ФАКУЛЬТЕТ ІНОФОРМАЦІЙНО-КЕРУЮЧИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Кафедра обчислювальної техніки та систем управління

С. Є. Бантюков, О. Є. Пенкіна, О. В. Казанко

ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

Конспект лекцій

Харків – 2017

Бантюков С. Є., Пєнкіна О. Є., Казанко О. В. Інформаційні системи та технології: Конспект лекцій. – Харків: УкрДУЗТ, 2017. – 45 с.

Конспект лекцій розроблено відповідно до програми курсу «Інформаційні системи та технології». До конспекту включено розділи, що необхідні студентам для опанування теоретичних та практичних основ застосування інформаційних систем, зокрема проектування спеціалізованих реляційних баз даних.

Рекомендується для студентів економічного факультету всіх форм навчання, що вивчають дисципліну «Інформаційні системи та технології».

Іл. 15, бібліогр.: 6 назв.

Конспект лекцій розглянуто та рекомендовано до друку на засіданні кафедри обчислювальної техніки та систем управління 27 березня 2017 р., протокол № 8.

Рецензент

проф. В. І. Мойсеєнко

С. Є. Бантюков, О. Є. Пенкіна, О. В. Казанко

ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

Конспект лекцій

Відповідальний за випуск Пєнкіна О.Є.

Редактор Еткало О. О.

Підписано до друку 12.04.17 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 2,0. Тираж 50. Замовлення №

Видавець та виготовлювач Українська державна академія залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.

Свідоцтво суб’єкта видавничої справи ДК № 2874 від 12.06.2007 р.

ЗМІСТ

ВСТУП

Головна мета викладання дисципліни “Інформаційні системи та технології” у стінах нашого університету − це підготувати студентів до свідомого та ефективного застосування інформаційних систем у своїй майбутній фаховій діяльності. При цьому головна увага приділяється всебічному розкриттю поняття автоматизованої інформаційної системи, класифікаціям інформаційних систем, їх характеристикам та напрямку їх розвитку, складу та властивостям їх основних компонентів, питанням практичного використання інформаційних систем у господарській сфері, зокрема в залізничних перевезеннях.

Іншою важливою частиною курсу слід вважати розгляд питань проектування баз даних, оскільки реляційна база даних є головною складовою кожної сучасної автоматизованої інформаційної системи. Після вивчення курсу студенти будуть спроможні самостійно розробляти реляційні бази даних (БД) відповідно до потреб своєї професійної діяльності.

1 ІНФОРМАЦІЙНІ СИСТЕМИ (ІС) І НАУКОВО-ТЕХНІЧНИЙ ПРОГРЕС. РОЛЬ ІС В УПРАВЛІННІ ПІДПРИЄМСТВОМ

Програмне забезпечення (ПЗ) за піввіку свого існування зазнало величезних змін, пройшовши шлях від програм, здатних виконувати тільки найпростіші логічні й арифметичні операції, до складних систем управління підприємством.

У ПЗ завжди можна було виділити два основних напрямки розвитку:

1) виконання обчислень;

2) накопичування й обробка інформації.

Хоча спочатку комп'ютери призначалися головним чином для виконання складних математичних розрахунків, у цей час домінуючим є другий напрямок.

Це пояснюється насамперед тим, що цивільні галузі застосування комп'ютерів набагато більш поширені, ніж наукові або військові, а зниження вартості комп'ютерів зробило їх

доступними для зовсім невеликих підприємств і навіть приватних осіб.

Сьогодні управління підприємством неможливе без засобів обчислювальної техніки. Комп'ютери давно й міцно ввійшли в такі галузі управління, як бухгалтерський облік, управління складом, асортиментами, закупівлями. Однак на благополуччі роботи сучасного підприємства стали негативно відображатися помилки в його управлінні. Кваліфікованість персоналу, інтуїція й особистий досвід керівника вже недостатні для успішної роботи підприємства, потрібно вже більш широке застосування інформаційних технологій у його управлінні. Для прийняття грамотного управлінського рішення в умовах невизначеності й ризику необхідно постійно тримати під контролем різні аспекти фінансово-господарської діяльності. Тому сучасний підхід до управління припускає вкладення засобів в інформаційні технології. І чим більше та значніше підприємство, тим серйозніші мають бути такі вкладення.

Виділяють дві основні причини, які сприяли створенню ІС.

1 Це неспростовний факт, що сучасне суспільство досягло такого стану, коли всі люди, що населяють землю, самі вже не можуть переробляти постійно зростаючі за кількістю потоки інформації.

2 Посилення протиріччя між своєчасністю й вірогідністю інформації, що використовується в керуванні.

Перша причина не має потреби в поясненнях, другу − пояснимо.

У процесі управління завжди задіяний об'єкт управління та орган управління. Орган управління керує об'єктом. Процес управління відбувається протягом часу, при цьому змінюють свої стани й об'єкт управління, й орган управління. Зобразимо зміну цих станів за допомогою часових діаграм (рисунок 1).

Щоб успішно керувати об'єктом управління, в органі управління повинна бути інформація про поточний стан об'єкта управління. Стани самого органа управління теж важливі для процесу управління, але ми не будемо на них загострювати свою увагу.

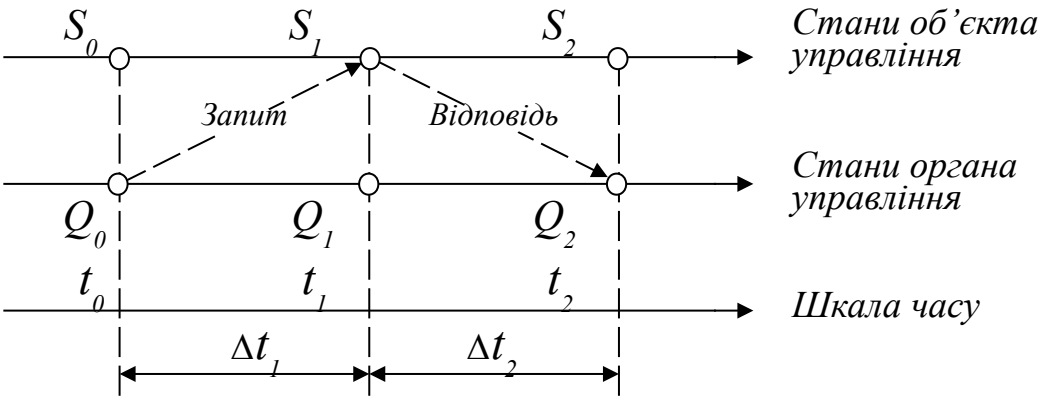

Рисунок 1

У момент часу t_0 орган управління, перебуваючи в стані Q_0, робить запит про поточний стан об'єкта управління (тобто про стан S_0). На підготовку й відсилання запиту потрібен час: зобразимо його на діаграмах у вигляді відрізка Δt_1. Збирання необхідної інформації про об'єкт і зворотне відсилання відповіді на запит органа управління теж триває певний час: на діаграмах він відображається відрізком Δt_2. Таким чином, орган управління в момент t_0 ніяк не буде мати інформацію про поточний стан об'єкта управління S_0, не буде він її мати й у момент одержання відповіді (тобто в момент t_2). Більш того, у момент t_2 орган управління отримає інформацію не про поточний стан об'єкта управління (не про стан S_2), а про зовсім інший і вже застарілий стан S_1. Інакше кажучи, орган управління завжди має несвоєчасну й недостовірну інформацію про стан об'єкта управління. Звісно, щоб підвищити вірогідність своєчасності отримання інформації та її достовірність, треба якось зменшувати Δt_1 і Δt_2. Це можна зробити тільки використовуючи інформаційні системи на базі комп'ютерів.

Однак існує зовнішня перешкода, що обмежує розвиток ІС зверху. Вона виражається в так званому "законі (принципі) необхідної розмаїтості", сформульованому В. Россом Ешбі*. Відповідно до цього закону система, що керує, повинна мати не меншу розмаїтість станів, ніж та система, якою вона керує, щоб мати можливість активного впливу на останню. Інакше кажучи, щоб уміти керувати об'єктом управління й реагувати на всі його

*Вільям Росс Ешбі (William Ross Ashby) народився 6 вересня 1903 р. у Лондоні, Англія. Помер 15 листопада 1972 р. Англійський психіатр, видатний фахівець у галузі кібернетики, основоположник дослідження складних систем.

зміни, орган управління повинен мати таку ж розмаїтість властивостей і можливостей, як і об'єкт управління, тобто мати детальну відповідність своїх властивостей і станів властивостям і станам об'єкта управління. Орган управління повинен містити в собі інформаційну модель об'єкта управління.

2 ТЛУМАЧЕННЯ ТЕРМІНА ІС

Основою більшості фінансово-господарських завдань є обробка інформації. Для полегшення й прискорення обробки інформації створюються інформаційні системи, тобто в широкому розумінні ІС − це будь-яка система обробки інформації (наприклад система використання карток і каталогів у бібліотеках).

Автоматизованими називаються ІС, у яких застосовують технічні пристрої, зокрема ЕОМ. Більшість сучасних ІС є автоматизованими, тому для стислості просто будемо називати їх ІС.

Однак існує й повсюдно використовується більш вузьке трактування терміна ІС, як сукупності апаратно-програмних засобів, задіяних для вирішення деякого прикладного завдання, пов'язаного зі збором, зберіганням і обробкою певних даних (наприклад, бухгалтерського обліку, обліку кадрів, матеріально-технічних засобів і т. д.). Більш того, оскільки зараз апаратною основою майже кожної ІС є обчислювальна система, зібрана на базі стандартних універсальних цифрових ЕОМ, пов'язаних у мережі за допомогою стандартних допоміжних пристроїв, можна розглядати ІС тільки як програмний продукт, тобто як спеціалізоване програмне забезпечення.

На ці три останні тлумачення терміна ІС ми будемо спиратися під час вивчення курсу.

3 РІВНІ ОБРОБКИ ІНФОРМАЦІЇ В ІС

Традиційно при функціонуванні ІС виділяють три рівні оброблюваної інформації:

1) оперативний,

2) тактичний,

3) стратегічний.

Інформація оперативного рівня потрібна в щоденній роботі. Вона первинна, часто обновляється і є основою ієрархії ІС. Її збереження й обробку автоматизують у першу чергу.

