ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

БАЗИ ДАНИХ:

конспект лекцій

Запоріжжя 2025

metinvest polytechnic

УДК 004.65 (072)
С12

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 6 від 28.02.2025 р.)

Автори:
Сагайда П.І., д.т.н.,
Костіков О.А., д. ф.-м. н.,

С12 Сагайда П. І., Костіков О. А. Конспект лекцій з дисципліни «Бази даних». Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2025. 65 с.

У виданні наведено конспект лекцій з дисципліни «Бази даних» для студентів економічних та технічних спеціальностей, які не спеціалізуються на вивченні інформаційних технологій. Наведено основні відомості про системи управління базами даних, реляційну модель даних, сучасні підходи до проєктування раціональних структур баз даних з використанням інформаційного та даталогічного моделювання предметних областей накопичення та обробки даних. Розглянуто можливості та особливості застосування SQL як мови запитів до реляційних баз даних, а також основи побудови застосунків баз даних.

Рекомендовано для здобувачів першого (бакалаврського) рівня освіти економічних та технічних спеціальностей, які не спеціалізуються на вивченні інформаційних технологій.

УДК 004.65 (072)

Зміст

Вступ до конспекту лекцій

У теперішній час, коли дані та інформація стають важливим національним ресурсом, який визначає рівень технологічного та соціального прогресу суспільства, а їхній об’єм одвоюється кожні п’ять років, особливого значення набуває розвиток теорії проектування і супроводу розвинених баз і банків даних і знань, а також апаратних і програмних засобів для їхньої реалізації. Технологія автоматизованої обробки даних пройшла серйозний і тривалий шлях розвитку, стала точкою дії допитливої наукової думки і полем великих практичних досліджень. Розвиток програмних засобів від перших програм на алгоритмічних мовах, у яких структури даних були залежні від програм їхньої обробки, до могутніх машин і систем обробки баз даних, пліч-о-пліч йшло з розвитком апаратних засобів і програмних технологій сучасних операційних систем. СУБД інтегрували найбільш передові механізми динамічного обміну даними і доступу до даних, що зберігаються в різних форматах, а також самі інтегрувалися в різноманітні прикладні програми і пакети програм, які вирішують задачі обробки та аналізу даних. Сучасні програмні засоби проектування виробів і технологічних процесів, керування матеріальними й інформаційними потоками, бізнес-процесами і організаційно-технічними системами в різних галузях економіки та промисловості ґрунтуються на великих сховищах даних, алгоритмах і математичних моделях, застосування яких забезпечується сучасними базами даних і знань. При цьому, незважаючи на удавану простоту і доступність, для прикладного програміста і навіть користувача, реляційної моделі даних, поширеної нині на ринку СУБД, проектування ефективних баз даних є складною задачею, що вимагає залучення інформаційного і даталогічного моделювання предметних областей, застосування навичок системного аналізу й використання сучасних технологій програмування. Окремою задачею є формулювання оптимальних запитів до баз даних на основі знання операцій реляційної алгебри, чіткого уявлення того, що відбувається на програмному й апаратному рівні, і практичного досвіду обробки даних. Фахівець з розробки і підтримки баз даних повинен, безумовно, володіти сучасними технологіями організації паралельної обробки даних, а також навичками організації баз знань, для чого необхідне залучення засобів формалізації знань і засобів штучного інтелекту для організації систем підтримки прийняття рішень.

Змістовий модуль 1. Інформаційне моделювання предметної області та даталогічне проектування систем баз даних і знань

Тема 1. Основні поняття систем баз даних, моделі зберігання даних, етапи проектування та вимоги до архітектури БД.

1.1. Бази даних, інформаційні системи, обчислювальний та штучний інтелект. Основні поняття, терміни і визначення теорії баз даних і знань

У даний час існує значний розрив між можливостями апаратних засобів комп'ютерів і застосовуваних методів рішення прикладних задач. Найбільш освоєні і використовувані в даний час методи засновані на добре формалізованих алгоритмах, отриманих у результаті побудови математичних моделей предметних областей. Найчастіше вони отримані при рішенні наступних задач: трудомісткі розрахунки за відомими формулами з використанням чисельних методів, що припускають численні ітерації обчислювальних процесів (наприклад, розрахунок методом кінцевих елементів напружених станів деталей і вузлів); прості послідовності дій, що приводять після багаторазового виконання до бажаного результату (обробка текстових і графічних документів, кодування, стиск даних і т.п.).

У практичній діяльності багато актуальних задач відносяться до наступних класів задач: таких, що погано формалізуються, для яких або невідомі аналітичні залежності чи послідовність дій, або неможливе їхнє визначення без утручання людського інтелекту (наприклад, така задача: прогнозувати зміну вартості акцій підприємства після реорганізації його з закритого у відкрите акціонерне товариство); формалізованих задач, рішення яких потрібно знайти в умовах недостатнього обсягу даних чи їхньої невизначеності (наприклад, визначити відсоток студентів навчальної групи, що вибрали свою спеціальність за покликанням).

Рішення будь-яких практичних задач зв'язано з обробкою даних, накопичених у результаті деяких вимірів, експериментів, збору інформації. Тому істотно важливе наступне: розгляд способів організації, збереження і вибірки даних про предметні області найбільш ефективним способом, тобто проектування систем баз даних; розробка методів рішення погано формалізованих задач і задач з недоліком і невизначеністю даних на основі використання методів штучного інтелекту (ШІ).

Методи ШІ на сьогоднішній день подані як наступна група методів: експертні системи (ЕС), що являють собою спосіб залучення знань експертів до рішення вищезгаданих задач (відповідно для функціонування ЕС необхідні розробка й організація баз знань (БЗ), формалізація знань, розробка методів роботи зі знаннями, прийняття рішень і т.д.); системи на основі нечіткої логіки, часто реалізовані для рішення статистичних задач; нейронні мережі, яких навчають методом проб і помилок; генетичні (еволюційні) алгоритми, які є одним зі способів стохастичної оптимізації.

Однак формалізації знань експертів найчастіше або недостатньо (у випадку, якщо таких експертів мало, чи в них немає досвіду рішення конкретної задачі), або така формалізація займає занадто багато часу й вимагає великих засобів. У такому випадку використовуються технології прогнозування: витягу закономірностей з даних – Data Mining; інтелектуального аналізу даних. На основі формалізованих знань експертів і систем із самонавчанням розробляються системи підтримки прийняття рішень – як у системах автоматизованого проектування, так і в систе-

мах автоматизованого управління.

Основою цих сучасних методик і технологій є оптимально організовані та доцільно використовувані бази даних.

За результатами вивчення даного конспекту лекцій і виконання практичних робіт студент повинний одержати наступні знання: принципів інформаційного моделювання предметної області; принципів побудови моделей даних, структури реляційної моделі даних; основ проектування реляційних баз даних методом декомпозиції; основ реляційної алгебри і побудови запитів до баз даних; архітектури і функціональних можливостей систем управління базами даних (на прикладі MS Access та/або MS SQL Server та ін.) і засобів розробки (на прикладі MS Visual Studio); основ архітектури «клієнт-сервер»; принципів розробки структур баз даних і прикладень для ведення баз і обробки даних.

Даний конспект лекцій не є вичерпним за своїм змістом, а являє собою лише вершину айсбергу знань і навичок, якими повинний володіти розроблювач і адміністратор баз даних і знань. Багато питань лише намічені, інші питання розглянуті з погляду цікавих науково-прикладних проблем, які вони вносять у практику автоматизації обробки даних.

Найважливіша причина широкого застосування засобів обчислювальної техніки і зросту програмного інструментарію зв'язана з інформаційним вибухом останнього часу – лавинним зростанням кількості інформації. Інформація і дані стали життєво важливим національним ресурсом, який повинний бути організований так, щоб цінність його і якість використання були максимальними. Виникнення безпаперової технології конструкторсько-технологічної підготовки виробництва, мультимедійних технологій, локальних і глобальних комп'ютерних мереж сприяють цим процесам.

Інформація створювалася і використовувалася до автоматизації її обробки і виникнення комп'ютерної техніки. Здавна відомі інформаційні системи (звід законів, бібліотеки, архіви і т.д.), обробка інформації у яких здійснюється людиною. Автоматизація цього процесу вимагає розробки і використання методологій, методик і алгоритмів проектування, організації і використання інформаційних систем, що дозволили б оптимізувати структуру та зміст таких систем, а також уникнути перекручування (порушення цілісності) інформації.

Для формалізованого підходу до інформаційних систем будемо використовувати ***теорію інформації***, що базується на відомих теоремах К. Шенона.

Через органи почуттів за допомогою сигналів різної природи (світлових, звукових, тактильних і т.д.) людина безупинно одержує відомості (дані) про навколишні предмети і явища, які можна назвати відомостями фізичного характеру. Далі виконується розпізнавання образів на основі отриманих відомостей, їхній аналіз, узагальнення, вироблення логічних суджень і понять предметного (студент, викладач і т.п.) і абстрактного (дисципліна, іспит і т.д.) видів. Потім відбувається фіксація результатів такої діяльності й обмін ними між членами суспільства (безпосереднє спілкування, писемність, графічні образи і т.д.). Ці результати є відомостями логічного характеру. З погляду теорії інформації, *дані* (повідомлення про настання подій) є зареєстрованими фактами про об'єкти чи явища реального світу, фіксованими у формі, придатній для їхнього наступного збереження, обробки і передачі. Однак *інформацією* є тільки ті відомості (і фізичного, і логічного характеру), що змінюють знання одержувача повідомлення про предмет, дають нові знання, відмінні від тих, що були отримані раніше.

Таким чином, інформацію можна оцінювати кількісно. Її мірою служить вели-

чина зменшення невизначеності статистичних відомостей (даних) про об'єкт повідомлення.

Кількість інформації в окремому дискретному повідомленні про подію, що має апріорну ймовірність p_i, дорівнює

$$I = k \log \frac{1}{p_i}. \qquad (1.1)$$

Пояснення. $1/p_i$ використовується відповідно до вищевикладеного визначення. Логарифм у формулі (1.1) застосований з наступних міркувань. Розглянемо два незалежних об'єкти, кожний з яких набуває n станів з однаковою ймовірністю $p_i = 1/n$. Число можливих комбінацій станів обох об'єктів, відповідно до правил комбінаторики, дорівнює n^2 з імовірністю настання кожного з них $1/n^2 = p_{ij}$.

$$I_{проста\_подія\_1} = \frac{1}{p_i} = \frac{1}{1/n} = n \; ; \; I_{проста\_подія\_2} = \frac{1}{p_j} = \frac{1}{1/n} = n ;$$

$$I_{складена\_подія} = \frac{1}{p_{ij}} = \frac{1}{1/n^2} = n^2 \; ;$$

$$I_{проста\_подія\_1} + I_{проста\_подія\_2} \neq I_{складена\_подія} .$$

Таким чином, використання логарифма необхідне для рівності суми кількостей інформації, що містяться у двох простих повідомленнях, кількості інформації, що міститься в одному складному (складеному).

$$I_{проста\_подія\_1} + I_{проста\_подія\_2} = \log \frac{1}{p_i} + \log \frac{1}{p_j} = \log n + \log n = 2 \log n \; ;$$

$$I_{складена\_подія} = \log \frac{1}{p_{ij}} = \log \frac{1}{1/n^2} = \log n^2 = 2 \log n \; .$$

Практичний інтерес має не величина (1.1), а середня кількість інформації, що приходиться на одне повідомлення:

$$I = \sum_{i=1}^{n} p_i I_i = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} = -\sum_{i=1}^{n} p_i \log p_i \; . \qquad (1.2)$$

Вираз у правій частині (1.2) характеризує невизначеність стану даного об'єкта (ентропію об'єкта):

$$H = -\sum_{i=1}^{n} p_i \log p_i \; ,$$

тобто $I = H$, але це відбувається лише у випадку, якщо після одержання повідомлення невизначеність відомостей про об'єкт зникає цілком. Це можливо, якщо повідомлення надійшли в належному обсязі, не перекручені при передачі й правильно сприйняті (інтерпретовані) одержувачем.

