узагальнюючи всі проведені вище міркування, одержимо діаграму "сутність-зв'язок", показану на наступному рисунку.



На закінчення пропонуємо питання для самостійного опрацювання:

1. Як зміниться діаграма "сутність-зв'язок" у тому випадку, якщо відсоток винагороди по всіх контрактах буде однаковий?
2. Що зміниться в діаграмі, якщо буде заборонене суміснництво посад, тобто кожен співробітник буде мати право займати тільки одну посаду зі ставкою 1?

ДОДАТОК 2
«Операції над даними (реляційна алгебра)»

1.Операції обробки відношень

На вході кожної такої операції використовується одне або кілька відношень, результатом виконання операції завжди є нове відношення.

У розглянутих нижче прикладах будуть використані такі відношення:

P(D1,D2,D3)	Q(D4,D5)	R(M,P,Q,T)	S(A,B)
1 11 x	x 1	x 101 5 a	5 a
2 11 y	x 2	y 105 3 a	10 b
3 11 z	y 1	z 500 9 a	15 c
4 12 x		w 50 1 b	2 d
		w 10 2 b	6 a
		w 300 4 b	1 b

У реляційній алгебрі визначені такі операції обробки відношень:

- ПРОЕКЦІЯ (ВЕРТИКАЛЬНА ПІДМНОЖИНА) Операція проекції представляє із себе вибірку з кожного кортежу відношення значень атрибутів, що входять до списку A, і вилучення з отриманого відношення повторюваних рядків.

Проекція / PROJECT /

Позначення	Визначення	LEAP
$R[A]$	$\{r[A] : r \in R\}$	r = project (R) $(A_1, A_2, \ldots, A_n)$

Приклад:

$$R[M,T] = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \\ w & b \\ w & b \end{bmatrix} = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix}$$

- ВИБІРКА (ОБМЕЖЕННЯ, ГОРИЗОНТАЛЬНА ПІДМНОЖИНА)
  На вході використовується одне відношення, результат - нове відношення, побудоване за тією же схемою, що містить підмножину кортежів вихідного відношення, які задовольняють умову вибірки.

Вибірка / SELECT /

Позначення	Визначення	LEAP
$R[A\,\theta v]$ $R[A_1\,\theta A_2]$	$\{r : r \in R \wedge (r[A]\,\theta v)\}$ $\{r : r \in R \wedge (r[A_1]\,\theta r[A_2])\}$	r = select (R) ((cond) bool (cond))

Приклад:

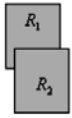
$$P[D_2 = 11] = \begin{bmatrix} 1 & 11 & x \\ 2 & 11 & y \\ 3 & 11 & z \end{bmatrix}$$

- ОБ'ЄДНАННЯ
  Відношення-операнди в цьому випадку повинні бути визначені за однією схемою. Результуюче відношення містить усі рядки операндів, за винятком повторюваних.

Об'єднання / UNION /

Позначення	Визначення	LEAP
$R_1 \cup R_2$	$\{r : r \in R_1 \vee r \in R_2\}$	r = (R1) union (R2)

Приклад:



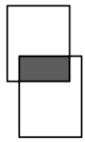
$$R[Q,T] \cup S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cup \begin{bmatrix} \cancel{5} & \cancel{a} \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ \cancel{1} & \cancel{b} \end{bmatrix} = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \end{bmatrix}$$

- ПЕРЕТИН
  На вході операції два відношення, визначені за однією схемою. На виході - відношення, що містить кортежі, які присутні в обох вихідних відношеннях.

Перетин / INTERSECT /

Позначення	Визначення	LEAP
$R_1 \cap R_2$	$\{r : r \in R_1 \wedge r \in R_2\}$	r = (R1) intersect (R2)

Приклад:



$$R[Q,T] \cap S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cap \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix}$$

- РІЗНИЦЯ
  Операція багато в чому схожа на ПЕРЕТИН, за винятком того, що в результуючому відношенні містяться кортежі, що присутні у першому й відсутні у другому вихідних відношеннях.

Різниця / SET DIFFERENCE /

Позначення	Визначення	LEAP
$R_1 - R_2$	$\{r : r \in R_1 \wedge r \notin R_2\}$	r = (R1) difference (R2) r = (R1) minus (R2)

Приклад:

$$R[Q,T] - S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cap \begin{bmatrix} 5 & a \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} 3 & a \\ 9 & a \\ 2 & b \\ 4 & b \end{bmatrix}$$

- ДЕКАРТОВИЙ ДОБУТОК

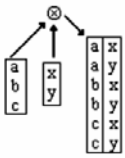
Вхідні відношення можуть бути визначені за різними схемами. Схема результуючого відношення включає всі атрибути вихідних. Крім того:

- ступінь результуючого відношення дорівнює сумі ступенів вихідних відносин
- потужність результуючого відношення дорівнює добутку потужностей вихідних відносин.