Тактичну інформацію отримують шляхом узагальнення інформації оперативного типу. Вона призначається для керівників середньої ланки. Вона являє собою різні варіанти рішень згідно з відповідними запитами.

Стратегічну інформацію одержують на основі оперативної й тактичної інформації. Призначається для керівництва. Вона складається з коротких підсумків, звітів, прогнозів. На її основі здійснюється довгострокове планування й розробка політики підприємства в цілому.

4 МОЖЛИВОСТІ СУЧАСНИХ ІС

ІС сьогоднішніх світових лідерів у галузі розроблення й упровадження корпоративних інформаційних систем (цей термін буде докладно розглянуто пізніше) надають такі можливості:

− видають два-три десятки узагальнених показників, що описують поточний стан бізнесу;

− виконують прогнозування й показують динаміку розвитку бізнесу;

− дають змогу моделювати ситуацію в майбутньому на основі поточних показників;

− містять засоби автоматизації управління;

− дають можливість оцінити ризик для різних варіантів рішень керівника;

− забезпечують можливість роботи керівника в режимі віддаленого доступу.

5 ОСНОВНІ ВЛАСТИВОСТІ ІС

Залежно від предметної області ІС можуть досить значно розрізнятися за своїми функціями, архітектурою, реалізацією. Однак можна виділити ряд властивостей, які є загальними:

1)інформаційні системи призначені для збору, зберігання й обробки інформації, тому в основі кожної з них лежить середовище зберігання й доступу до даних;

2)ІС орієнтовані на кінцевого користувача, що не має високої кваліфікації в галузі обчислювальної техніки. Тому ІС повинні завжди мати простий, зручний та легкий у застосуванні інтерфейс, що надає кінцевому користувачеві всі необхідні для роботи функції й у той самий час не дає користувачу можливості виконувати які-небудь зайві дії.

Таким чином, при розробленні ІС доведеться вирішувати два основних завдання:

1) розроблення бази БД, призначеної для зберігання інформації;

2) розробка графічного інтерфейсу користувача.

6 ГОЛОВНІ ГАЛУЗІ ЗАСТОСУВАННЯ ІС

Розглянемо найбільш важливі *завдання, вирішувані за допомогою ІС*.

1 *Бухгалтерський облік*. Це класична й найбільш часто затребувана на сьогоднішній день галузь застосування інформаційних технологій. Причини: 1) помилка бухгалтера може коштувати дуже дорого, тому очевидна користь автоматизації бухгалтерії; 2) завдання бухгалтерського обліку досить легко формалізується, так що розроблення систем автоматизації бухобліку не є технічно складною проблемою.

2 *Управління фінансовими потоками.* Упровадження інформаційних технологій в управління фінансовими потоками також обумовлено критичністю цієї галузі управління підприємством до помилок. Неправильна побудова системи фінансових розрахунків може спровокувати кризу готівки підприємства. І навпаки, будучи точно переліченою й жорстко

контрольованою ця система може істотно збільшити оборотні кошти підприємства.

3 *Управління складом, асортиментом, закупівлями.* Автоматизація процесів аналізу руху товару, визначення асортименту допомагає одержати максимальний прибуток при обмежених фінансових ресурсах, дає змогу вчасно виявляти "заморожені" оборотні кошти в складському запасі й правильно працювати з номенклатурою, що має найбільш динамічний рух.

4 *Управління виробничим процесом.* Оптимальне управління виробничим процесом є трудомістким завданням, де основним механізмом є планування. Автоматизація дає можливість грамотно планувати, ураховувати витрати, проводити технічну підготовку виробництва, оперативно керувати виробничим процесом відповідно до виробничої програми й технології.

Очевидно, що чим більше виробництво, тим більше виробничих процесів беруть участь у створенні прибутку, а отже, використання ІС у великому виробництві життєво необхідне.

5 *Управління маркетингом.* Мається на увазі збір і аналіз даних про конкурентів: їхню продукцію, цінову політику, а також моделювання параметрів зовнішнього оточення для визначення оптимального рівня цін, прогнозування прибутку й планування рекламних компаній. Ці завдання можуть бути формалізовані і вирішуватися за допомогою ІС.

6 *Документообіг.* Це дуже важливий процес діяльності будь-якого підприємства. Вдало налагоджена система обліку обігу документів відбиває виробничу діяльність, що реально відбувається на підприємстві, і дає керівникам можливість впливати на неї. Тому тут ІС дає змогу підвищити ефективність управління.

7 *Оперативне управління підприємством.* ІС, що вирішують завдання оперативного управління, будуються на основі БД, у якій фіксується вся можлива інформація про підприємство. Така ІС є інструментом для управління бізнесом усього підприємства й містить у собі безліч програмних рішень з автоматизації процесів у виробництві, що наявні на конкретному підприємстві.

7 ВИМОГИ, ЩО ВИСУВАЮТЬСЯ ДО ІС

1 *Гнучкість.* Мається на увазі здатність ІС до адаптації й подальшого розвитку разом з новими умовами роботи й потребами підприємства. Це можливо, якщо на етапі розроблення ІС використовувалися загальноприйняті засоби й методи документування. Це дає змогу згодом розібратися в структурі системи і внести в неї відповідні зміни.

Варто мати на увазі, що психологічно легше розібратися у власних розробках (нехай навіть створених давно), ніж у чужих. Тому рекомендується відразу супроводження системи довіряти особам, які її проектували.

2 *Надійність.* Мається на увазі функціонування ІС без випадків перекручування або втрати інформації. Надійність забезпечується створенням резервних копій збереженої інформації, виконання операцій протоколювання, підтримування якості каналів зв'язку й фізичних носіїв інформації. Сюди ж варто віднести наявність захисту від випадкових втрат інформації внаслідок недостатньої кваліфікації персоналу.

3 *Ефективність.* Система є ефективною, якщо з урахуванням виділених їй ресурсів вона дає змогу вирішувати покладені на неї завдання в мінімальний термін.

Ефективність системи забезпечується оптимізацією даних і методів їх обробки, застосуванням оригінальних розробок, ідей, методів проектування. Крім того, оскільки працювати з ІС доводиться людям, що є фахівцями у своїй предметній області, але маючи середні навички в роботі з комп'ютером, то інтерфейс ІС повинен бути їм інтуїтивно зрозумілим.

4 *Безпека.* Мають на увазі властивості ІС обмежувати доступ особам до інформаційних ресурсів організації, крім тих, які для них призначені. Захист інформації забезпечується управлінням доступом до ресурсів системи, використанням сучасних програмних засобів захисту інформації, застосуванням паролів і протоколюванням, постійним моніторингом стану безпеки операційних систем і засобів їх захисту.

8 ОСНОВНІ ПОНЯТТЯ ТЕОРІЇ ІС

Основні ідеї сучасних інформаційних технологій опираються на концепції баз даних. Відповідно до цього основою інформаційної технології є дані, які організовані в БД з метою відображення реального стану предметної області й задоволення потреб користувача.

Дані — це відомості про деяку сутність, що зафіксовані у вигляді значень і зберігаються на деяких носіях. Дані перетворюються в змістовну інформацію, будучи осмислено обробленими й поданими в потрібний час у потрібне місце.

Предметна область — набір об'єктів, що мають інтерес для актуальних або передбачуваних користувачів, коли реальний світ відображається сукупністю конкретних і абстрактних понять, між якими фіксуються певні зв'язки.

База даних — іменована сукупність взаємозалежних даних, що відображає стан об'єктів і їх відношень у деякій предметній області, використовувана декількома користувачами й зберігається з мінімальною надмірністю.

Система управління базою даних (СУБД) — це комплекс мовних і програмних засобів, призначений для створення, ведення й спільного використання БД багатьма користувачами.

Вище вже розглядалися тлумачення терміна ІС. Наведемо загальне визначення ІС, що, звичайно, дається у відповідній довідковій літературі.

Інформаційна система — це система, що реалізує автоматизований збір, зберігання й обробку даних і містить у собі технічні засоби, програмне забезпечення й відповідний персонал.

Банк даних (БнД) — це основана на технології баз даних система спеціально організованих даних, а також програмних, мовних, організаційних і технічних засобів, призначених для централізованого нагромадження й колективного багатоцільового використання даних.

БнД, по суті, є різновидом ІС, у яких реалізовані функції централізованого зберігання й нагромадження оброблюваної інформації, організованої в одну або кілька баз даних.

Вирішенням проблеми в обробці даних є побудова БД. Однак бази даних можуть відрізнятися в принципах свого

проектування й тому в більшому або меншому ступені відповідати рішенню конкретної проблеми обробки даних.

Головну роль у проектуванні баз даних відіграє вибір *способу організації даних*.

9 МОДЕЛІ ОРГАНІЗАЦІЇ ДАНИХ

Збережені в базі дані мають певну логічну структуру, іншими словами, вони описуються деякою моделлю подання (або організації) даних (моделлю даних), що підтримується СУБД. До числа класичних і таких, що найбільш часто застосовуються, належать три моделі даних:

1) ієрархічна,
2) мережева,
3) реляційна.

Крім того, останнім часом з'явилися й почали впроваджуватися на практиці ще такі моделі даних: постреляційна модель, багатовимірна модель, об'єктно-орієнтована модель. Розробляються також різні системи, основані на моделях даних, що комбінують відомі моделі. Є СУБД, що підтримують одночасно кілька моделей даних.

Ми розглянемо тільки три перші з названих моделі як найпоширеніші.

Ієрархічна модель. У цій моделі зв'язки між даними можна описати за допомогою впорядкованого графа (або дерева). Спрощене подання зв'язків між даними в ієрархічній моделі можна зобразити так, як це зроблено на рисунку 2.

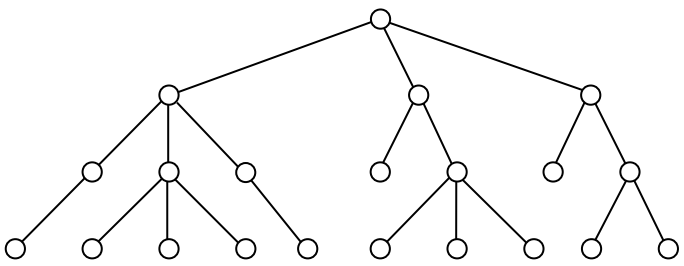

Рисунок 2

В ієрархічній моделі даних наявні два типи даних: тип “дерево” (“вузол”) і тип “запис” (“аркуш”).