Автоматизований банк даних визначають як систему інформаційних, математичних, програмних, мовних, організаційних і технічних засобів, призначених для централізованого нагромадження і колективного багатоаспектного використання даних з метою одержання необхідної інформації. В автоматизованому банку даних частина функцій виконується різними елементами обчислювальної техніки,

а інша – людиною.

Базою даних (БД) називається масив даних, збережений в обчислювальній системі й організований відповідно до інформаційної моделі предметної області.

Предметною областю (ПО) називається частина реального світу, для якої розробляється інформаційна модель, облік даних про яку автоматизується.

Система управління базами даних – це пакет програм, за допомогою якого реалізується управління БД і організується доступ до неї користувачів.

База даних разом із системою управління нею є складовою частиною *банку даних*.

При створенні баз даних необхідно приділити особливу увагу тому, щоб дані можна було широко використовувати в різного роду застосунках і щоб способи використання даних можна було легко й швидко змінювати. До появи баз даних було надзвичайно важко змінити спосіб організації використовуваних даних. Різні програмісти по-різному подавали дані й постійно прагнули їх модифікувати в міру виникнення нових задач. Ці модифікації викликали значні зміни існуючих програм і тому їхнє виконання обходилося дорого. Для забезпечення гнучкості використання даних необхідно враховувати два аспекти розробки баз даних: по-перше, дані повинні бути незалежні від програм, що їх використовують, для того, щоб дані можна було додавати чи перебудовувати без зміни програм; по-друге, повинна бути забезпечена можливість запитувати і відшукувати інформацію в базі даних без трудомісткого написання програм звичайною мовою програмування. Таким чином, проектування баз даних повинне ґрунтуватися на цілком визначеній системі положень – чітко сформульованій концепції.

Концепція баз даних стала визначальним фактором при створенні ефективних систем автоматизованої обробки інформації. Бази й банки даних є одними з основних компонентів автоматизованих систем різних рівнів і типів.

Поняття про інформацію як про знання про що-небудь склалося в людини вже давно. Інформація створюється й використовується у всіх галузях людської діяльності: будь-який взаємозв'язок і координація робіт можливі тільки завдяки інформації. Людина створила інформаційні системи, оскільки існувала насущна потреба постачати виробництво інформацією, необхідною при контролі й прийнятті рішень, навчилася збирати цю інформацію, обробляти й передавати її за призначенням.

Процес осмислювання поняття інформації і її ролі у житті й діяльності людини продовжується. Поняття інформації разом з іншими науковими поняттями дозволяє більш глибоко пізнати закони розвитку матеріального світу. На сучасному етапі вважається, що інформація як властивість є загальною для усіх видів і форм руху матерії і зв'язується з тією чи іншою невід'ємною властивістю чи атрибутом матерії (відображенням, розмаїтістю, структурою, неоднорідним розподілом речовини й енергії у просторі й часі і т.д.).

Дані визначають як інформацію, фіксовану у визначеній формі, придатній для наступної обробки, збереження й передачі.

Відповідно двом поняттям — «інформація» і «дані» — у банках даних розрізняють два аспекти розгляду питань: інфологічний і даталогічний.

Інфологічний аспект уживається при розгляді питань, зв'язаних із значеннєвим змістом даних незалежно від способів їхнього уявлення в пам'яті системи. На етапі інфологічного проектування інформаційної системи повинні бути вирішені наступні питання: 1) про які об'єкти чи явища реального світу потрібно накопичувати й обробляти інформацію в системі; 2) які їхні основні характеристики і взаємозв'язки між собою будуть враховуватися; 3) уточнення понять, що вводяться в ін-

формаційну систему, про об'єкти і явища, їхні характеристики і взаємозв'язки. На етапі інфологічного проектування виділяється частина реального світу, що визначає інформаційні потреби системи, тобто її ПО.

Даталогічний аспект уживається при розгляді питань подання даних у пам'яті інформаційної системи. При даталогічному проектуванні системи, виходячи з можливостей наявних засобів сприйняття, збереження й обробки інформації, розробляються відповідні форми подання інформації в системі за допомогою даних, а також обираються моделі й методи подання й перетворення даних, формулюються правила значеннєвої інтерпретації даних. Дані відповідають зареєстрованим фактам про об'єкти чи явища реального світу. Щоб надалі використовувати дані, потрібно враховувати їхній значеннєвий зміст – *семантику даних.* Тому в інформаційній системі повинні бути сформульовані правила значеннєвої інтерпретації даних.

У деяких випадках значеннєвий зміст даних відомий на основі апріорної інформації про місце, спосіб і час їхнього формування. Наприклад, відомо, з яких датчиків (наприклад, датчика кута повороту валу чи датчика температури нагрівання поверхні виробу і т.п.) і в який час дані надходять. Тоді семантика даних визначається контекстом їхнього застосування в конкретній системі, тому досить обробляти тільки конкретні значення даних. Розглянутий спосіб інтерпретації даних широко застосовується в технічних інформаційних системах.

На практиці зустрічаються інформаційні системи, для яких названий спосіб інтерпретації даних не забезпечує нормальне функціонування. Для таких систем більш характерні способи, що дозволяють семантику даних виражати в самих даних. Це диктується необхідністю виконання в таких системах визначених видів значеннєвої обробки даних. Таким чином, дані вже не можна розглядати просто як сукупність деяких значень. Їх розцінюють як семантично значиме зображення деякої частини реального світу. Це зображення носить абстрактний характер, оскільки відповідно до цілей інформаційної системи при формуванні даних виконується абстрагування від несуттєвих деталей опису тих чи інших фактів. Тобто це зображення є цільовою інформаційною моделлю ПО, реалізованою у системі за допомогою даних.

Основний засіб зображення семантики даних – природна мова. Але можна використовувати формалізовані мови, які дозволяють більш ефективно організувати обробку даних на обчислювальній техніці й зобразити необхідну семантику даних, що задовольняє практичним потребам цілого ряду прикладних задач. До класу інформаційних систем, які використовують формалізовані мови, відносяться і банки даних.

Послугами банку даних (БнД) користується велике число різнорідних користувачів. Тому в БнД передбачається спеціальний засіб приведення всіх запитів до єдиної термінології - словник даних. Крім того, використовуються спеціальні методи еквівалентних граматичних перетворень запитів для побудови оптимальних процедур їхньої обробки, спеціальні методи організації доступу до даних різних користувачів при збігу в часі запитів, що надійшли.

Різні групи користувачів називають зовнішніми користувачами БнД. З погляду вимог до БнД із боку зовнішніх користувачів, БнД повинний: 1) задовольняти актуальним інформаційним потребам зовнішніх користувачів, забезпечувати можливість збереження і модифікації великих обсягів багатоаспектної інформації, задовольняти виявленим і знову виникаючим потребам зовнішніх користувачів; 2) забезпечувати заданий рівень вірогідності збереженої інформації і її несуперечність; 3) забезпечувати доступ до даних тільки користувачів з відповідними повноважен-

нями; 4) забезпечувати можливість пошуку інформації з довільної групи ознак; 5) задовольняти заданим вимогам продуктивності при обробці запитів; 6) мати можливість реорганізації і розширення при зміні меж ПО; 7) забезпечувати видачу інформації користувачам у різній формі; 8) забезпечувати простоту і зручність звертання зовнішніх користувачів за інформацією; 9) забезпечувати можливість одночасного обслуговування великого числа зовнішніх користувачів і т.д.

Прагнення до максимального задоволення названих вимог приводить до необхідності вирішувати питання про централізацію керування даними.

У порівнянні із традиційним забезпеченням монопольними файлами кожного застосунку, централізоване керування даними має ряд важливих переваг: скорочення надмірності збережених даних; усунення їхньої суперечливості; багатоаспектне використання; комплексна автоматизація; забезпечення можливості стандартизації; санкціонування доступу до даних. Дані – ресурс автоматизованої системи. БнД за допомогою СУБД централізовано керує цим ресурсом в інтересах усієї системи й користувачів.

Рівні абстракції у системі баз даних.

Фізична база даних – це нижній рівень організації збереження даних, зв'язаний з апаратними засобами БнД (у тому числі з фізичними носіями інформації, методами запису й кодування даних і т.д.) і низькорівневими програмними засобами (функціями BIOS для звертання до нагромаджувачів і зовнішніх пристроїв і т.д.).

Внутрішня модель даних – це механізм використання функцій і методів розвинутих операційних систем для маніпулювання даними при реалізації концептуальної моделі даних. Цей механізм дозволяє забезпечити відносну незалежність від використовуваних технічних засобів.

Концептуальна модель даних (чи просто модель даних БД) – це глобальне логічне зображення інформаційного вмісту БД, виражене засобами конкретної СУБД. З погляду адміністратора БнД, концептуальною моделлю найчастіше є набір таблиць (відношень) БД.

Зовнішня модель даних (зображення, використовувані адміністраторами й користувачами різних рангів) – це різні моделі ПО, що існують у такому вигляді, який не залежить від конкретної використовуваної СУБД і спирається на знання про ПО і на використання природної мови. Цей рівень є інформаційним рівнем абстракцій баз даних, зв'язаним з виділенням і описом властивостей ПО і їх обмеженням.

На рисунку 1.1 наведена схема взаємодії різних рівнів абстракції. Відображення (перетворення, показані на схемі у вигляді стрілок) – це програмний або апаратно-програмний інтерфейс, що використовує описи відповідностей між елементами моделей даних, які знаходяться на різному рівні абстракцій.

При більш детальному підході до даного питання можна виділити набагато більше число рівнів абстракцій.