Декартовий добуток / CARTESIAN PRODUCT /

Позначення	Визначення	LEAP
$R_1 \otimes R_2$	$\{(r_1 \| r_2) : r_1 \in R_1 \wedge r_2 \in R_2\}$	r = (R1) product (R2)

Приклад:



$$R[M,T] \otimes (R[Q,T] \cap S) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix} \otimes \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} x & a & 5 & a \\ x & a & 1 & b \\ y & a & 5 & a \\ y & a & 1 & b \\ z & a & 5 & a \\ z & a & 1 & b \\ w & b & 5 & a \\ w & b & 1 & b \end{bmatrix}$$

- З'ЄДНАННЯ
  Дана операція має подібність із ДЕКАРТОВИМ ДОБУТКОМ. Однак тут додана умова, відповідно до якого замість повного добутку всіх рядків у результуюче відношення включаються тільки рядки, що задовольняють визначеному співвідношенню між атрибутами з'єднання (A_1,A_2) відповідних відношень.

З'єднання / JOIN /

Позначення	Визначення
$R_1[A_1 \theta A_2] R_2$	$\{(r_1 \| r_2) : r_1 \in R_1 \wedge r_2 \in R_2 \wedge (r_1[A_1] \theta r_2[A_2])\}$

LEAP : r = join (R1) (R2) ((cond) bool (cond))

Приклад:

$$P[D_3 = D_4] Q = \begin{bmatrix} 1 & 11 & x & x & 1 \\ 1 & 11 & x & x & 2 \\ 2 & 11 & y & y & 1 \\ 4 & 12 & x & x & 1 \\ 4 & 12 & x & x & 2 \end{bmatrix} = \begin{bmatrix} 1 & 11 & x & 1 \\ 1 & 11 & x & 2 \\ 2 & 11 & y & 1 \\ 4 & 12 & x & 1 \\ 4 & 12 & x & 2 \end{bmatrix}$$

- ДІЛЕННЯ

Нехай відношення **R** , що називається діленим, містить атрибути $(A_1,A_2,...,A_n)$. Відношення **S** - дільник містить підмножину атрибутів **A**: $(A_1,A_2,...,A_k)$ $(k<n)$. Результуюче відношення **C** визначене на атрибутах відношення **R**, яких немає в **S**, тобто $A_{k+1},A_{k+2},...,A_n$. Кортежі включаються в результуюче відношення **C** тільки в тому випадку, якщо його декартовий добуток з відношенням **S** міститься в діленому **R**.

Ділення / DIVISION /

Позначення	Визначення	LEAP:
$R_1[A_1 \div A_2]R_2$	$\{r[A_1] : r \in R_1 \wedge R_2[A_2] \subseteq g_R(r[A_1])\}$	не підтримується

Приклад:

нехай

$$R_1(A_1, A_2) = \begin{bmatrix} a & x \\ a & y \\ a & z \\ b & x \\ c & y \end{bmatrix} \quad R_2(A_3, A_4) = \begin{bmatrix} 1 & x \\ 2 & x \\ 1 & y \end{bmatrix}$$

тоді

$$R_1[A_2 \div A_4] = \begin{bmatrix} a & x \\ a & y \\ a & z \\ b & x \\ c & y \end{bmatrix} \div \begin{bmatrix} x \\ y \end{bmatrix} = [a]$$

Навчальне видання

Методологія інформаційних систем та баз даних: теоретичний і практичний підходи

Навчальний посібник

Укладачі:
Ушенко Ю.О., **Ковальчук** М.Л.,
Гавриляк М.С., **Негрич** А.Л.

Відповідальний за випуск ***Ангельський О.В.***
Літературний редактор ***Макарова О.П.***
Технічний редактор та дизайнер обкладинки ***Цибуляк В.Д.***

Підписано до друку 09.07.2021. Формат 60x84/16.
Папір офсетний. Друк різографічний. Ум.-друк. арк. 13,2.
Обл.-вид. арк. 14,1. Тираж 50. Н-075.
Видавництво та друкарня Чернівецького національного університету
58002, Чернівці, вул. Коцюбинського, 2
e-mail: ruta@chnu.edu.ua
Свідоцтво суб'єкта видавничої справи ДК №981 від 08.04.2002