Тип “дерево” є складеним. Він містить у собі деякі підлеглі дані, які у свою чергу можуть бути даними двох типів − типу “дерево” і типу “запис”. Тип “запис” може бути простим або складеним. У першому випадку це є яке-небудь значення (числове, символьне, логічне), у другому − являє собою сукупність значень. Підлеглі дані є “нащадками” стосовно даних, які виступають для них у ролі “предків” (“батьків”). Існує ще кореневий тип даних, які мають підлеглі дані, але самі не є підлеглими.

Ієрархічна БД являє собою впорядковану сукупність екземплярів даних типу “дерево”, деякі з яких містять підлеглими екземпляри типу “запис”.

До переваг ієрархічної моделі даних належать ефективне використання пам'яті ЕОМ і непогані показники часу виконання основних операцій над даними. Ієрархічна модель даних зручна для роботи з ієрархічно впорядкованою інформацією.

Недоліками ієрархічної моделі є її громіздкість для обробки інформації з досить складними логічними зв'язками, а також складність розуміння для звичайного користувача.

Мережева модель. Ця модель даних дає змогу відображати різноманітні взаємозв'язки елементів даних у вигляді довільного графа, узагальнюючи тим самим ієрархічну модель даних. На рисунку 3 зображений спрощений приклад таких зв'язків між даними в мережевій моделі.

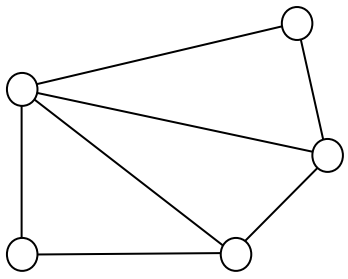

Рисунок 3

Для опису схеми мережевої БД використовують дві групи типів даних: “запису” і “зв'язку”. Екземпляр даних типу “зв'язок” визначається для двох екземплярів типу “запис”. Дані типу

“запис” мають структуру, аналогічну “записам” ієрархічної моделі. Мережева БД складається з набору записів і набору відповідних зв'язків. На формування зв'язків особливих обмежень не накладається. Якщо в ієрархічній структурі „запис-нащадок” міг мати тільки одного предка, то в мережевій моделі даних „запис-нащадок” може мати довільне число „записів-предків” (“зведених батьків”).

Перевагою мережевої моделі даних є можливість її ефективної реалізації за показниками витрат пам'яті й оперативності. У порівнянні з ієрархічною мережева модель надає більше можливостей у розумінні допустимості утворення довільних зв'язків.

Недоліком мережевої моделі даних є висока складність і твердість схеми побудованої на її основі БД. Набори зв'язків і структуру записів доводиться задавати наперед. Зміна структури даних означає перебудову всієї БД. Крім того, мережева БД складна для розуміння й виконання обробки інформації звичайним користувачам і через можливість установлення довільних зв'язків слабшає контроль цілісності даних, тобто збільшується ймовірність помилок при сприйнятті даних і їх обробці.

Реляційна модель. Ця модель даних ґрунтується на понятті відношення. Відношення являє собою множину елементів, що називаються кортежами. Кожний кортеж являє собою однотипний набір описів конкретного екземпляра деякого об'єкта (сутності, явища тощо) із предметної області, для якої будується БД.

Наочною формою подання відношення є звична для людського сприйняття двовимірна таблиця. Таблиця має рядки (записи) і стовпці. Кожний рядок таблиці має однакову структуру й складається з полів. Поля зберігають значення описів об'єкта. Рядки таблиці відповідають кортежам відношення, а кожний стовпець містить у собі значення якого-небудь одного опису, що характеризує об'єкт, для якого задане відношення.

За допомогою однієї таблиці зручно описувати велику кількість зв'язків між даними, а саме розподіл одного об'єкта на безліч його екземплярів. Але, оскільки в рамках однієї таблиці не вдається описати більш складні логічні структури даних із

предметної області, у реляційних БД задають ще зв'язування таблиць.

Перевага реляційної моделі даних полягає в простоті, зрозумілості й зручності фізичної реалізації на ЕОМ. Саме простота й зрозумілість для користувача стали основною причиною їх широкого використання. Крім того, незалежність даних у реляційній структурі дає змогу реляційним БД рости й перетворюватися без перебудови вже накопиченої в них раніше інформації.

Основним недоліком реляційної моделі є відсутність стандартних засобів ідентифікації окремих записів, а також складність опису ієрархічних і мережевих зв'язків.

Про широке розповсюдження й популярність реляційних СУБД свідчить факт, що майже всі закордонні СУБД для ПЕОМ, які використовуються у всьому світі, є реляційними. До них належать: dBase III Plus і dBase IV (фірма Ashton-Tate), DB2 (IBM), RBase (Microrim), FoxPro ранніх версій і FoxBase (FoxSoftware), Paradox і dBase for Windows (Borland), FoxPro більш пізніх версій, Visual FoxPro і Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer Systems) і Oracle (Oracle).

До українських СУБД реляційного типу належить система ПАЛЬМА (ІК АН України), до російських – система HyTech (МІФІ).

10 КЛАСИФІКАЦІЯ ІС

ІС класифікуються за різними ознаками. Найбільш часто використовуються такі класифікації:

1) за масштабом;
2) за сферою застосування;
3) за способом організації.

За масштабом ІС підрозділяються на такі групи:

1) одиночні;
2) групові;
3) корпоративні.

Одиночні ІС реалізуються, як правило, на автономному персональному комп'ютері (тобто мережа не використовується). Така система може містити кілька простих додатків, пов'язаних загальним інформаційним фондом. Вона розрахована на роботу одного користувача або групи користувачів, що розділяють за часом одне робоче місце.

Групові ІС орієнтовані на колективне використання інформації членами робочої групи й будуються на базі локальної обчислювальної мережі.

Корпоративні ІС є розвитком систем для робочих груп. Вони орієнтовані на великі компанії й можуть підтримувати територіально рознесені вузли або мережі. В основному вони мають ієрархічну структуру з декількох рівнів.

За сферою застосування ІС звичайно підрозділяються на чотири групи:

1) системи обробки транзакцій;
2) системи підтримки ухвалення рішення;
3) інформаційно-довідкові системи;
4) офісні інформаційні системи.

Системи обробки транзакцій (а транзакціями називають послідовності операцій над БД, які СУБД відслідковує як єдине ціле, тобто як одну “подію”) застосовуються для відображення реального стану предметної області в будь-який момент часу.

Системи підтримки ухвалення рішення становлять такий тип ІС, у яких за допомогою досить складних запитів виконується відбір і аналіз даних у різних площинах: часових, географічних або за іншими показниками.

Інформаційно-довідкові системи основані на обробці гіпертекстових документів і мультимедіа. Найбільший розвиток такі ІС одержали в мережах Інтернет.

Офісні ІС призначаються для переведення паперових документів до електронного вигляду, автоматизації діловодства й управління документообігом.

За способом організації групові й корпоративні ІС підрозділяються на такі класи:

1) системи на основі архітектури файл-сервер;
2) системи на основі архітектури клієнт-сервер;
3) системи на основі багаторівневої архітектури;

4) системи на основі Інтернет і інтернет - технологій.

Ці ІС являють собою обчислювальні мережі, більшість вузлів яких є клієнтськими компонентами системи й тільки невелика частина − серверними компонентами. Користувачі такої ІС звертаються до неї через клієнтські компоненти. Дані про предметну область розміщаються в серверних компонентах.

Щоб зрозуміти організаційні розходження в архітектурі ІС, у будь-якій ІС виділяють кілька функціональних складових:

− компоненти діалогу (вони діляться на обслуговування подання даних і логіку подання даних);

− компоненти обробки даних (вони діляться на прикладну логіку й логіку оперування даними);

− компоненти управління даними (вони діляться на обслуговування баз даних і обслуговування файлової системи).

При архітектурі ІС за типом файл-сервер поділ компонентів діалогу відсутній, і клієнтські комп'ютери виконують усі функції з уведення й відображення даних та їх прикладної обробки. Файл-сервер тільки витягує дані з файлів БД. Основне навантаження з оперування даними лежить на клієнтах. Кожний новий клієнт додає обчислювальну потужність до мережі (це перевага такої архітектури). Однак при виконанні деяких запитів клієнтові можуть передаватися більші обсяги даних, що призводить до надмірного завантаження мережі й непередбачуваності часу реакції (це істотний недолік).

Коли ІС організована за типом клієнт-сервер, то в ній існує дворівнева обробка даних. Компоненти забезпечення діалогу та частина компонентів обробки даних (логіка оперування даними) як і раніше є функціями клієнта. Однак для скорочення навантаження на мережу й спрощення адміністрування компоненти прикладної логіки розміщаються на сервері.

Багаторівнева архітектура ІС має три і більш рівневу обробку даних. Організація діалогу належить клієнтові. Однак прикладна логіка й логіка оперування даними здійснюються на середньому рівні спеціальними серверами додатків. Верхній рівень, звичайно, являє собою вилучений спеціалізований сервер БД, який виділено для послуг з управління даними.

Інтернет/інтранет архітектура характеризується функціональною різноманітністю серверів. Клієнтським

комп'ютерам залишається тільки обслуговування подання даних, а все інше беруть на собі різні сервери мережі.

11 ПРОЕКТУВАННЯ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

11.1 Базові концепції. Поняття реляційної бази даних

Теорія реляційних баз даних (БД) у своїй основі була розроблена Едгаром Ф. Коддом, відомим американським фахівцем у галузі БД. Основні концепції він уперше опублікував у 1970 р.

Будучи математиком за освітою, Кодд запропонував використати для обробки даних апарат теорії множин. Він показав, що будь-яке подання даних зводиться до сукупності двовимірних таблиць особливого виду, відомого в математиці як відношення (по-англійському − relationship, звідси й назва реляційні бази даних). Ця теорія зараз використовується в алгоритмічних методах проектування БД.

БД можна визначити як уніфіковану сукупність даних, якою спільно користується весь персонал підприємства. Завдання БД можна визначити як збереження всіх необхідних даних в одному місці.