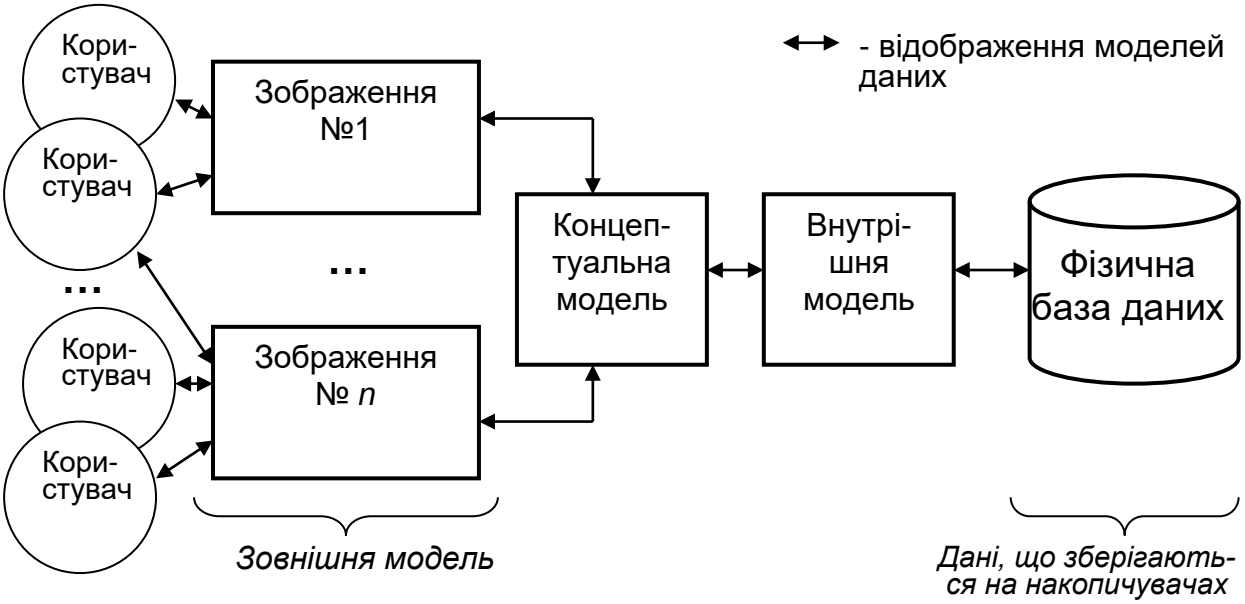


Рисунок 1.1 – Рівні абстракцій в системі баз даних

Етапи проектування баз даних.

При проектуванні баз даних необхідно провести системний аналіз ПО і досліджувати запити (вимоги) можливих користувачів проектованої інформаційної системи. Результати цих етапів, як показано на рисунку 1.2, зводяться разом шляхом узгодження уявлень про ПО, можливостей автоматизації обробки інформації й вимог користувачів. ПО, про яку буде накопичуватися й оброблятися інформація, повинна бути істотно обмежена й розглянута з позицій конкретних користувачів. Вимоги і запити користувачів повинні бути також скорочені, тому що повна автоматизація обробки всієї інформації про ПО або нездійсненна, або спричинить величезні витрати.

Під час аналізу ПО треба побудувати ієрархію користувачів і розглядати вимоги до інформаційної системи з точки зору користувача верхнього рівня ієрархії. У такому разі інформаційна система, що розробляється, задовольнить також і вимогам усіх інших користувачів.

У ході проектування створюється концептуальне представлення бази даних, що включає визначення типів найважливіших сутностей і існуючих між ними зв'язків, виконується перетворення концептуального представлення в логічну структуру бази даних, включаючи проектування відносин. Фізичне проектування має на увазі ухвалення рішення про те, як логічна модель буде реалізована за допомогою таблиць у базі даних, створюваної за допомогою обраної СУБД.

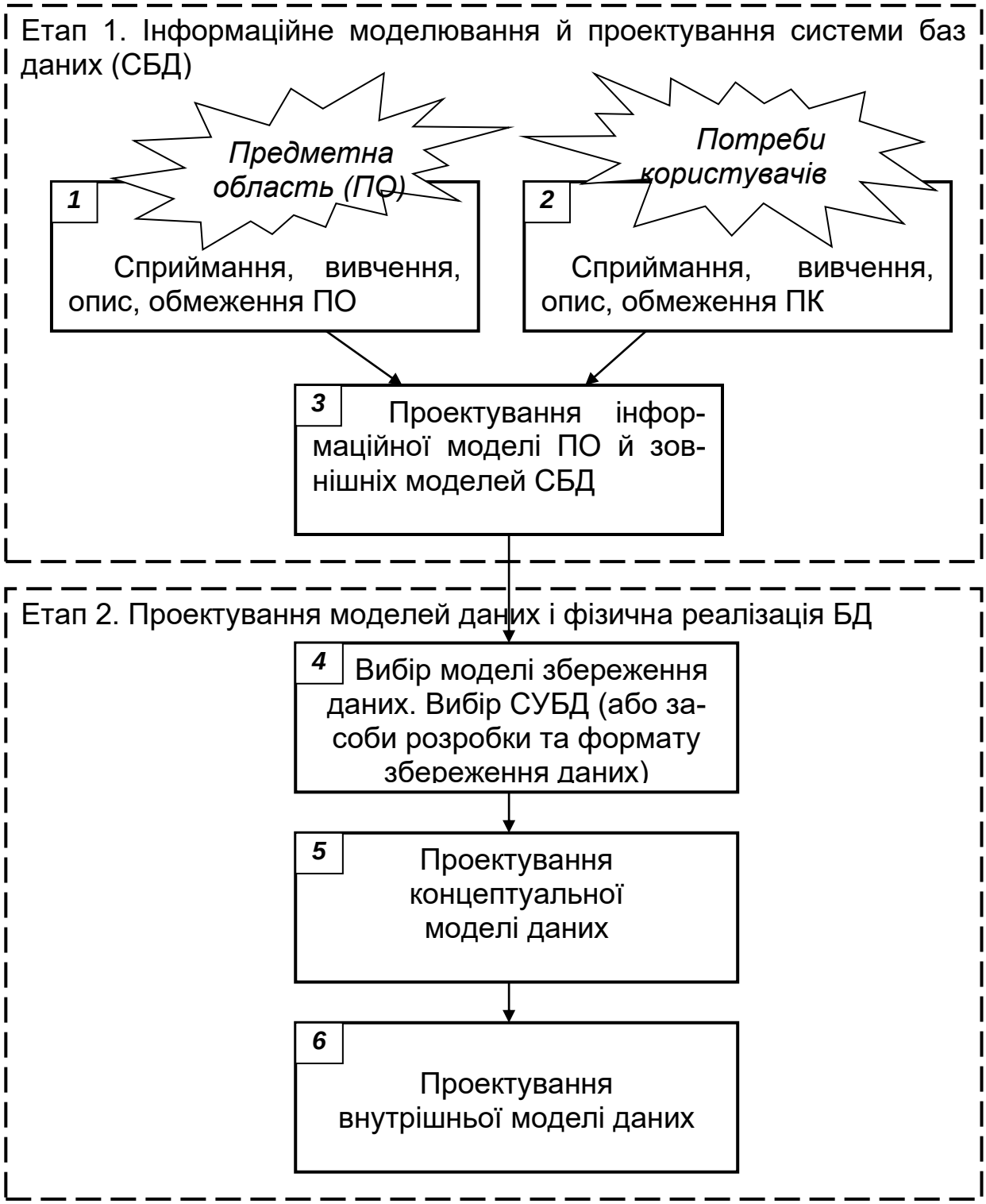


Рисунок 1.2 – Схема процесу проектування системи баз даних

1.2 Основи інтелектуалізації застосунків баз даних і їх інтеграції в інформаційні та керуючі системи

1.2.1 Місце баз даних в інтелектуальних системах автоматизації проектування і управління

Як показує досвід розробки і впровадження систем автоматизованого проектування і управління, відсутність комплексної автоматизації не дає очікуваного підвищення ефективності функціонування технологічних процесів і виробництв. Спільне впровадження інтегрованих САПР, що автоматизують основні етапи проектування виробів і процесів, із системами управління виробництвом і документообігом, дозволяє вирішити проблему комплексної автоматизації. Побудова відкритих розподілених автоматизованих систем для проектування і керування в промисловості складає основу *технології комплексної комп'ютеризації виробництва* (Computer Acquisition and Lifecycle Support - CALS). Така технологія має на увазі могутнє і багатоаспектне інформаційне забезпечення, тому що в CALS-системах передбачається збереження, обробка й передача інформації в комп'ютерних середовищах, оперативний доступ до даних у потрібний час у потрібному місці.

Системи автоматизованого проектування (САПР) відносяться до числа найбільш складних і наукомістких автоматизованих систем. Поряд з виконанням власне проектних процедур автоматизації також підлягає керування проектуванням, оскільки сам процес проектування стає усе більш складним і має розподілений характер. На великих і середніх підприємствах існує тенденція до інтеграції САПР з АСУ підприємством і технологічними процесами, у тому числі документообігом.

Інформаційне забезпечення функціонування САПР складають наступні їхні підсистеми.

Підсистема управління ходом проектування є експертною системою на основі бази знань. База знань містить інформаційні моделі ПО, опис типових проектних процедур, типові фрагменти маршрутів проектування, об'єктивні критерії чи експертні оцінки якості проектування. Часто базу знань доповнює навчальна підсистема й інтегровані засоби (мови програмування) для поповнення бази і розробки підсистем САПР (для автоматизації спеціалізованих проектних процедур для нестатків конкретного підприємства чи підрозділу).

Підсистема управління проектними даними (PDM – Product Data Manager) своїм основним компонентом має БнД. Роботу PDM характеризує приступність ієрархічно організованих даних для непідготовленого користувача, обслуговування запитів і видача звітів не тільки в текстовій, але й у графічній формі, прив'язаній до конструкції виробу. Функціями PDM є наступні: ведення розподілених архівів документів; ведення версій проекту; створення специфікацій; захист інформації; підтримка типових форматів і конвертація даних.

Банк даних у САПР є, таким чином, важливою обслуговуючою підсистемою і має ряд особливостей. У ньому зберігаються: рідко змінювані дані (архіви, довідкові дані, типові проектні рішення); відомості про поточний стан різних версій виконуваних проектів. Побудова БнД для САПР характеризується наступними особливостями: розмаїтість і багатоаспектність проектних даних з семантики і форм зображення; висока швидкість обміну великими інформаційними об'єктами; труднощі підтримки цілісності даних (рівнобіжне існування декількох версій проекту, питання захисту та безпеки даних і т.д.); складність виконання транзакцій (виконання

математичного моделювання, відображення асоціативності даних і каскадування змін, дружність інтерфейсу при відображенні результату і т.д.); ієрархічна структура проектних даних.

Перераховані особливості враховуються багато в чому об'єктними й об’єктно-реляційними СУБД, що використовують сучасні інформаційні технології і розширений набір типів збережених даних. При цьому БнД у САПР класифікуються як *сховища даних* (DW - Data Warehouse). У DW використовується модель збереження даних “зірка”, у якій є загальна таблиця фактів, і кожному факту ставиться у відповідність декілька таблиць з необхідними атрибутами. Такий підхід забезпечує багатомірну обробку даних і виконання складних запитів, а також організацію збереження багатоаспектних даних за своєю семантикою, що наближають DW до бази знань.

Системи управління в промисловості є складними системами і мають ієрархічну структуру. *Система управління підприємством* є системою верхнього рівня, що здійснює стратегічне планування керуючих заходів. Наступними рівнями за спадною є рівні виробничої ділянки, технологічного процесу, технологічного устаткування. На цих рівнях здійснюється ситуаційне управління з урахуванням керуючих впливів верхнього рівня і тактичних задач. Відповідно серед автоматизованих систем управління розрізняються *автоматизовані системи управління підприємством* (АСУП) і *автоматизовані системи керування технологічними процесами* (АСУТП).

Типовими підсистемами АСУП є підсистеми, що виконують наступні функції: календарне планування виробництва; оперативне керування; сіткове планування проектів; управління проектуванням виробів; облік і нормування затрат праці; керування фінансами й облік основних фондів; керування запасами і постачанням; аналіз ринку і маркетинг. Реальний зміст цих функцій для конкретних підприємств називають бізнес-функціями, а маршрути рішення задач управління називають бізнес-процесами.