Дані деякої предметної області в реляційній БД являють собою набір відношень, що змінюються в часі.

Математично відношення можна визначити таким способом. Нехай дано n множин: D_1, D_2, D_3, ... , D_n, тоді R є *відношення* над цими множинами, якщо R являє собою деяку множину із m кортежів виду $\langle \{a_1, d_1^i\}, \{a_2, d_2^i\}, \dots, \{a_n, d_n^i\} \rangle$, де $i = \overline{1, m}$, $k = \overline{1, n}$ $d_k^i \in D_k$. Тобто елемент d_1^i належить множині D_1, елемент d_2^i належить множині D_2 і т.д. Останній елемент d_n^i належить множині D_n. Елементи кортежу $a_1, a_2, \dots, a_n$ — це є *імена атрибутів*, множини D_1, D_2, D_3, ... , D_n називають *доменами*, їхні елементи $d_1^i, d_2^i, \dots, d_n^i \ (i = \overline{1, m})$, що входять до всіх кортежів відношення R, є *значеннями атрибутів*, імена яких

задані величинами $a_1, a_2, \ldots, a_n$ ... Натуральне число n називають *ступенем відношення* R, число m — *кардинальністю відношення* R (ще — *потужністю* R).

Розглянемо, як ці абстрактні поняття відображають таблиці реляційної БД, з якою так зручно працювати користувачеві.

Щоб створити БД, потрібно спочатку досліджувати предметну область. Змістовно предметна область завжди являє собою систему об'єктів якогось реального або віртуального середовища (оточення), що взаємодіють між собою за допомогою різноманітних зв'язків. Об'єкти (предмети і явища) різняться між собою, але з них завжди можна виділити групи однорідних по своїй суті об'єктів, які описуються однаковим набором властивостей і які відрізняються один від одного в групі тільки значеннями (кількісним або якісним) цих властивостей. Такі групи однорідних об'єктів з розглянутої предметної області називають *сутностями*. Властивості, сукупність значень яких характеризує кожний об'єкт (або екземпляр) сутності, називають *атрибутом* сутності. Кожне відношення відображає в БД одну якусь конкретну сутність у вигляді набору списків (кортежів) значень атрибутів, що описують екземпляри цієї сутності.

Таким чином, *сутність* — це сукупність однорідних об'єктів будь-якої природи, дані про які зберігаються у відношенні.

Атрибути являють собою властивості, що характеризують екземпляри сутності. У структурі відношення кожний атрибут іменується.

Домен являє собою множину всіх можливих значень атрибута відношення. Дані у відношеннях вважаються порівнянними тільки в тому випадку, якщо вони належать до одного домену. Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, у загальному випадку, позбавлено значення.

Список імен атрибутів, що входять у структуру відношення (набір $a_1, a_2, \ldots, a_n$ у нашому визначенні відношення R) називають *схемою* (або *заголовком*) відношення.

Кортеж, що відповідає цій схемі відношення, являє собою множину пар {*ім'я атрибута*, *значення*}, кожна з якого містить одне конкретне ім'я атрибута, що належить схемі відношення, і

одне значення цього атрибута, що належить відповідному домену.

Множину кортежів, що належать одному відношенню, часто називають *вмістом* (або *тілом*) цього відношення.

У загальному випадку порядок кортежів у відношенні, як і в будь-якій множині, не визначений. Однак у реляційних СУБД для зручності роботи кортежі все-таки впорядковують. Для цього вибирається деякий атрибут. Якщо користувач не призначає атрибут упорядкування, система автоматично присвоює номери кортежам у порядку їх уведення.

Формально, якщо переставити атрибути у відношенні, то виходить нове відношення. Однак у реляційних БД перестановка атрибутів не приводить до утворення нового відношення.

Користувачеві будь-яке відношення, що входить у реляційну БД, подається графічно двовимірною таблицею. Вкажемо відповідність між елементами теоретико-множинного подання реляційної моделі даних і поняттями-аналогами, що використовуються в описах до роботи з реляційними СУБД. Для цього розглянемо приклад табличного подання відношення "Співробітник" із БД деякого підприємства (рисунок 4).

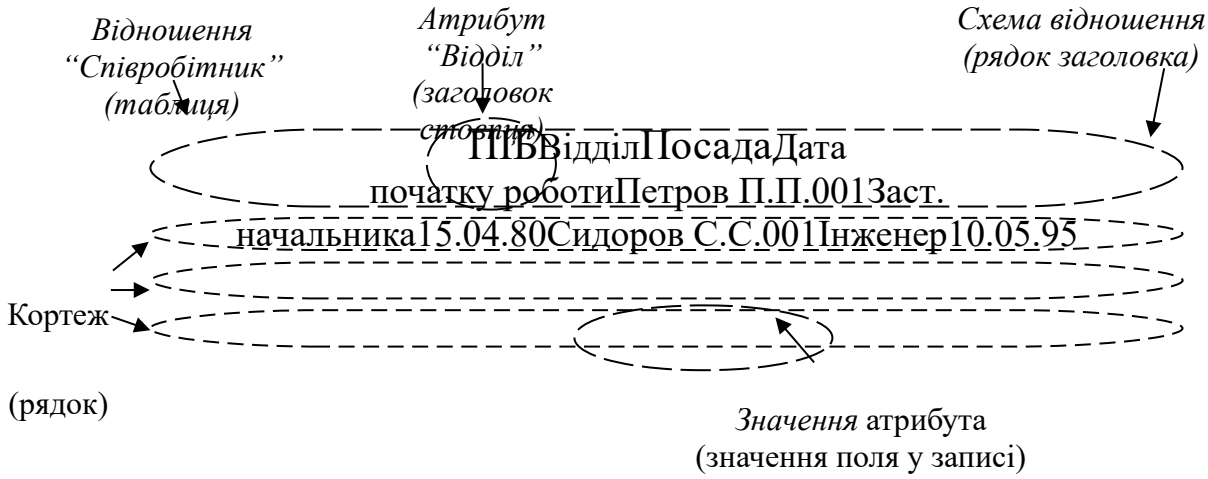

Рисунок 4

Відношення "Співробітник" пов'язане із чотирма доменами. Домен 1 містить прізвища всіх співробітників, домен 2 − номери всіх відділів підприємства, домен 3 − назви всіх посад, які є на

підприємстві, домен 4 − дати початку роботи всіх співробітників підприємства.

Зауваження. У багатьох реляційних СУБД поняття домену не використовується. У таких випадках уважається, що значення того самого атрибута у відношенні повинні бути того самого типу даних (наприклад були тільки числовими або тільки символьними). Однак це є спрощенням поняття домену для зручності користувачів.

Відношення “Співробітник” має три кортежі. У таблиці відношення значення атрибутів одного кортежу заносяться в той самий рядок. Кожному кортежу відповідає свій рядок у таблиці відношення. Кортеж розглянутого відношення складається з чотирьох елементів − значень атрибутів, кожний з яких вибирається з відповідного домену.

Імена атрибутів з відношення розміщаються в заголовному (верхньому) рядку таблиці. Нижче імені атрибута в тому ж стовпці таблиці розміщаються всі значення атрибута, що входять у кортежі відношення. Таким чином, завжди значення атрибутів з одного стовпця таблиці належать одному домену.

Часто в посібниках до роботи з реляційними СУБД рядки таблиці, що описують відношення, називають *записами*, комірки, куди заносять значення атрибутів, називають *полями*, а верхній рядок таблиці, де зберігаються імена атрибутів, називають *шапкою* таблиці. Іноді під терміном “поле” розуміють цілий стовпець таблиці, де повинні зберігатися значення одного домену.

На основі вищесказаного стає зрозуміло, що *реляційна БД* являє собою набір відношень, які містять усю інформацію про предметну область, необхідну для функціонування підприємства.

11.2 Індекси

У більшості СУБД кожне відношення із БД зберігається в окремому файлі (тобто в іменованому наборі даних, що зберігається на зовнішньому носії інформації в ЕОМ). У великих СУБД кожне відношення зберігається в індексованому файлі.

Під *індексом* розуміють засіб прискорення операції пошуку записів у таблиці-відношенні, а отже, також інших операцій, що використовуються в БД і де є пошук даних.

Індекс можна уявити як таблицю, з якою пов'язана процедура, що сприймає на вході інформацію про значення деяких атрибутів із відношення та видає на виході інформацію, яка сприяє швидкому визначенню місця розташування на носії запису або записів, які мають задані значення атрибутів.

Пояснимо ідею індексування на прикладі. Нехай у деякій реляційній БД є відношення, що має кілька десятків атрибутів, і припустимо, що для подальшої обробки даних потрібно його кортежі впорядкувати в порядку зростання значень деякого числового атрибута А. При упорядкуванні елементів у послідовності потрібно багаторазово міняти їх місцями один з одним. Якщо таку процедуру проводити безпосередньо над записами з таблиці нашого відношення, які самі нараховують у собі десятки елементів-значень, то виявиться, що треба зробити занадто велику кількість операцій, що потребують багато часу й інших ресурсів. У таких випадках застосовуються індекси. Будується таблиця, що за кількістю рядків (потужністю) дорівнює нашому відношенню. Кожний рядок цієї таблиці має вигляд, як показано на рисунку 5, де “значення А” − значення

Значення А	*Адреса початку*

Рисунок 5

атрибута А одного із записів таблиці розглянутого відношення, “Адреса початку” − адреса місця розташування початку цього ж запису на фізичному носії. Зрозуміло, що кожному запису таблиці нашого відношення буде відповідати рядок таблиці індексу.

Здійснюється необхідне упорядкування рядків таблиці індексу, при якому буде зроблено в десятки разів менше операцій перестановок елементів, ніж було б зроблено при упорядкуванні безпосередньо таблиці розглянутого відношення.

Для апарата СУБД функціонально однаково, яка із двох таблиць була впорядкована, оскільки завжди перед

використанням даних спочатку, так чи інакше, визначається їхня фізична адреса.

У прикладі був розглянутий простий спосіб задавання індексу. Існують і більш складні способи. Але скрізь головна причина підвищення швидкості виконання різних процедур з індексованими таблицями полягає в тому, що основна частина роботи здійснюється з невеликими індексами, а не із самими таблицями.

Повернемося до розгляду структури відношення в реляційній БД.

11.3 Порожні значення атрибутів

Раніше говорилося, що атрибут сутності, заданої відношенням у реляційній БД, у кожному кортежі відношення подано своїм значенням. Однак у деяких випадках який-небудь атрибут у кортежі може не мати конкретного значення. Наприклад, нехай у відношенні “Співробітник” із БД підприємства є ще один атрибут − “Номер домашнього телефону”. Значення цього атрибута в деяких кортежах може бути відсутнім, якщо є на підприємстві співробітники, які не мають стаціонарного телефону вдома. Варто розуміти, що порожнє значення атрибута − це не нуль і не порожній рядок, а значення, що не визначене в цей момент часу, але яке, загалом кажучи, може бути визначене пізніше.

Для позначення порожніх значень полів у записах таблиць із БД використовується універсальне *недійсне значення* NULL.

11.4 Первинний ключ відношення

Оскільки відношення, з математичної точки зору, є множина, що не містить однакових елементів, то ніякі два кортежі у відношенні не можуть бути дублікатами один одного в будь-який довільно заданий момент часу. Таким чином, у відношенні завжди повинен бути наявним деякий атрибут (або набір атрибутів), що однозначно ідентифікує кожний з кортежів. Такий атрибут (або набір атрибутів) називається *первинним ключем відношення*. Для кожного відношення властивість

унікальності має принаймні повний набір його атрибутів. Однак первинний ключ повинен складатися з мінімально набору атрибутів, тобто не бути надлишковим.

Більш строго визначити поняття первинного ключа можна так. Якщо R — відношення з атрибутами $A_1, A_2, \ldots, A_n$ та підмножина атрибутів $K = (A_i, A_j, \ldots, A_k)$ відношення R є первинним ключем цього відношення тоді і тільки тоді, коли задовольняються дві незалежних від часу умови:

1) унікальність — у довільний момент часу ніякі два кортежі відношення R не мають того самого значення для атрибутів з K;

2) мінімальність — жоден з атрибутів $A_i, A_j, \ldots, A_k$ не може бути виключений з K без порушення унікальності.

Можливі випадки, коли відношення має кілька комбінацій атрибутів, кожна з яких однозначно визначає всі кортежі відношення. Усі ці комбінації атрибутів є *можливими* (або *потенційними*) ключами відношення. Кожен з можливих ключів може бути обраний як первинний.

Розрізняють ключі *прості* (вони складаються з одного атрибута) і *складені,* або *складні* (вони складаються з декількох атрибутів).

Серед складених ключів одного відношення можуть зустрічатися ключі, що перекриваються, − складені ключі, які мають один або кілька загальних атрибутів.

11.5 Роль ключів в організації реляційної БД

Ключі використовуються для досягнення таких цілей:

1) для виключення дублювання значень у кортежах по ключових атрибутах (інші атрибути в розрахунок не беруться);

2) для упорядкування кортежів (це зручно робити за неповторюваним значенням ключових атрибутів);

3) для прискорення роботи з кортежами (зручно із ключовими атрибутами задавати індекси);

4) для організації зв'язування таблиць, що описують відношення.

Останнє розглянемо докладніше.

11.6 Зв'язані відношення

У реляційній моделі дані подаються у вигляді сукупності взаємозалежних відношень (таблиць). Зв'язок між відношеннями − це ще одне важливе поняття в реляційній моделі даних. Для його пояснення введемо ще одне визначення.

Нехай у відношенні *R1* є неключовий атрибут *A*, значення якого є значенням ключового атрибута *B* іншого відношення *R2*. Тоді говорять, що атрибут *A* відношення *R1* є *зовнішній ключ*. Тобто зовнішній ключ − це атрибут (або множина атрибутів) одного відношення, що є ключем іншого відношення.

Зовнішні ключі використовуються для встановлення логічних зв'язків між відношеннями. У реляційній БД *зв'язок між двома таблицями*, що описують два відношення, установлюється шляхом присвоювання значень зовнішнього ключа однієї таблиці значенням ключа другої.

11.7 Види зв'язків таблиць реляційної БД

При зв'язуванні двох таблиць виділяють *основну* й *додаткову (підлеглу)* таблиці.

Існує чотири види зв'язку:

1) ”один – один” (1:1). В основній таблиці поля зв'язку є ключем − у додатковій таблиці поля зв'язку теж є ключ;

2) ”один – багато” (1:M). В основній таблиці поля зв'язку є ключем, а в додатковій таблиці поля зв'язку не є ключем;

3) ”багато – один” (M:1). В основній таблиці атрибути для зв'язку не ключ − у додатковій таблиці атрибути для зв'язку є ключем.

4) ”багато – багато” (M:M). В основній таблиці атрибути для зв'язку не ключ − у додатковій таблиці атрибути для зв'язку також не є ключем.

Зовнішні ключі теж можуть бути простими й складеними.

Але для зв'язків використовувати складені ключі часто буває незручно. Тоді у відношення вводиться *штучний (сурогатний)* ключ. Штучний ключ завжди простий і являє собою формальний ідентифікатор (звичайно це номер) запису.

11.8 Умови цілісності даних

Інформація, що зберігається в БД, повинна бути однозначною й несуперечливою. Тому в реляційній моделі використовуються деякі *обмежувальні умови*.

Обмежувальні умови — це правила, що визначають можливі (дозволені для БД) значення даних. Вони забезпечують логічну основу для підтримки коректних значень даних у базі. Їх ще називають *обмеженнями цілісності*. Вони дають змогу звести до мінімуму помилки, що виникають при обробці й відновленні даних.

Найважливішими обмеженнями цілісності є:

1) категорійна цілісність;

2) посилальна цілісність.

Категорійна цілісність. Кортежі відношення подають у БД об'єкти реального світу (тобто категорії). Наприклад, згадаємо відношення “Співробітник”, де кожний кортеж описує конкретного співробітника підприємства. Первинний ключ таблиці однозначно визначає кожний кортеж. Тому для добування даних із БД про конкретний об'єкт або маніпулювання цими даними треба знати значення ключа відповідного запису. *Правило категорійної цілісності*: запис про об'єкт не може бути занесений до бази даних доти, поки не будуть визначені всі атрибути його первинного ключа. Є й коротке формулювання: поле ніякого атрибута первинного ключа у записі не може бути порожнім.

Посилальна цілісність. Її правило: якщо дві таблиці, що подають відношення, зв'язані між собою, то зовнішній ключ однієї таблиці повинен містити тільки значення, що уже наявні серед значень ключа в іншій таблиці, за якою здійснюється зв'язок. Якщо цього правила не дотримуватися й не контролювати коректність зовнішніх ключів, то порушиться цілісність даних. Приклад: нехай у базі підприємства крім відношення “Співробітник” є пов'язане з ним відношення “Грошове нарахування”. Якщо співробітник був звільнений (його кортеж видалили з відношення “Співробітник”), то в таблиці “Грошове нарахування”, якщо не зробити відповідні корективи, залишаться дані про зарплату співробітника, що вже пішов з

підприємства. Така ж ситуація буде, якщо зовнішньому ключу таблиці “Грошове нарахування” помилково буде присвоєне значення, відсутнє в значеннях ключа зв'язаної таблиці.

Вирішення проблеми посилальної цілісності в СУБД:

1) при уведенні нових і модифікації кортежів, які вже є, стежать, щоб не з'явилися некоректні значення зовнішнього ключа;

2) при видаленні кортежу з відношення, на який указує посилання, використовують один із трьох підходів:

а) забороняється видаляти кортеж, на який існує посилання;

б) при видаленні кортежу, на який є посилання, у всіх кортежах, що посилаються, значення зовнішнього ключа стають невизначеними;

в) при видаленні кортежу з відношення, на який посилаються, з відношення, що посилається на нього, повинні видалятися всі кортежі, що посилаються (це називається каскадним видаленням).

11.9 Цілі проектування БД

Серед багатьох цілей, які ставляться при проектуванні БД, виділимо основні:

1) можливість збереження всіх необхідних даних в одній БД;

2) виключення надмірності даних;

3) зведення кількості відношень у БД до мінімуму;

4) нормалізація БД для виключення проблем, пов'язаних з відновленням і видаленням даних.

Прокоментуємо ці цілі.

Мета 1. Передбачається, що БД повинна містити всі дані, що становлять інтерес для підприємства. І тому першим кроком у проектуванні БД є визначення всіх атрибутів, які будуть поміщені в БД. Причому намагаються також уводити в БД атрибути “на перспективу”, тобто й такі, які на сучасний момент не дуже важливі для управління й роботи підприємства.

Тільки після цього визначається кількість відношень, що входять у БД, яка проектується, і розподіляються між ними атрибути.

Мета 2. БД повинна проектуватися так, щоб містити в собі якнайбільше потрібної інформації й щоб для цього було потрібно якнайменше місця на фізичному носії. А цього не можливо досягти, якщо дані, які зберігаються в БД, будуть у ній дублюватися. Крім того, дублювання даних може призводити до проблем при обробці даних у БД.

Але варто розрізняти *необхідне дублювання* даних (або *прост*е, або *ненадлишкове*) і *надлишкове дублювання* даних. Перше може бути наявним у БД, друге − треба виключати.

Пояснимо на прикладах розходження між необхідним і надлишковим дублюванням даних.

Припустимо, є в нас таке відношення: “Співробітник-Телефон” (рисунок 6).

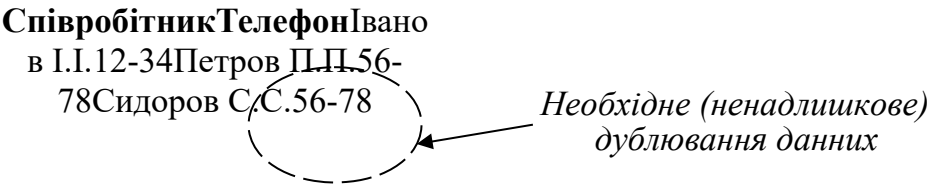

Рисунок 6

Маємо дублювання даних − номери телефонів. Але це необхідне (ненадлишкове) дублювання. Співробітники (2-й і 3-й кортежі) працюють в одному приміщенні, тому номери телефонів збігаються, хоча для кожного співробітника номер телефону унікальний.