Особливостями сучасних АСУП є наступні: підтримка різних апаратно-програмних платформ і інформаційних технологій міжпрограмного обміну даними і розподіленими обчисленнями; модульна структура; адаптовність і конфігурування відповідно до запитів конкретних замовників і умовами ринку; наявність інтегрованих мов розробки.

Відповідно до розвитку САПР, інтегрованих з АСУП, а також з розвитком мережних технологій існує тенденція створення віртуальних виробництв, у яких процес створення документації і керуючих програм для керованого технологічного устаткування розподілений у часі й просторі між багатьма організаційно автономними проектними підрозділами.

На рівні управління технологічними процесами й устаткуванням програмне й інформаційне забезпечення подане *системами диспетчерського керування і збору даних* (Supervisory Control and Data Acquisition – SCADA). У функції таких систем входять: збір первинної інформації від датчиків; збереження, обробка й візуалізація даних; управління за заданими алгоритмами, організація зворотного зв'язку й реєстрація аварійних сигналів; зв'язок з верхніми рівнями управління; автоматизована розробка програмного забезпечення.

Для автоматизації управління процесами й об'єктами за допомогою SCADA-систем використовуються мікроконтролери, убудовані в технологічне устаткування.

1.2.2 Використання сучасних інформаційних технологій при розробці та реалізації систем автоматизації управління

Системи автоматизації управління (САУ) складними технологічними комплексами (ТК) та технологічними процесами (ТП) у даний час працюють в умовах нечітко визначених критеріїв ефективності функціонування й оптимізації керованих процесів і систем управління, нечітких обмежень і невизначених збурювань, внесених навколишньої ТК середовищем, недостовірними показаннями датчиків. Ефективно функціонувати в таких умовах, бути адаптивними, а в деяких випадках і просто зберігати працездатність САУ можуть лише за умови їх інтелектуалізації шляхом інтегрування *сучасних інформаційних технологій* (ІТ).

Інтеграція ІТ у САУ і використання їх у процесі проектування і розробки істотно змінила стан даної галузі. У таблиці 1.1 наведені результати дослідження цього впливу. Узагальнюючи вміст таблиці, потрібно сказати, що впровадження ІТ дозволяє не просто домогтися деякого удосконалення (що робила у свій час "проста" інтеграція ЕОМ у САУ), а одержати кардинальне підвищення ефективності за рахунок автоматизації проектування, переходу до динамічних моделей ТП і їхньої динамічної ідентифікації, одержання реальної адаптивності й оптимальності САУ, побудови ієрархічних систем з підтримкою прийняття рішень.

Таблиця 1.1 - Вплив ІТ на проектування і функціонування САУ ТП

Стан до впровадження ІТ	**ІТ**	**Стан після впровадження ІТ**
1	**2**	**3**
Регулятори з незмінною структурою і параметрами, що настроюються в невеликому діапазоні	Упровадження мікропроцесорних засобів і промислових ЕОМ і ПК	Регулятори зі структурою, що змінюється, і довільним настроюванням параметрів, з пам'яттю про стани ТП і процесів управління
Практична відсутність адаптивних регуляторів і оптимальних САУ внаслідок складності аналогових і мікропрограмних реалізацій, статичної постановки оптимізаційних задач		Практична реалізація адаптивних регуляторів і оптимальних САУ, динамічна постановка оптимізаційних задач
Обмежена кількість вхідних і вихідних сигналів, а також фазових координат ОУ, що враховуються в САУ при реалізації закону керування	Технічні бази даних, у тому числі розподілені; технічні бази знань з використанням нових методів представлення знань	Практично необмежена кількість аналізованих сигналів, реалізація методів теорії дискретних систем і спостерігачів для оцінки фазових координат, що не спостерігаються

Продовження таблиці 1.1

1	2	3
Статична ідентифікація ТП на основі планування експерименту і подачі вхідних сигналів визначеного виду, експертних оцінок		Динамічна ідентифікація ТП, у тому числі нестаціонарних систем, без виводу з робочого режиму на основі математичного моделювання і подачі вхідних сигналів довільного виду, спостереження за операторами
Визначення закону функціонування ОУ і закону керування САУ в момент проектування	Експертні системи реального часу, що самонавчаються, з можливістю витягу знань, системи підтримки прийняття рішень	Адаптивний вибір закону керування на основі динамічної ідентифікації
Прийняття рішень щодо встановлення закону керування експертом-розроблювачем		Прийняття рішень з встановлення закону керування системою керування
Істотний вплив невизначеності у виробничій обстановці й у показаннях датчиків на якість регулювання		Компенсація впливу невизначеності шляхом використання методів штучного інтелекту й введення інформаційної надмірності у вимірювальні канали
Недружній інтерфейс оператора зі складними САУ ТП	Засоби візуального й об'єктно-орієнтованого програмування, COM-, CORBA-, WEB-технології, звертання до апаратних засобів з мов високого рівня	Дружній графічний інтерфейс, використання мнемосхем з інформацією про стан ТК, видача довідок і попереджень, згортка вимірювальної і керуючої інформації
Тривалий низькорівневий процес розробки хитливих до помилок і виняткових ситуацій програм		Швидка розробка ефективного програмного забезпечення для САУ ТП, можливість використання стандартизованих бібліотек і власних наробітків
Графоаналітичний метод аналізу ТП і розробки САУ, інформаційна ізольованість розроблювачів один від одного	САПР і математичні і моделюючі пакети на основі розвитої теорії, засобів машинної графіки, бібліотек типових елементів	Імітаційне моделювання ТП і САУ, скорочення термінів проектування, реалізація варіантного й оптимального проектування
Відсутні чи слабкі автоматизація верхніх рівнів керування й інтеграція різних рівнів керування ТП і виробництвом унаслідок проблем обміну даними й ефективної їхньої згортки	Локальні і глобальні мережі, розвиток мережних обчислень	Реалізація багаторівневих ієрархічних САУ ТП, автоматизація не тільки ситуаційного керування (локальних САУ), але і стратегічного планування (вироблення оптимальних завдань)
Розробка локальних САУ без системного підходу до автоматизації ПО	CASE-технологія і засоби для структурного системного аналізу	Системний підхід до автоматизації, структурний аналіз предметних областей

1.2.3 Методи і технології штучного інтелекту (ШІ), використовувані для підвищення ефективності засобів автоматизації

Термін "*інтелектуальність САУ*" припускає не тільки параметричну, але й структурну адаптивність (адаптивність в умовах зміни складу устаткування) до різних економічних і виробничих умов, до новим цілям функціонування і критеріям оптимальності, а в загальному випадку - і до зміни реалізованої технології. При цьому однією із ключових можливостей інтелектуальної САУ (ІнСАУ) є наявність засобів прийняття рішень на багатоальтернативній основі в умовах нечіткої інформації, чи, як мінімум, наявності засобів підтримки прийняття рішень оператором (чи іншою особою, що приймає рішення - ОПР). Крім того, інтелектуальність має на увазі, як і у випадку визначення наявності такої в людини, можливість до навчання і самонавчання, витягу знань і їхньому використанню в процесі функціонування. Уміння витягати і використовувати знання є основним як з погляду характеристики "інтелектуальність системи", так і для забезпечення (генерування) інших функціональних можливостей, що дозволяють виявити цю інтелектуальність.

Для організації ІнСАУ необхідне залучення методів штучного інтелекту, найважливіші з якій будуть далі розглянуті.

У даний час виділяють наступні основні напрямки реалізації в ШІ:

- використання нейронних мереж (НС);
- організація експертних систем (ЕС) чи, інакше, систем, заснованих на знаннях;
- організація природно-мовних систем на основі методів теорії нечітких множин і нечіткої логіки (систем на нечіткій логіці - СНЛ).

Засоби ШІ можна розділити, у свою чергу, на власне системи ШІ (додатки) і інструментальні засоби (ІЗ), при цьому ІЗ забезпечують автоматизацію всіх етапів життєвого циклу застосунків ШІ, є більш універсальними і здатні підтримувати розробку відповідного засобу ШІ "на місці". Еволюція ІЗ призвела до дружнього і прозорого інтерфейсу, при якому для розробки застосунків тепер не вимагаються вузькі фахівці - системні аналітики і програмісти, інженери по знаннях і т.д. Разом з тим вартість ІЗ складає сотні тисяч доларів, а економічний ефект від їхнього використання, у порівнянні з розробкою спеціалізованого застосунку, поки невеликий.

Нейромережева (НМ) технологія має наступні особливості. Нейрони (універсальні нелінійні елементи з можливістю широкої зміни і настроювання їх характеристик) з'єднуються в мережу. Традиційно нейрон має один вихід **y** і декілька входів **x**, на які надходять зовнішні впливи від рецепторів і інших нейронів. При цьому вихідна величина є деякою функцією від зваженої суми вхідних значень:

$$y_j = f\left(\sum_i c_{ij} x_i + c_{0j}\right),$$

де c_{ij} - вага зв'язку, c_{0j} - поріг спрацьовування нейронів. Цю функцію називають функцією активації (передатною функцією нейрона).

Одне з правил модифікації вагових коефіцієнтів було запропоновано ще в 1949 р. Д. Хебом: якщо два нейрони збуджуються разом, то сила зв'язку між ними зростає; якщо вони збуджуються роздільно, то сила зв'язку між ними зменшується. Використовуючи таке правило, при подачі на вхід НМ навчального набору сигналів

ваги зв'язків у НМ модифікуються таким чином, що на виході з'являється необхідний сигнал.

Після такого роду навчання НМ здатні вирішувати наступні задачі: розпізнавання образів, виділення сигналів на тлі шумів, керування складною адаптивною системою управління при неможливості формалізувати експертні знання чи при відсутності таких. При цьому розроблювач, маніпулюючи деяким конструктором з типових блоків, при створенні нм особливо не задумується над процесами, що відбуваються в керованій системі. Після навчання стан мережі, знання, що нагромадилися в ній, формалізуються матрицею вагових коефіцієнтів зв'язків **с.** Така матриця не несе практично ніяких узагальнених знань поза структурою даної нм: вона не може бути пояснена (принаймні для складної нм), не може бути використана для прискорення навчання інший нм (зі структурою, що відрізняється від даної) чи для навчання на іншому об'єкті керування.

З іншого боку, нм може запам'ятовувати дії досвідченого оператора, що керує системою, а потім відтворювати їх, переміняючи зразки поводження і вибираючи серед них той, котрий найбільш близький і адекватний поточній ситуації. Система не алгоритмізує діяльність оператора, а схоплює її форми цілісно як нерозривне ціле. При цьому нм має здатність прийняти рішення (видати сигнал) при відсутності частини вхідних сигналів (ознак розпізнаваної ситуації - образа) або при комбінації сигналів, що ніколи не з'являлися в процесі навчання.

У функціонуванні нм розрізняють наступні задачі:

- навчання і запам'ятовування поведінкових зразків (еталонів);
- розпізнавання зовнішньої ситуації (віднесення її до одному з еталонів);
- нейтралізація перешкод і збурювань, виправлення помилок.

При цьому практично неможливо пояснення отриманих нм знань, а контроль їхньої адекватності можливий лише досвідченим шляхом, що знижує практичну цінність знань.