Розглянемо ще одне відношення “Співробітник-Кімната-Телефон” у двох варіантах, які подано на рисунку 7.

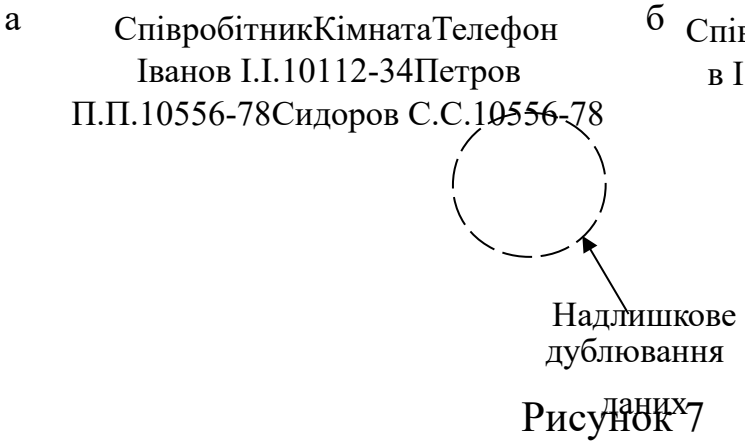

Рисунок 7

У варіанті а є надлишкове дублювання даних (номери телефонів у 2-му і 3-му кортежах). Співробітники Петров і Сидоров сидять в одній кімнаті. Природно припустити, що всі співробітники, що працюють в одній кімнаті, мають той самий телефонний номер. Тому відомості про телефонний номер Сидорова можна довідатися з кортежу з відомостями про Петрова.

Звернемося до варіанта б. Тут замість номера телефону Сидорова стоїть прочерк, тобто недійсне (невизначене) значення. Це є невдалим способом уникнути надлишкового дублювання даних з таких причин: 1) при побудові СУБД доведеться витратити додаткові зусилля на створення механізму пошуку інформації для прочерків таблиці; 2) місце на фізичному носії однаково виділяється під атрибут із прочерком, хоча дублювання даних і виключено; 3) це дуже важливо, при звільненні Петрова кортеж з відомостями про нього буде виключений з відношення і, виходить, буде знищена інформація про телефон кімнати, де він працював, що є неприпустимим.

Можливий вихід з цієї ситуації − подати відношення “Співробітник-Кімната-Телефон” у вигляді двох відношень: “Співробітник-Кімната” і “Кімната-Телефон” (рисунок 8).

КімнатаТелефон	СпівробітникКімната
101	Іванов І.І. 101
1	Петров П.П. 105
105	Сидоров С.С. 105
2-34-10-55-6-78	

Рисунок 8

Мета 3. З попереднього пояснення до мети 2 ми бачимо, що відношення іноді має сенс розбивати на два відношення або більше, щоб позбутися надлишкового дублювання даних. Однак велика кількість відношень у БД робить її незручною у використанні й незрозумілою простому користувачеві. Тому не можна допускати необдуманого збільшення кількості відношень у БД при її проектуванні.

Мета 4. Ця мета (нормалізація відношень у БД) тісно пов'язана з метою 2. При поясненні до мети 2 було відзначено, як невдале рятування від надлишкового дублювання даних може призвести до втрати даних. Такі ситуації, пов'язані з надлишковим дублюванням даних, Едгар Ф. Кодд назвав “аномаліями відновлення відношення”. Він показав, що для деяких відношень проблеми виникають при спробі видалення, додавання або редагування їхніх кортежів. Зараз у літературі з проектування БД *аномаліями* називають такі ситуації в таблицях БД, які призводять до протиріч у БД або істотно ускладнюють обробку даних.

Проектування БД повинне допомагати визначати вчасно аномалії у відношеннях і нормалізувати їх.

Нормалізація являє собою процес реорганізації відношень у БД шляхом ліквідації в них повторюваних груп даних та інших протиріч із метою приведення відношень у БД до вигляду, що дає змогу здійснювати коректну обробку даних.

11.10 Метод нормальних форм

Основним завданням, що вирішується у процесі проектування БД, є завдання нормалізації її відношень. Ми розглянемо метод нормальних форм, який слід вважати класичним методом проектування реляційних БД.

Процес проектування БД із використанням методу нормальних форм полягає в послідовному переведенні відношень з однієї нормальної форми в інші нормальні форми більш високого рівня за певними правилами. Кожна нормальна форма забезпечує усунення конкретного типу аномалій при виконанні операцій над відношенням БД і зберігає властивості нормальних форм більш низького порядку. Процедуру, за допомогою якої здійснюються переходи відношень від форми до форми, називають “*декомпозицією без втрат*”. Ця процедура являє собою розкладання відношення на два або більше відношень, де принаймні одне зі знову отриманих відношень перебуває в нормальній формі більш високого рівня.

Застосовується така послідовність нормальних форм:

1) перша нормальна форма (1НФ);

2) друга нормальна форма (2НФ);

3) третя нормальна форма (3НФ);

4) посилена третя нормальна форма або нормальна форма Бойса-Кодда (БКНФ);

5) четверта нормальна форма (4НФ);

6) п'ята нормальна форма або нормальна форма проекції-з'єднання (5НФ).

Теоретично процес нормалізації повинен тривати доти, поки БД не буде складатися тільки з відношень, що перебувають у 5НФ. Але на практиці звичайно обмежуються перетвореннями відношень до БКНФ або навіть зупиняються на 3НФ. У більшості випадків цього досить, щоб усі операції, які потрібно реально проводити над БД, здійснювалися коректно. Більш того, деякі фахівці в галузі проектування БД вважають, що досить привести таблиці БД до БКНФ, і це гарантує те, що вони перебувають у 5НФ. Це твердження має потребу в перевірці, але поки що не має ефективного алгоритму такої перевірки.

Ми теж, розглядаючи докладніше процес нормалізації відношень, не будемо підніматися вище БКНФ.

При реалізації процесу нормалізації застосовуються два підходи, рівносильні між собою.

Перший підхід полягає в послідовній декомпозиції відношень так, щоб усі знову одержувані відношення перебували у нормальній формі більш високого рівня.

Другий підхід відрізняється від першого тим, що намагаються щораз проводити декомпозицію відношення так, щоб у результаті хоча б одне зі знову отриманих відношень перебувало у БКНФ. Це відношення далі вже не змінюють, а процес декомпозиції проводять із рештою.

Обидва ці варіанти нормалізації мають одну вихідну точку — *універсальне відношення*, і опираються на одну загальну основу — поняття *функціональної залежності*.

11.10.1 Універсальне відношення

Проектування БД починається з визначення всіх об'єктів, відомості про які будуть включені в базу, і визначення їхніх

атрибутів. Потім усі ці атрибути зводяться в одну загальну таблицю − універсальне відношення.

Пояснювати це зручно на прикладі. Нехай нам необхідно спроектувати БД, наприклад для куратора курсу в університеті, тобто викладача, якому насамперед треба знати успішність кожного студента на курсі, а ще де і як студента можна знайти поза заняттями. Для простоти будемо вважати, що всі студенти живуть в одному гуртожитку. Однак нехай у кожній кімнаті гуртожитку є телефон.

Визначаємо всі ті показники, властивості, характеристики (одним словом, − атрибути), значення яких кураторові треба знати, щоб мати уявлення про успіхи курсу й про те, як при необхідності відшукати будь-якого студента. Атрибути тут визначити легко, оскільки ми добре знаємо предметну область. Перелічимо імена цих атрибутів і дамо кожному короткий опис. НомСт − номер студента: ціле позитивне число, служить для ідентифікації студента в списку або БД, є унікальним для кожного студента. ПІБ − прізвище студента і його ініціали (допускається, що у декількох студентів можуть бути однаковими прізвища та ініціали). НомКом − номер кімнати в гуртожитку; допускається, що в одній кімнаті може проживати декілька студентів. НомТел − номер телефону студента; усі студенти, що проживають в одній кімнаті, користуються одним телефоном. Предм − назва предмета; зберігаються в базі дані тільки про той предмет, вивчення якого студент закінчив. Сем − номер семестру, у якому студент закінчив вивчення предмета. Оцінка − підсумкова оцінка з предмета.

При проектуванні реальної БД перший етап являє собою дуже кропітку роботу: за результатами спілкування із замовником необхідно визначити повний перелік відомостей, якими замовник буде користуватися, і визначити, яку інформацію замовник хоче одержати в процесі експлуатації БД. Установити, як взаємодіють об'єкти предметної області, і як це відбивається на значеннях їхніх атрибутів.

Звернемось до прикладу. Уявимо, як виглядала б одна зведена таблиця, зроблена, скажімо, на аркуші паперу, у якій розмістилися б усі значення зазначених нами атрибутів. Її початок був би, приблизно, таким, як показано на рисунку 9.

НомСт	ПІБ	НомКом	НомТел	Предм	Сем	Оцінка
001	Іванов І. І.	15	12-21	Фізика Хімія Ін. мова	2 2 1	3 4 3
002	Петров П. П.	15	12-21	Матем. Інформ. Хімія	2 1 1	5 4 4
003	Сидоров С. С.	10	22-23	Фізика	1	4

Рисунок 9

Ми знаємо, що реляційна БД являє собою набір відношень. Але ця таблиця не може вважатися поданням якого-небудь відношення, оскільки одну з вимог до таких таблиць, а саме атомарність кожного значення для будь-якого атрибута, тут порушено. Щоб ця таблиця могла бути відображенням відношення, її треба перебудувати. Треба доповнити її новими записами, перерозподілити серед них значення полів, що вже є, та продублювати у порожні поля значення сусідів. Вигляд перетвореної таблиці повинен стати таким, який показаний на рисунку 10.

НомСт	ПІБ	НомКом	НомТел	Предм	Сем	Оцінка
001	Іванов І. І.	15	12-21	Фізика	2	3
001	Іванов І. І.	15	12-21	Хімія	2	4
001	Іванов І. І.	15	12-21	Ін. мова	1	3
002	Петров П. П.	15	12-21	Матем.	2	5
002	Петров П. П.	15	12-21	Інформ.	1	4
002	Петров П. П.	15	12-21	Хімія	1	4
003	Сидоров С. С.	10	22-23	Фізика	1	4

Рисунок 10

Ця таблиця вже може розглядатися як коректне зображення деякого відношення. Таке відношення називають *універсальним відношенням* проектованої БД. В одне універсальне відношення включаються всі атрибути, які становлять інтерес для користувача (замовника) БД, і воно, у принципі, може містити всі

дані, які передбачається розміщати в БД. При проектуванні БД із застосуванням методу нормальних форм універсальне відношення використовується як відправна точка.

Звичайно, у реальності при проектуванні БД побудовою зведеної таблиці або заповненням конкретними даними універсального відношення ніхто не займається. Однак визначення всього набору атрибутів, а також з'ясування, до якого типу даних будуть належати їх значення, виконуються обов'язково.

Універсальне відношення як БД практично не використовується. Тільки невеликі БД вузького призначення можуть бути подані однією таблицею (такі БД називають *плоскими таблицями*). Є відповідні програмні додатки, які дають змогу організувати ці БД і працювати з ними (наприклад, MS Works або MS Excel). Але, звичайно, в ІС їх ніколи не застосовують.

11.10.2 Проблеми, що виникають при використанні єдиного відношення

Використання універсального відношення в БД спричиняє ряд проблем.

1 *Надмірність даних.* Значення стовпців таблиці універсального відношення багаторазово повторюються.

2 *Потенційна суперечливість*. Звернемося до прикладу. Якщо, скажімо, у слові “Фізика” − значення атрибута “Предмет”, була допущена помилка, то для її виправлення треба знайти всі рядки, що містять відомості про цю дисципліну, і у всіх цих рядках зробити зміни. Більш того, при заповненні таблиці універсального відношення можуть бути використані різні форми запису того самого значення. Так, у нашому прикладі можна записати: “Ін. мова.”, “Іноз. мова” і “Іноземна мова” тощо.

3 *Проблема обробки неповних даних*. Знову звернемося до нашого прикладу. Нехай є студент, що не встиг закінчити навчання з жодного предмета. Якщо вносити про нього відомості в таблицю, то треба використати значення NULL і, отже, задавати додаткові процедури для його обробки. Якщо ж відомості про

такого студента не вносити в БД, то її користувачеві (тобто кураторові курсу) нічого про цього студента відомо не буде.

Ці проблеми вирішуються шляхом розподілу даних у базі, і засобом для цього служить дослідження атрибутів на функціональну залежність.

11.10.3 Функціональна залежність

Функціональна залежність визначається так: нехай дано два атрибути *A* і *B*, тоді *У функціонально залежить від A*, якщо для кожного значення *A* існує рівно одне, пов'язане з ним, значення для *B*.

Атрибути *A* і *B* можуть бути складними (складовими), тобто бути не одиночними атрибутами, а групами атрибутів.

Математично функціональна залежність *У* від *A* позначається записом *A→B*. Це означає, що у всіх кортежах з однаковим значенням атрибута *A* атрибут *У* буде мати також те саме значення.

Функціональну залежність *У* від *A* можна ще зображувати графічно (рисунок11).

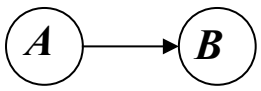

Рисунок 11

Уведемо ще кілька визначень.

↔ *Функціональна взаємозалежність*. Якщо існує функціональна залежність виду *A→У* і *B→A*, то між *A* і *B* є взаємно однозначна відповідність, або функціональна взаємозалежність. Наявність функціональної взаємозалежності між атрибутами *A* і *B* позначають як *B* ↔ *A* або *B*↔*A*.

Ми вже говорили про нормальні форми відношень. Говорили, що при проектуванні БД застосовують процедуру нормалізації, за допомогою якої відношення, що становлять БД, переводяться з однієї нормальної форми в іншу більш високого рівня. Ми знаємо також, що найнижчим рівнем нормалізації відношення є 1НФ.

Визначення 1НФ. Відношення перебуває в 1НФ, якщо всі його атрибути задаються в кортежах атомарно (мають єдине значення).

Із цього визначення висновок: будь-яке відношення (у тому числі універсальне відношення) завжди перебуває принаймні в 1НФ.

Відношення в 1НФ має властивість: якщо відношення перебуває в 1НФ, то всі неключові його атрибути функціонально залежать від ключа з різним ступенем залежності.

Частковою залежністю (частковою *функціональною* залежністю) називається залежність неключового атрибута від частини складеного ключа.

Повною функціональною залежністю називається залежність неключового атрибута від усього складеного ключа.

Транзитивна залежність. Атрибут *C* залежить від атрибута *A* транзитивно (є транзитивна залежність), якщо для атрибутів *A*, *B*, *C* виконуються умови $A \rightarrow B$ і $B \rightarrow C$, але зворотна залежність відсутня.

Повернемося до нашого прикладу (проектування БД для куратора курсу). Розглянемо функціональні залежності серед атрибутів нашого універсального відношення.

Основний спосіб визначення наявності функціональних залежностей − це ретельний аналіз семантики атрибутів. Для цього, звичайно, потрібно добре знати предметну область, розуміти її сутності й зв'язок між ними. У кожній конкретній ситуації функціональні залежності визначаються шляхом логічної деталізації властивостей атрибутів і їхньої взаємодії один з одним.

Після семантичного аналізу атрибутів нашого універсального відношення будемо мати такі функціональні залежності (ФЗ):

НомСт → ПІБ	НомКом → НомТел
НомСТ → НомКом	НомТел → НомКом
НомСт → НомТел	НомСт, Предм, Сем → Оцінка

Зобразимо всі ці залежності графічно (рисунок 12).

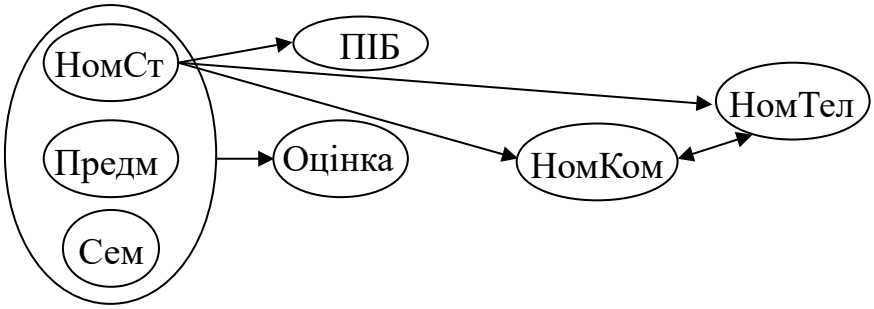

Рисунок 12

Наведені функціональні залежності були зазначені відповідно до нижченаведених міркувань.

1 ФЗ Номст > ПІБ. Номер студента є унікальним атрибутом, оскільки відомо, що в списку одним номером можна визначити тільки одне прізвище. Зворотне твердження не є правильним (однаковими прізвища й ініціали можуть бути в кількох людей у списку).

2 ФЗ Номст > Номком. Кожний студент, позначений своїм номером, проживає в якійсь одній кімнаті гуртожитку. Зворотної залежності немає, оскільки в одній кімнаті може жити кілька студентів.

3 ФЗ Номст > Номтел. З кожним студентом можна зв'язатися за якимось одним номером телефону. Зворотне твердження не буде правильним, оскільки телефонні апарати стаціонарно перебувають у кімнатах гуртожитку й одним телефоном користуються всі студенти, що живуть у кімнаті.

4 ФЗ Номком > Номтел і ФЗ Номтел > Номком. За кожною кімнатою гуртожитку закріплений тільки один телефонний номер. І якщо подзвонити за якимось конкретним телефонним номером, то ми “потрапимо” тільки в одну конкретну кімнату гуртожитку. Тут ми маємо приклад функціональної взаємозалежності.

5 ФЗ Номст, Предм, Сем > Оцінка. Оцінка може бути однозначно визначена тільки тоді, коли відомо, кому вона поставлена, за що й коли. Ця ФЗ являє собою приклад залежності простого атрибута від складеного атрибута.

11.10.4 Спосіб послідовної нормалізації реляційної БД

Розглянемо процес *послідовної нормалізації* нашого вихідного універсального відношення *з поступовим підвищенням рівня нормальних форм відношень*.

Процес нормалізації складається з послідовності процедур декомпозиції відношень. Кожна декомпозиція ґрунтується на одній або декількох операціях проекції.

Визначимо *операцію проекції* так. Припустимо, що у відношенні *R(A, B, C, D, E, ...)*, де *A, B, C, D, E, ...* — атрибути відношення *R*, усунення функціональної залежності *C→D* дає змогу перевести його в нову нормальну форму більш високого рівня. Тоді відношення *R* розкладається на два нових відношення *R1(A, B, C, E, ...)* і *R2(C, D)*. Відношення *R2* є *проекцією відношення R на атрибути C і D*. При цьому атрибути *C* і *D* можуть бути складовими (бути групами простих атрибутів).

Ми знаємо, що універсальне відношення перебуває в 1НФ. Необхідно перетворити його й одержати відношення, що перебувають у 2НФ.

Визначення 2НФ. Відношення перебуває у 2НФ, якщо воно перебуває в 1НФ і кожний неключовий атрибут функціонально повно залежить від первинного ключа.

Звернемося до схеми функціональних зв'язків нашого універсального відношення (рисунок 12). Первинним ключем щодо цього служить група атрибутів НомСт, Предм, Сем (тобто ключ є складовим). Неключові атрибути всі (як і має бути) залежать від ключа, але тільки атрибут “Оцінка” пов'язаний з ним повною функціональною залежністю. Із цього факту випливає: 1) наше універсальне відношення не перебуває у 2НФ; 2) щоб одержати з нього відношення, що перебуває у 2НФ, треба за допомогою операції проекції виключити ФЗ:

НомСт → ФИО, НомКом, НомТел.

Виконаємо таку операцію: наше універсальне відношення *R(НомСт, Предм, Сем, ПІБ, НомКом, НомТел, Оцінка)* перетвориться у два відношення *R1(НомСт, Предм, Сем, Оцінка)* і *R2(НомСт, ПІБ,*

НомКом, НомТел). Графічне зображення функціональних залежностей у відношеннях *R1* і *R2* наводиться на рисунку 13.