Недоліком нм також є повільна збіжність процесу їхнього навчання (строго збіжність доведена для диференціальних рівнянь, тобто для нескінченно малих кроків у просторі вагів, що означає безконечно великий час навчання; при кінцевих кроках збіжність не гарантується, тобто нм може бути не навчальна для конкретного застосунку). Крім того, щоб уникнути влучення в процесі навчання в локальний мінімум при мінімізації цільової функції в просторі ваг, необхідно використовувати різні емпіричні порозуміння (уводити додаткову силу, що не залежить від градієнта цільової функції, що дозволила б забезпечити пошук глобального екстремуму). У даний час не існує яких-небудь оцінок чи рекомендацій з необхідного обсягу пам'яті (кількості нейронів і зв'язків між ними) у нм для різних застосунків.

Вивчення особливостей функціонування нм показує, що місце даної технології в системах обробки інформації наступне. Робота нм відповідає нижчому інтуїтивному рівню реакцій, коли потрібна швидка відповідь на досить стандартну ситуацію. Якщо відповідь не знайдена або система сумнівається в його правильності, керування повинне бути передане більш високому логічному рівню. Таким, більш високим рівнем, є системи, що володіють широкою базою знань і можливістю продукування обґрунтованих логічних висновків (а також можливістю контролю прийнятих рішень).

Експертні системи (системи, засновані на знаннях) - це програмно-апаратні засоби, що можуть досить ефективно вирішувати задачі, доступні лише людині-

експерту в заданій предметній області. При цьому використання ЕС не відкидає доцільності розробки традиційних систем для рішення формалізованих задач. ЕС призначені для рішення задач, що володіють наступними характеристиками: задачі не можуть бути задані в числовій формі; цільова функція точно не визначена; не існує прийнятного алгоритмічного рішення; вихідні дані й знання про предметну область (ПО) помилкові, неоднозначні, неповні і суперечливі; великий простір пошуку рішення; дані і знання динамічно змінюються.

ЕС відрізняються від систем обробки даних тим, що в них використовується в основному символьний висновок і евристичний пошук рішення. Рішення ЕС можуть бути пояснені (мають прозорість) користувачу-оператору (на відміну від рішень, отриманих числовими чи алгоритмами статистичними методами), тому що ЕС здатні міркувати про свої знання й умовиводи.

ЕС здатні поповнювати свої знання в ході взаємодії з експертом, шляхом витягу знань з баз даних (БД) і т.п.

Продукційна ЕС формалізується трійкою

$$PS = <R,B,I> ,$$

де R - база даних, що містить поточні дані про ПО; У - база знань, що містить множину продукцій (правил виду "умова -> дія"); І - інтерпретатор (вирішувач), що реалізує процес висновку, що у циклі виконує наступні дії: визначає множину означувань {правило | набір поточних даних, при яких це правило виконується}, а також виконує визначені прирівнювання, роблячи зміни в базі даних.

Інтерпретатор формально може бути представлений четвіркою

$$I = <V, S, K, W> ,$$

де V - процес вибору з B і з R підмножини активних продукцій Ba і підмножини активних даних Ra, що будуть використані в черговому циклі роботи інтерпретатора. У складних системах механізм вибору може використовувати ієрархію правил, мережні алгоритми і т.д.;

S - процес зіставлення, що визначає множину відповідних (що задовольняються) означувань {правило | набір поточних даних};

К - процес дозволу конфліктів (чи процес планування), що визначає, яке з означувань буде виконуватися, за допомогою множини метаправил чи процедур;

W - процес, що здійснює виконання обраного зазначеного правила; результатом виконання є модифікація даних у БД чи операція введення-висновку.

Системи на основі нечіткої логіки можуть використовувати природні мови в ході свого функціонування за рахунок того, що нечіткі поняття таких мов апроксимуються нечіткими перемінними з використанням теорії нечітких множин, що в САУ, наприклад, дозволяє формалізувати досвід операторів по керуванню складними технологічними процесами. Разом з тим СНЛ цілком можуть бути використані в складі алгоритмічного забезпечення ЕС, а саме, для реалізації відповідної БЗ і правил висновку на її основі.

1.3 Моделі даних алгоритмічних мов і СУБД

Даталогічний аспект проектування БД – подання даних у пам'яті інформаційної системи. *Даталогічне проектування* вимагає: 1) розробки форми подання інформаційної моделі ПО за допомогою даних; 2) визначення моделей і методів подання й перетворення даних; 3) формулювання правил значеннєвої інтерпретації даних (бізнес-правил).

Розвиток теорії й практики проектування й експлуатації баз даних супроводжується інтенсивним розвитком *моделей даних* (МД). Кожна ЕОМ володіє відносно простою, але добре визначеною моделлю даних — це припустимі в ЕОМ формати даних і склад операцій, виконуваних над ними. За допомогою вхідної МД можна побудувати більш складні моделі даних, тобто виконати перехід до деякої абстрактної машини, що володіє більш зручною МД для подання вхідних даних і рішення прикладних задач.

Прикладами абстрактних машин є інтерпретатори (чи транслятори) з алгоритмічних мов програмування високого рівня. Кожна мова програмування високого рівня має свою модель даних, що не залежна від машинної реалізації і спроектована для поліпшення моделювання визначених видів реальних ситуацій для зручності виконання визначених видів обчислень і подання відповідних типів вхідних, проміжних і вихідних даних у програмах (для обчислювальних мов).

Моделлю даних мови програмування чи СУБД є сукупність операторів декларативного та процедурного типу, що розуміється як сукупність методів і засобів опису й динамічної зміни стану структур даних і процесів, у яких вони беруть участь, із метою моделювання деяких реальних процесів чи явищ.

З позицій прикладного програміста СУБД можна розглядати як деяку алгоритмічну мову, однак її відмінність від класичної мови в тому, що оператори декларативного типу та процедурного типу розділені й найчастіше оформлені у вигляді самостійних мов чи навіть програмних компонентів, і називаються, відповідно, *мовою опису даних* (МОД, Data Definition Language - DDL) і *мовою маніпулювання даними* (ММД, Data Manipulation Language - DML).

За допомогою ЯОД здійснюється побудова *схеми структури даних*, у якій описуються ті властивості структури, що є статичними, не міняються в ході виконання прикладних програм і які можна оголосити заздалегідь (після компіляції схема перетвориться в таблиці адресних, розмірних та інших констант). *Екземпляром схеми* є конкретна структура даних відповідного типу.

ММД використовується для вираження запитів до даних БД, збережених у конкретних екземплярах схеми.

Схема взаємодії СУБД із БД і користувачем наведена на рисунку 1.3.

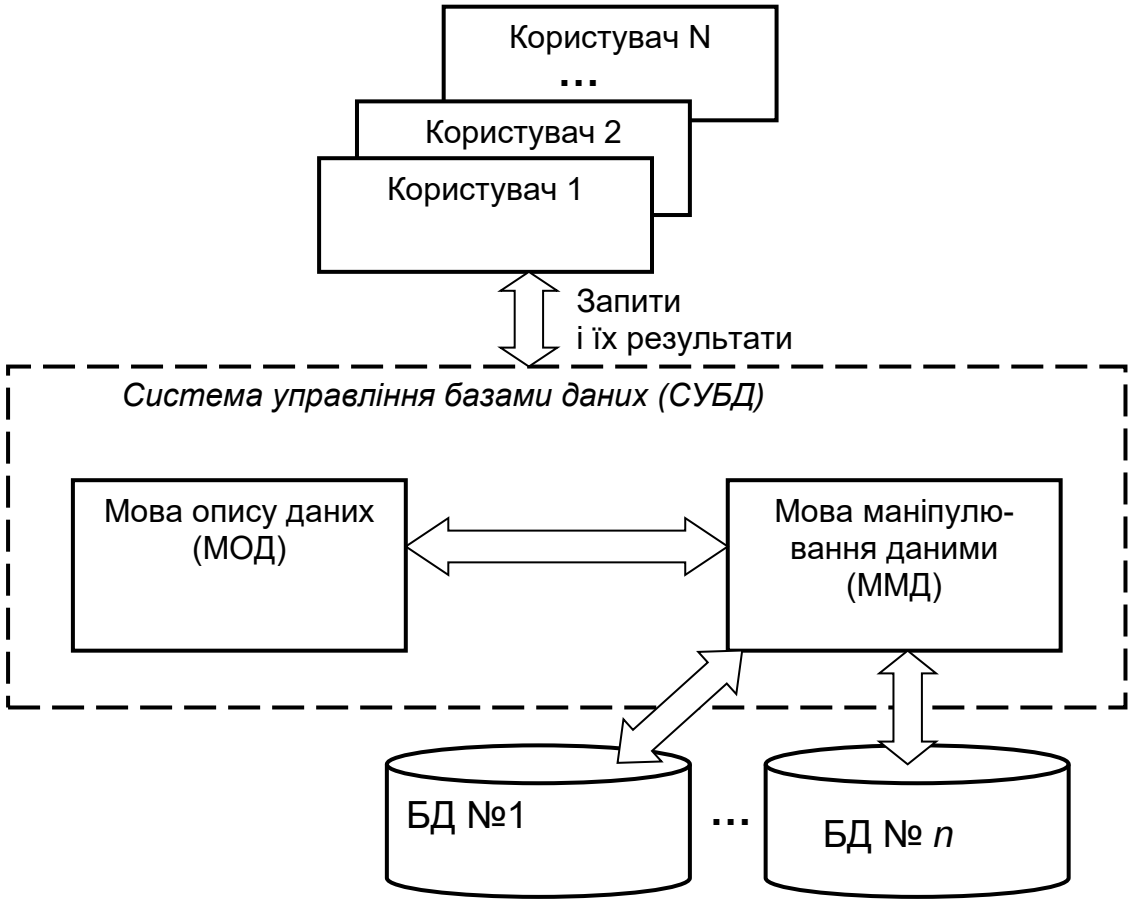


Рисунок 1.3 – Схема взаємодії користувачів з БД за посередництвом СУБД

1.4 Моделі збереження даних у СУБД

Як було вже зазначене, у БД зберігаються дані про об'єкти реального світу. У БД ці дані формалізуються у вигляді записів, полями яких є значення властивостей об'єктів. Кожен запис відповідає окремому об'єкту чи окремому зв'язку між двома чи більш об'єктами. Організацію записів, які відбивають стан частини реального світу (ПО), назвемо *моделлю збереження*.

У даний час уживаються наступні моделі збереження.

При вживанні *ієрархічної моделі* записи організовані у вигляді ієрархічних структур (дерев), при цьому існує кореневий і дочірні записи; між дочірніми записами зв'язки неприпустимі. Приклади реалізації: файлова підсистема, реєстр операційної системи й ін.

При вживанні *мережної моделі* записи організовані у вигляді мереж (графів із ненаправленими ребрами), при цьому обмежень на існування зв'язків між записами немає. І перша, і друга моделі збереження ґрунтуються на тім, що відношення між записами (зв'язки) зберігаються у вигляді фізичних покажчиків, розташованих у записах. Однак мережна модель більш підходить до моделювання реальних ПО.

При вживанні найбільш поширеної в даний час *реляційної моделі* записи організовані у вигляді упорядкованих наборів (таблиць). Зв'язки між записами зберігаються в неявному вигляді, а саме, у вигляді визначених даних у визначених полях (так званих полях для зв'язку).

Поняття реляційної моделі даних і основи теорії реляційних баз даних.

Реляційна модель даних у порівнянні з іншими моделями збереження має меншу швидкодію, тому що пошук у записах утруднений через відсутність фізичних покажчиків у зв'язаних один з одним записах. Однак важлива перевага реляційної моделі – наочність і приступність для користувача – привела до практичного домінування цієї моделі в сучасних СУБД і засобах розробки. Після того, як була розроблена теорія реляційних баз даних, методи проектування і теоретичні мови запитів, а також була реалізована реальна мова запитів – мова SQL, реляційна модель придбала статус промислового стандарту при розробці баз даних.

У зв'язку з наближеністю моделі до рядового користувача виникла термінологічна плутанина в поняттях реляційної моделі. Системні програмісти часто використовують понятійний базис “плоский файл”, користувачі – понятійний базис “таблична форма”, а аналітики-системотехніки оперують власне поняттями теорії реляційних баз даних. Відповідність цих понятійних базисів показано на рисунку 1.4.

1.5 Завдання етапу проектування БД

Можливість збереження всіх необхідних даних у БД. Передбачається, що БД повинна містити всі дані, які інтересні для користувачів. Першим кроком у процесі проектування є визначення всіх атрибутів, що далі будуть поміщені в БД. Варто вирішити, чи достатньо для їхнього розміщення й використання однієї БД.

Виключення надмірності даних і порожніх полів. *Дубльовані дані* - повторювані в БД дані, видалення яких призведе чи може призвести надалі до втрати інформації. *Надлишково дубльовані дані* - повторювані в БД дані, видалення яких не призведе до втрати інформації, тому що та ж інформація може бути отримана з інших кортежів відношень.

Керування надмірністю шляхом видалення надлишкових даних незадовільне з двох причин. По-перше, порожніх полів у БД варто уникати, тому що виникають складності при обробці таких даних (проблема інтерпретації нульових значень у числових полях і порожніх рядках у символьних, а також необхідність знаходження інших кортежів, що містять необхідну інформацію). По-друге, відношення з видаленими надлишковими даними буде мати структуру, яка призводить при видаленні кортежів до втрати необхідної для інших кортежів інформації.

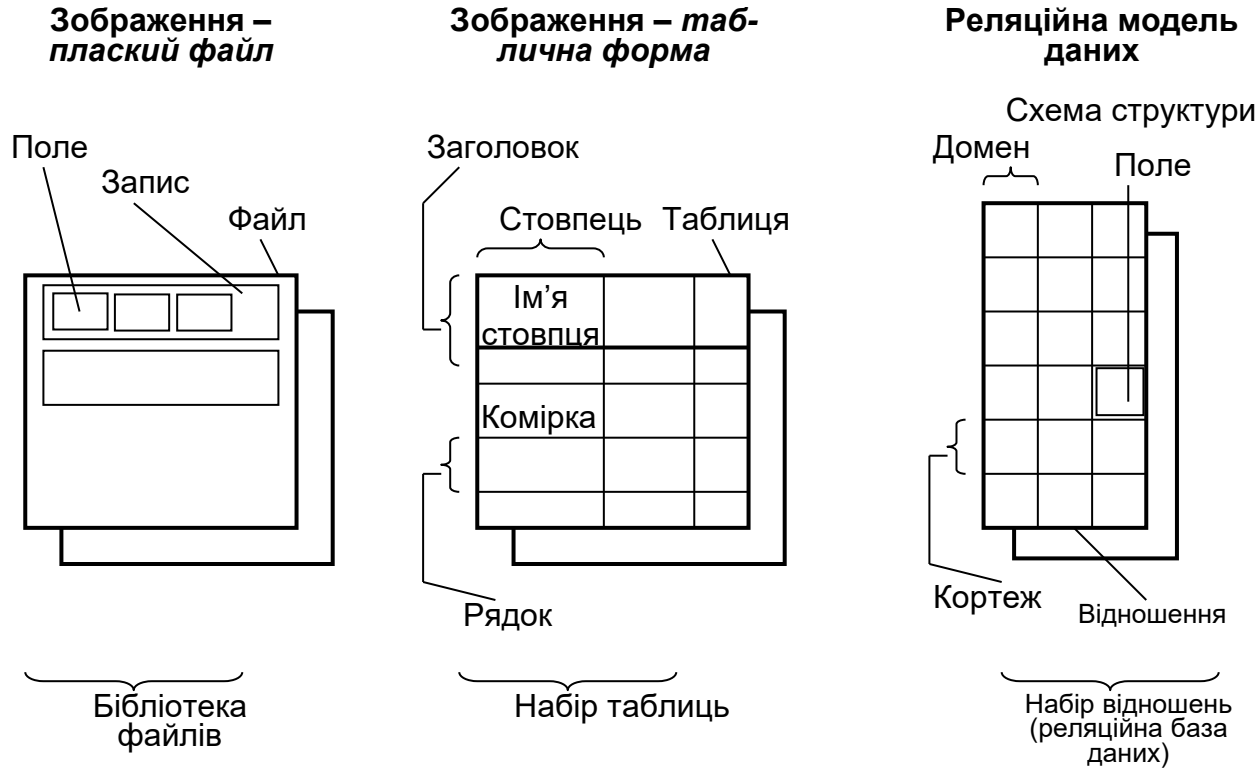


Рисунок 1.4 – Схема відповідності понятійного базису різних зображень реляційної моделі даних

Логічним шляхом усунення надмірності є створення наборів відношень. Кожній сутності ПО і кожному зв'язку між сутностями має відповідати окрема таблиця.

Зведення числа збережених у БД відношеннях до мінімуму. При цьому не можна допускати необмежене зростання числа відношень, тому що це створює проблеми для прикладного програміста й користувача.

Нормалізація відношень. Нормалізація – розбивка одного відношення на два і більш у відповідності зі спеціальною процедурою розбивки. Ця операція часто суперечить зведенню числа відношень до мінімуму.

Універсальне відношення (УВ) – відношення, що включає в себе всі атрибути, що визначені для використання в базі даних. Значеннєвий зміст атрибутів, їхнє найменування, типи даних і умови (обмеження), з ними зв'язані, визначаються в ході ітераційного процесу вивчення ПО і співбесід з адміністраторами й користувачами.

Таблична форма може і не бути відношенням у випадку, якщо не виконується одне з обмежень реляційної моделі - значення в полях записів (в комірках табличної форми) повинні бути атомарними (неподільними, що не мають внутрішньої повторюваної структури чи складного значеннєвого змісту та які можуть потребувати обробки такого значення вроздріб). У такому випадку необхідно перетворити табличну форму у відношення шляхом додавання рядків із дубльованими в них даними чи шляхом додавання нових стовпців (атрибутів).

При використанні єдиного (*універсального*) відношення виникають наступні проблеми.

Проблема вставки. УВ зберігає в одному кортежі дані про декілька об'єктів, тому при додаванні в БД даних про новий об'єкт, що виконується шляхом вставки нового кортежу, дані про інші об'єкти можуть бути ще невідомі, а це спричинить за собою наявність порожніх полів у новому кортежі.

Проблема модифікації. В УВ присутні дубльовані дані (тобто повторювані дані про ті ж самі об'єкти), що є наслідком перетворення табличної форми у відношення. При необхідності внесення змін, які стосуються значення атрибута одного об'єкта, буде потрібно відшукати й змінити всі дані, що дублюються у різних кортежах та відображають змінюване значення атрибута. При цьому велика ймовірність помилок користувачів.

Проблема видалення. УВ зберігає в одному кортежі дані про декілька об'єктів, тому при необхідності видалити з бази дані про непотрібний об'єкт буде потрібно видалити цілий кортеж. При цьому можуть безповоротно втратитися дані про потрібний об'єкт, що зберігалися тільки в кортежі, який видаляється.

Потрібно враховувати, що вказані вище проблеми можуть виникати і при використанні декількох відношень, якщо вони не нормалізовані належним образом. Ці проблеми породжують погрозу цілісності даних у БД (цілісність даних - відповідність даних один одному в різних складових частинах БД, чи інакше, відповідність змісту БД стану ПО). Крім того, вони створюють незручності прикладним програмістам і користувачам при організації й веденні БД.

Вирішуються дані проблеми розбивкою УВ на набір відношень, кожне з яких по можливості містить кортежі з даними про один об'єкт чи кортежі з даними про один зв'язок між об'єктами.

Тема 2. Проектування раціональних структур БД з використанням інформаційних моделей предметних областей та на основі концепції функціональних залежностей.

2.1 Інформаційне моделювання предметних областей

Відповідно до інфологічного підходу при проектуванні необхідно розрізняти: 1) явища реального світу; 2) інформацію про ці явища; 3) зображення цієї інформації за допомогою даних. В інфологічному підході виділені наступні три сфери: 1) реальний світ чи об'єктна система; 2) інформаційна сфера; 3) даталогічна сфера.

Об'єктна система має наступні основні складові: об'єкт, властивість, зв'язок (чи об'єктне відношення), час. *Об'єкт* в інфологічному підході - це те, про що повинна накопичуватися інформація в інформаційній системі. Вибір об'єктів виконується відповідно до цільового призначення інформаційної системи. Об'єкти можуть розглядатися як атомарні чи як складені, причому той самий об'єкт в одному прикладенні може розглядатися як атомарний, а в іншому - як складений. Для складеного об'єкта повинні бути визначені його внутрішні складові (які у свою чергу можуть бути атомарними чи складеними), внутрішня структура, відповідно до якої визначається порядок композиції складових. Кожен об'єкт у конкретний момент часу характеризується визначеним станом. Цей стан описується за допомогою обмеженого набору властивостей і зв'язків (відношень) з іншими об'єктами.

Для інформаційного моделювання, зв'язаного із проектуванням БД, частіше використовують *модель ПО типу «сутність-зв'язок»*. Така інформаційна модель є складовою частиною структурного аналізу, виконуваного за допомогою CASE-технології, і в складі цієї технології носить найменування Entity-Relaionship Diagrams (ER-діаграми). Ця модель дозволяє моделювати об'єкти ПО і взаємини цих об'єктів. Відносна простота, застосування природньої мови й легкість розуміння дозволяють використовувати модель як інструмент для спілкування з майбутніми користувачами для збору інформації про ПО при проектуванні БД. Основне призначення моделі «сутність зв'язок» - значеннєвий опис ПО й подання інформації для обґрунтування вибору видів моделей і структур даних, що надалі будуть використані в системі.

Існує кілька підходів до побудови моделей «сутність - зв'язок». Загальним для всіх підходів є використання трьох основних конструктивних елементів для подання складових ПО - сутність, атрибут і зв'язок. Інформація про проект поєднується за допомогою графічних діаграм. Складова «час» у складі конструктивних елементів у явному вигляді відсутня. Час настання подій може бути поданий в моделі шляхом використання атрибутів. Наприклад, РІК-НАРОДЖЕННЯ, ДАТА-ВСТУПУ, ДАТА-ЗАКІНЧЕННЯ і т.д.