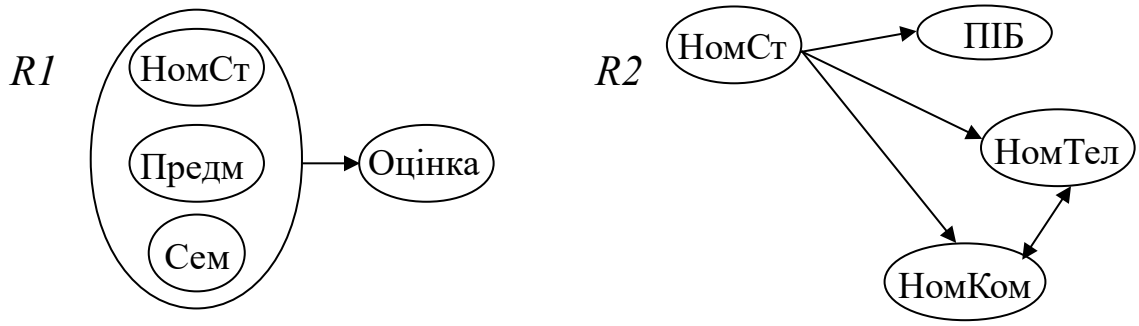

Рисунок 13

Визначення 3НФ. Визначення 1. Відношення перебуває в 3НФ, якщо воно перебуває у 2НФ, і кожний неключовий атрибут нетранзитивно залежить від первинного ключа.

Є й альтернативне визначення. *Визначення 2.* Відношення перебуває в 3НФ у тому і тільки в тому випадку, якщо всі неключові атрибути відношення взаємно незалежні й повністю залежать від первинного ключа.

Очевидно, що в нашому прикладі *R1* уже перебуває в 3НФ. Але про відношення *R2* такого сказати не можна, оскільки, по-перше, відповідно до визначення 1, у ньому є атрибути, залежні транзитивно:

НомСт → НомКом → НомТел і НомСт → НомТел → НомКом.

Крім цього, відповідно до визначення 2 3НФ, у відношенні *R2* є поміж неключовими атрибутами функціональні залежності:

НомКом → НомТел і НомТел → НомКом.

Однак, якщо ми піддамо декомпозиції відношення R2 і за допомогою операції проекції виключимо взаємозалежність НомКом ↔ НомТел, то знову отримані відношення *R3(НомСт, ПІБ, НомКом)* і *R4(НомКом, НомТел)* будуть уже перебувати в 3НФ.

На практиці побудова 3НФ схем відношень у більшості випадків є достатньою і процес проектування реляційної БД на них закінчується.

Якщо ж у відношеннях є залежності атрибутів складеного ключа від неключових атрибутів, то необхідно перейти до посиленого 3НФ.

Визначення посиленої 3НФ або нормальної форми Бойса-Кодда (БКНФ). Відношення перебуває в БКНФ, якщо воно перебуває в 3НФ і в ньому відсутні залежності ключів (атрибутів складеного ключа) від неключових атрибутів.

У нашому прикладі таких залежностей немає, тому проектування БД для куратора курсу закінчено. Його результат — набір відношень *R1, R3, R4*. Схема функціональних зв'язків цих відношень показана на рисунку 14.

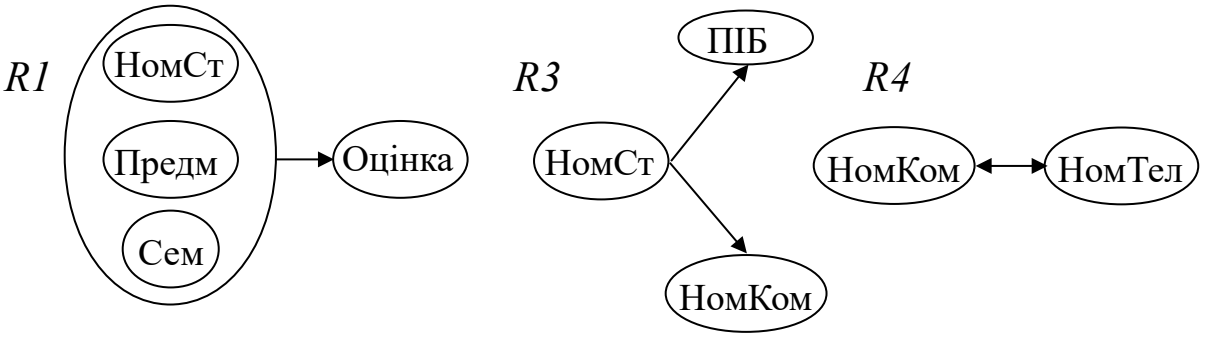

Рисунок 14

Розглянемо зв'язки між відношеннями в нормалізованій БД нашого прикладу. Очевидно, що зв'язок між двома відношеннями треба встановити за такими їхніми атрибутами, значення яких черпаються із одного домену. Такими атрибутами є, звичайно, ті, які перебувають у лівих частинах записів функціональних залежностей, що стали основами для операцій проекції.

Вид виявлених зв'язків можна встановити або шляхом семантичного аналізу, або визначенням типу ключів, якими є атрибути, що утворили зв'язок.

Схематичне зображення зв'язків між відношеннями нашого прикладу наведено на рисунку 15.

Рисунок 15

Нижче ознайомимося ще з одним підходом в організації процесу проектування БД із використанням методу нормальних форм.

11.10.5 Спосіб прямої нормалізації реляційної БД

Цей спосіб опирається на два поняття: можливий (потенційний) ключ і детермінант. Перше поняття вже було відоме, визначимо друге.

Детермінант. Якщо $A \rightarrow B$ є функціональною залежністю і B не залежить функціонально від якої-небудь підмножини A, то говорять, що A є детермінантом B.

Дамо ще одне формулювання БКНФ.

Альтернативне визначення БКНФ. Відношення перебуває в БКНФ тоді й тільки тоді, коли всі детермінанти у відношенні є можливими ключами.

Першим кроком процедури нормалізації БД тут, як і у попередньому способі, є визначення універсального відношення, а другим − установлення функціональних залежностей між атрибутами цього відношення. Далі проводиться серія декомпозицій відношень. При цьому перевіряється кожне зі знову отриманих відношень, чи перебуває воно в БКНФ. Якщо − так, то таке відношення далі не перетворюється, інші − знову зазнають декомпозиції. Причому для кожної операції проекції опорна ФЗ вибирається така, щоб її виключення з відношення приводило до утворення відношення, що вже перебуває у БКНФ.

Проілюструємо й цей спосіб нормалізації БД на прикладі, що розглядався раніше (проект БД для куратора курсу). Оскільки початкові кроки першого й другого способів збігаються,

звернемося відразу до набору ФЗ і їх графічного зображення на рисунку 12.

З рисунка ясно, що універсальне відношення *R* має тільки один можливий ключ: {НомСт, Предм, Сем}. Детермінантами цього відношення є ліві частини всіх функціональних залежностей: НомСт, НомКом, НомТел і {НомСт, Предм, Сем}. Із цього робимо висновок — універсальне відношення

R(НомСт, Предм, Сем, ПІБ, НомКом, НомТел, Оцінка)

не перебуває в БКНФ. Отже, його потрібно піддати декомпозиції.

Є декілька ФЗ для здійснення декомпозиції:

НомСт → ПІБ, НомСт →НомКом,
НомСт →НомТел, НомКом→НомТел,
НомТел→НомКом.

Але спочатку ми будемо використовувати ФЗ транзитивного типу, тобто ланцюжки виду *A→B→C*, при цьому для здійснення операції проекції треба вибирати праву ланку ланцюжка.

У нашому випадку таким ланцюжком є послідовність ФЗ: НомСт → НомКом → НомТел, а її правим кінцем є ФЗ: НомКом → НомТел, що і варто взяти основою для проекції.

У результаті цієї проекції універсальне відношення *R* розкладеться на відношення

R1(НомСт, Предм, Сем, ПІБ, НомКом, Оцінка)

і відношення *R2(НомКом, НомТел)*. Проаналізуємо обидва ці відношення. Відношення *R1* має один можливий ключ {НомСт, Предм, Сем} і два детермінанти: {НомСт, Предм, Сем} і НомСт. Висновок — *R1* не перебуває в БКНФ. Відношення *R2* має два можливих ключі (НомКом і НомТел) і два детермінанти (ті ж НомКом і НомТел). Отже, *R2* перебуває в БКНФ і не потребує подальшої декомпозиції. Відношення *R1* необхідно ще розщеплювати. Очевидно, з нього варто видалити детермінант НомСт. Цей детермінант має відразу два залежних атрибути: НомСт → НомКом і НомСт → ПІБ, що можна розглядати як складну ФЗ НомСт → НомКом, ПІБ.

Операція проекції відношення *R1*, що породжується цією функціональною залежністю, дає два відношення

R3(НомСт, Предм, Сем, Оцінка) і *R4(НомСт, ПІБ, НомКом)*.

В *R3* можливий ключ один, і детермінант один, і це все є група {НомСт, Предм, Сем}. В *R4* ми спостерігаємо ту ж картину: і можливим ключем, і детермінантом є атрибут НомСт.

Таким чином, остаточним проектом нашої БД є відношення:

R2(НомКом, НомТел), R3(НомСт, Предм, Сем, Оцінка),
R4(НомСт, ПІБ, НомКом).

Тобто, ми прийшли до такого ж результату, що й у попередньому варіанті нормалізації.

СПИСОК ЛІТЕРАТУРИ

1 Избачков Ю. С., Петров В. Н. Информационные системы: Учеб. для вузов. − 2-е изд. − СПб.: Питер, 2005.

2 Хоменко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных: Учеб. для вузов. − 4−е изд., доп. и переработ. − СПб.: Корона принт, 2004.

3 Голицына О. Л., Максимов Н. В., Попов И. И. Базы данных: Учеб. пособие. − М.: Форум, 2004.

4 Кузнецов С. Д. Основы современных баз данных. − www.citforum.ru , 2002.

5 Дейт К. Дж. Введение в системы баз данных. − 6-е изд. − М.: Вильямс, 2000.

6 Хэлворсон М., Янг М. Эффективная работа с Microsoft Office 2000. − СПб.: Питер, 2001.