Сутність - це збірне поняття, деяка абстракція реально існуючих об'єктів, процесів чи явищ, про які необхідно зберігати інформацію в системі. Як сутності в моделях ПО розглядаються класи матеріальних (підприємство, виріб, співробітники і т.п.) і не матеріальних (опис деякого явища, застосовуваних у системі структур даних, реферати наукових статей і т.д.) об'єктів реальної дійсності. При моделюванні використовується також поняття «екземпляр сутності». Тип сутності визначає набір однорідних об'єктів, а екземпляр сутності - конкретний об'єкт у наборі. Кожен розглянутий у моделі тип сутності повинний бути пойменований.

Для ідентифікації конкретних екземплярів сутностей використовуються спеціальні атрибути - ідентифікатори. Це може бути один чи декілька атрибутів, значення яких дозволяють однозначно відрізняти один екземпляр сутності від іншого.

Атрибут - це пойменована характеристика сутності, що набуває значення з деякої множини значень. У моделі атрибут виступає як засіб, за допомогою якого

моделюються властивості сутностей. Наприклад, для опису властивостей сутності КНИГА можна використовувати атрибути НАЗВА, ПРІЗВИЩЕ-АВТОРА, РІК-ВИДАННЯ. Щоб задати атрибут у моделі, необхідно привласнити йому найменування, подати значеннєвий опис атрибута, визначити множину його припустимих значень і вказати, для чого він використовується. Основне призначення атрибута - опис властивості сутності, а також ідентифікація екземплярів сутностей. Наприклад, атрибут ШИФР-ДЕТАЛІ, якому відповідає множина унікальних значень шифрів деталей, дозволяє однозначно ідентифікувати конкретні екземпляри сутності ДЕТАЛЬ у відповідному наборі. Атрибут можна використовувати і для зображення зв'язків (відношень) між сутностями, оскільки зв'язок (відношення) характеризує саме ті об'єкти, між якими він існує (наприклад, відношення БАТЬКО - характер споріднення), і тому може виступати в ролі властивості, ознаки сутності.

Зв'язки виступають у моделі як засіб, за допомогою якого зображуються відношення між сутностями, що мають місце в ПО. Тип зв'язку розглядається між типами сутностей, а конкретний екземпляр зв'язку розглянутого типу існує між конкретними екземплярами розглянутих типів сутностей. При аналізі зв'язків між сутностями можуть зустрічатися бінарні (між двома сутностями) і, у загальному випадку, n-арні зв'язки (між n сутностями).

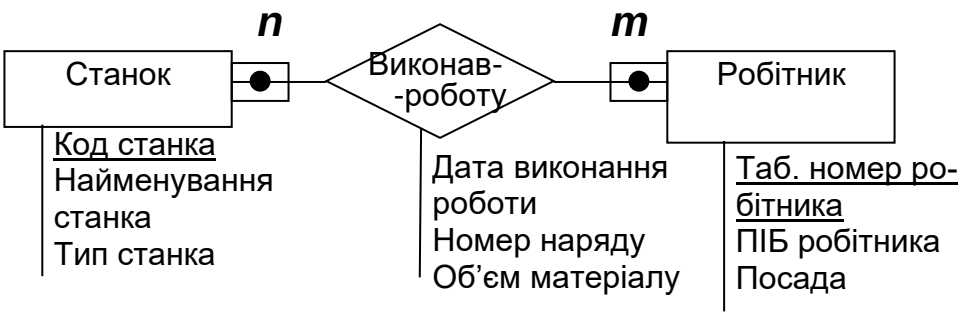
Участь сутностей у зв'язках характеризується двома критеріями: ступенем зв'язків і класом приналежності сутностей.

Ступінь зв'язку – це характеристика зв'язку, що вказує, зі скількома екземплярами однієї сутності може вступити у зв'язок (за допомогою відповідних екземплярів зв'язків) екземпляр іншої сутності.

Клас приналежності сутності – це характеристика сутності, що вказує (для даного розуміння ПО), чи кожен екземпляр цієї сутності буде у зв'язку з одним чи декількома екземплярами іншої. Звичайно, для визначення класу приналежності (обов'язкового чи необов'язкового) розроблювач задає собі питання: чи необхідно зберігати в БД дані про об'єкт, що ще не вступив у зв'язок з екземпляром іншої сутності? Наприклад, чи необхідно зберігати в БД «Робота навчального закладу» дані про викладача, що ще не читав жоден курс лекцій?

Діаграми, отримані з використанням сутностей, зв'язків і атрибутів, називають діаграмами ER-типу, а діаграми, що ілюструють екземпляри зв'язків між екземплярами сутностей, називають діаграмами ER-екземплярів. За допомогою останніх досліджують ступені зв'язків і класи приналежності сутностей. Діаграми ER-екземплярів не повинні містити в собі всі екземпляри сутностей, що існують у ПО, а повинні ілюструвати логіку роботи ПО й логіку міркувань розроблювача БД. Зразки нотації обох видів діаграм наведені на рисунку 2.1.

ER-діаграма для ПО «Облік робіт у цеху»



Діаграма ER-екземплярів для ПО «Облік робіт у цеху»

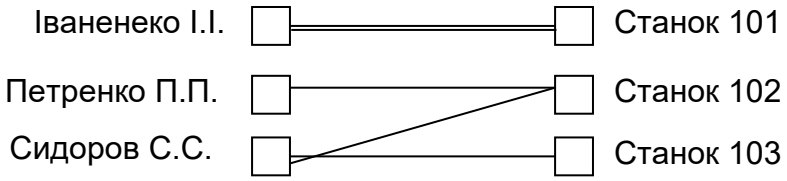


Рисунок 2.1 - Приклад ER-діаграми та діаграми ER-екземплярів

Далі розглянемо ***етапи інформаційного моделювання*** ПО.

1 При моделюванні великої ПО модель доцільно розбивати на декілька *локальних зображень*, по п'ятьох-шістьох сутностях у кожній. При цьому локальні зображення можуть відповідати окремій функціональній задачі інформаційної системи або одній передбачуваній групі користувачів. При цьому розбивання на надмірно вузькі області призводить до зниження інтеграції даних, а на дуже широкі – до складності проектування і можливих помилок.

2 Для кожного локального зображення *формулюються сутності* (набори подібних об'єктів, про які в системі повинна накопичуватися інформація). Деяка порція інформації може бути подана й сутністю, і атрибутом сутності, й зв'язком між сутністю. Необхідно відібрати з можливих варіантів моделі найбільш *гнучкий* з погляду подання інформації. Під *гнучкістю* мається на увазі простота доробки (зміни) моделі у випадку, якщо замовник інформаційної системи побажає розширити її можливості, чи якщо необхідність такого розширення виникне згодом при експлуатації. Крім того, гнучкість додасть великі можливості системі з обробки довільних запитів і при спільному використанні БД різними користувачами.

Між уведеними сутностями встановлюються типи зв'язків (далі – просто зв'язки), екземпляри яких присутні між екземплярами сутностей у ПО. Кожній уведеній сутності й зв'язку повинне бути привласнене чітке і недвозначне найменування, що відбиває значеннєвий зміст уведеного елемента моделі. Розпливчасті найменування, наявність синонімів і омонімів призводять до помилок у проектуванні. Сутностям звичайно призначають найменування-іменник в однині (СТУДЕНТ, ВИКЛАДАЧ), зв'язкам – дієслово, якщо таке можна підібрати (ВИКЛАДАЄ, ВИВЧАЄ, ВХОДИТЬ-ДО-СКЛАДУ), а в найбільш важких випадках – просто абревіатуру імен сутностей, що вступають у зв'язок (так доводиться іноді робити при найменуванні чотирьох-сторонніх зв'язків та зв'язків між ще більшим числом сутностей).

3 Для кожної сутності необхідно вказати атрибут (чи групу атрибутів), що є ідентифікатором для екземплярів сутності. *Ключем сутності називають атрибут чи їхню групу, значення яких унікально ідентифікує кожен об'єкт у наборі*

об'єктів. Ключ повинний містити в собі мінімально необхідну кількість атрибутів (бути не надлишковим). З огляду на необхідність надалі програмної обробки даних, бажано вибрати ключем порядковий номер об'єкта, інвентарний номер, серійний номер, номер документа, зв'язаного з об'єктом, та ін. (а при відсутності подібного атрибуту – увести його додатково).

4 Усі атрибути, збереження значень яких у БД визначено властивостями ПО й вимогами майбутніх користувачів, необхідно призначити як описові атрибути введеним у модель сутностям і зв'язкам. При цьому необхідно звернути увагу на *неприпустимість моделювання одного інформаційного компонента декількома елементами моделі* (наприклад, сутністю й атрибутом, зв'язком і атрибутом, і т.д.). Крім того, потрібно визначити, які атрибути у вашій моделі відбивають динаміку (час), і чи необхідні вони в моделі. Звичайно, якщо тільки модель не відбиває моментальний зріз ПО (її статичний стан), такі атрибути необхідні. Наприклад, у ПО «Робота навчального закладу» необхідні атрибути СЕМЕСТР-НАВЧАННЯ, ДАТА-ПРОВЕДЕННЯ-ІСПИТУ, ДАТА-ВСТУПУ й ін.

5 Після проведення моделювання локальних зображень необхідне об'єднання їх у єдину інформаційну модель. При цьому знову потрібно не припустити *моделювання одного інформаційного компонента декількома елементами моделі.* Інформаційна модель із великою кількістю сутностей і зв'язків повинна бути, по можливості, спрощена з використанням наступних концепцій.

Концепція агрегації має на увазі, що, з урахуванням вимог до майбутнього БД і при дотриманні гнучкості моделі, частина сутностей і зв'язків між ними може бути перетворена в одну сутність (зв'язок). Наприклад, у моделі «Робота навчального закладу» сутності ДИСЦИПЛІНА, СТУДЕНТ і ВИКЛАДАЧ, а також зв'язок СКЛАДАЄ-ІСПИТ можуть бути замінені сутністю ІСПИТ із прив'язкою до неї всіх атрибутів цих елементів.

Концепція узагальнення має на увазі, що клас різних подібних сутностей (типів об'єктів) трактується як один пойменований узагальнений тип об'єкта. Наприклад, у моделі «Робота меблевого магазину» група сутностей СТІЛЕЦЬ, СТІЛ, ДИВАН і ін., що мають однотипний зв'язок ВХОДИТЬ-ДО-СКЛАДУ із сутністю ГАРНІТУР, можуть бути замінені однією сутністю КОМПОНЕНТ-ГАРНІТУРА. У ході застосування концепції узагальнення часто необхідно погоджувати атрибути сутностей, що узагальнюються. Наприклад, якщо сутності мають однакові атрибути різної фізичної природи, в атрибути узагальненої сутності доводиться включати такі, як ВИМІР, ОДИНИЦЯ-ВИМІРУ і їм подібні.

Етапи розробки локальних зображень у ході інформаційного моделювання, а також результати використання концепцій узагальнення й агрегування проілюстровані на рисунках 2.2 - 2.3.

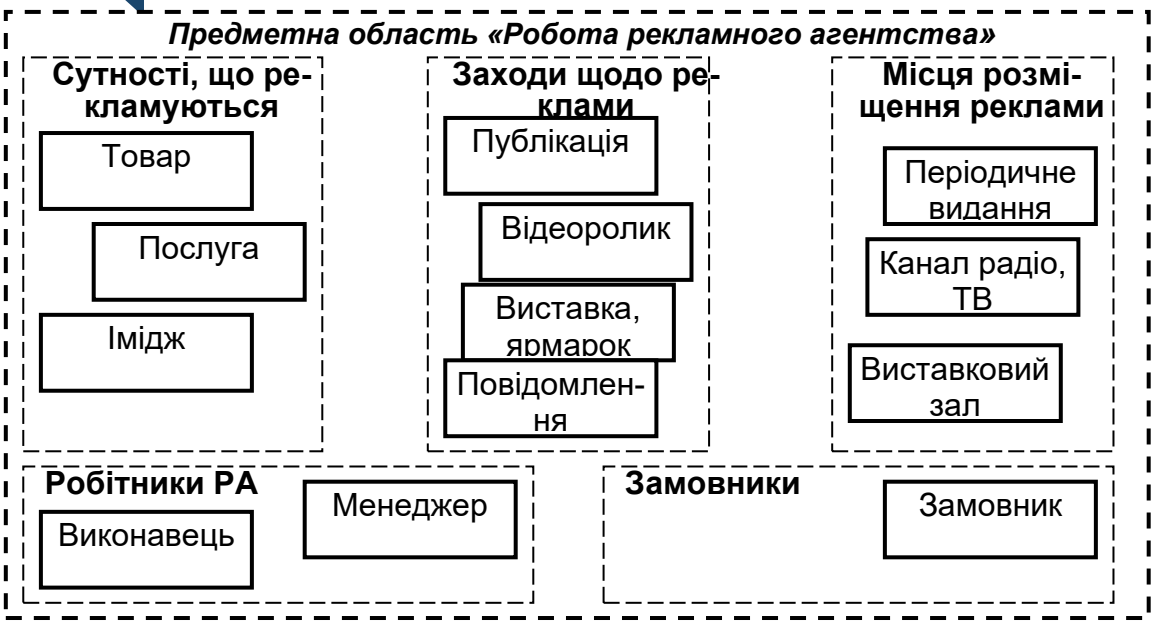


Рисунок 2.2 - Етапи розробки локальних зображень у ході інформаційного моделювання: узагальнення класів сутностей.

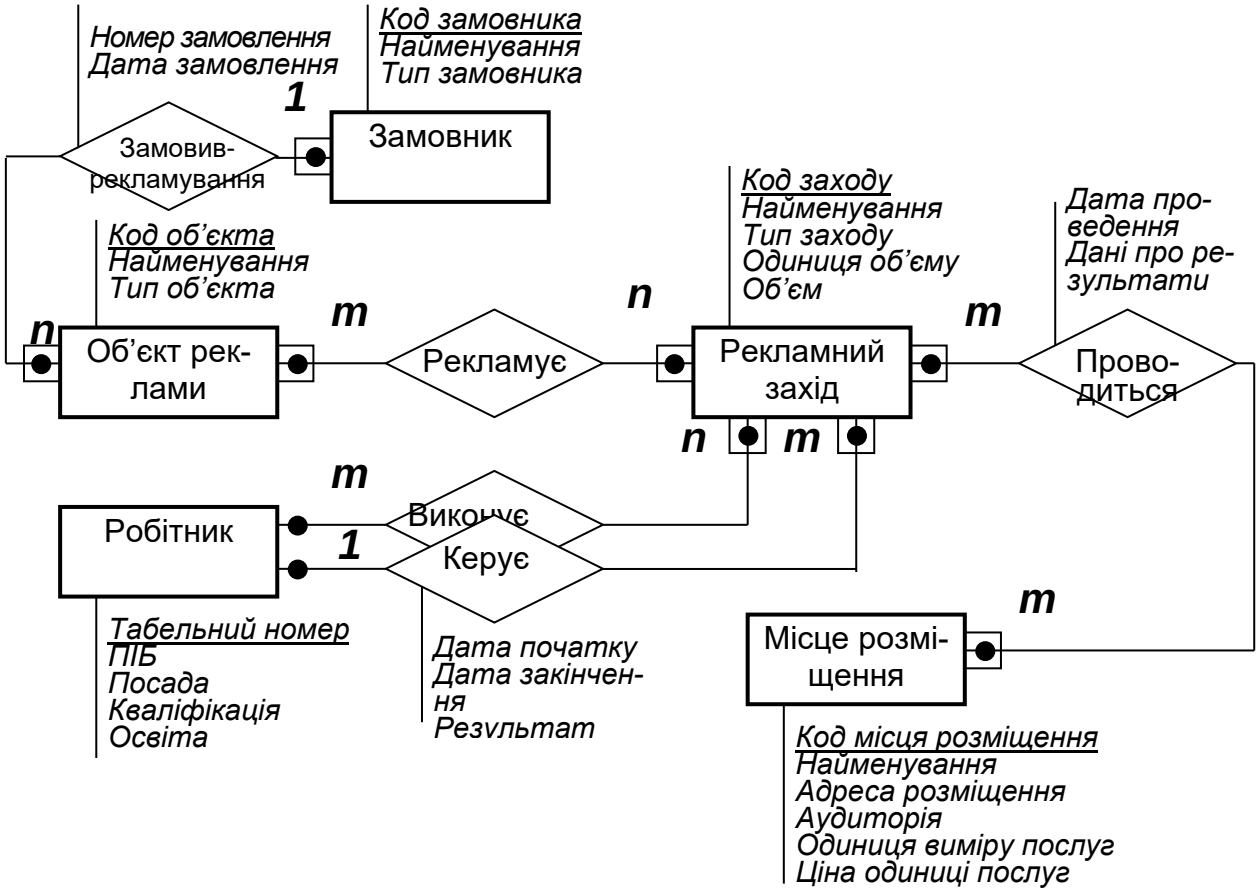


Рисунок 2.3 - Етапи розробки локальних зображень у ході інформаційного моделювання: діаграма ER-типу, що збудована в результаті використання концепцій узагальнення й агрегування

2.2 Структурний аналіз і побудова інформаційних моделей

Відповідно до наведених в пункті 1.1.2 відомостей, проведемо розробку інформаційної моделі для ПО «Робота рекламного агентства». На рисунку 2.3 проілюстровані етапи моделювання, що потребують: 1) виявлення об'єктів, які діють у ПО, і формулювання сутностей; 2) використання концепції узагальнення для об'єднання подібних сутностей в один клас; при цьому в атрибути такої узагальненої сутності обов'язково включаються атрибути ОДИНИЦІ-ВИМІРУ, ВИД, ТИП і ін.; 3) формулювання зв'язків між сутностями; 4) побудову діаграм ER-екземплярів з визначенням ступенів зв'язків і класів приналежності сутностей; 5) призначення ключових атрибутів (на рисунку 2.3 підкреслені) і розміщення описових атрибутів, у тому числі атрибутів зв'язків.

Одержання наборів відношень з інформаційних моделей ПО. Для одержання набору відношень, вільних від проблем вставки, видалення і модифікації (тобто без порожніх полів і надлишково дубльованих даних, а також таких, що не зберігають в одному записі дані про декілька об'єктів) на основі розроблених інформаційних моделей, необхідно користатися наступними правилами.

Правило 1. Якщо ступінь бінарного (між двома сутностями) зв'язку дорівнює 1:1 і клас належності обох сутностей є обов'язковим, то потрібно тільки одне відношення. Первинним ключем цього відношення може бути ключ кожної із двох сутностей. Атрибути, привласнені зв'язку, включаються у відношення, виділене для сутностей.

Правило 2. Якщо ступінь бінарного зв'язку дорівнює 1:1 і клас належності однієї сутності є обов'язковим, а іншої - необов'язковим, то необхідна побудова двох відношень - по одному для кожної сутності, при цьому ключ кожної сутності повинний служити ключем для відповідного відношення. Крім того, ключ сутності, для якого клас приналежності є необов'язковим, додається як атрибут у відношення для сутності з обов'язковим класом належності. Атрибути, привласнені зв'язку, включаються у відношення, виділене для сутності з обов'язковим класом належності.

Правило 3. Якщо ступінь бінарного зв'язку дорівнює 1:1 і клас належності обох сутностей є необов'язковим, то необхідна побудова трьох відношень - по одному для кожної сутності, ключі яких служать у якості первинних для відповідних відношень, і одного для зв'язку. Серед своїх атрибутів відношення, виділюване для зв'язку, повинне мати по одному ключу сутності від кожної сутності. Атрибути, привласнені зв'язку, включаються у відношення, виділене для зв'язку.

Правило 4. Якщо ступінь бінарного зв'язку дорівнює 1:n і клас належності n-зв'язної сутності є обов'язковим, то необхідна побудова двох відношень - по одному для кожної сутності, при цьому ключ кожної сутності повинний служити ключем для відповідного відношення. Крім того, ключ 1-зв'язної сутності додається як атрибут у відношення, що відводиться для n-зв'язної сутності. Атрибути, привласнені зв'язку, включаються у відношення, виділене для n-зв'язної сутності.

Правило 5. Якщо ступінь бінарного зв'язку дорівнює 1:n і клас належності n-зв'язної сутності є необов'язковим, то необхідна побудова трьох відношень - по одному для кожної сутності, ключі яких служать як первинні для відповідних відношень, і одного для зв'язку. Серед своїх атрибутів відношення, виділене для зв'язку, повинне мати по одному ключі сутності від кожної сутності. Атрибути, привласнені зв'язку, включаються у відношення, виділене для зв'язку.

Правило 6. Якщо ступінь бінарного зв'язку дорівнює m:n, то для збереження даних необхідна побудова трьох відношень - по одному для кожної сутності, ключі

яких служать як первинні для відповідних відношень, і одного для зв'язку. Серед своїх атрибутів відношення, виділене для зв'язку, повинне мати по одному ключу сутності від кожної сутності. Атрибути, привласнені зв'язку, включаються у відношення, виділене для зв'язку.

За допомогою бінарних зв'язків можуть бути описані багато ситуацій реального світу, однак неминуче виникнення ситуацій, у яких побудова ефективної моделі неможлива без залучення додаткових конструкцій.

При побудові діаграм екземплярів іноді виникає проблема відсутності унікального шляху, що з'єднує разом екземпляри різних сутностей, які беруть участь у зв'язках. У цьому випадку вводяться тристоронні зв'язки і зв’язки між більшим числом сутностей.

Друга проблема, розв'язувана аналогічним образом – перевантаженість діаграм зв'язками, коли виникає циклічний набір зв'язків у групи сутностей. У результаті проектування отримується набір відношень, аналогічний набору для багатобічного зв'язку, однак одержання його складне і може викликати помилки.

Правило 7. У випадку багатобічного зв'язку необхідно використовувати по одному відношенню для кожної сутності, що беруть участь у зв'язку; ключі сутностей служать як первинні для відповідних відношень; також використовується одне відношення для зв'язку, куди включаються ключі всіх сутностей і атрибути зв'язку. При цьому ступені зв'язку сутностей і їхні класи приналежностей на результат проектування не впливають.

Деякі ситуації у ПО при побудові ефективних моделей вимагають використання додаткової конструкції. Така ситуація виникає тоді, коли екземпляри деякої сутності грають різні ролі в ході функціонування ПО.

Правило 8. Уводиться одне відношення для головної сутності, ключем якого служить ключ сутності, а рольові елементи і зв'язки між ними розглядаються як звичайні сутності й зв'язки. Набір відношень одержують за відповідними правилами. При цьому ключ головної сутності повинний бути атрибутом кожного з рольових елементів.

У загальному випадку, при одержанні набору відношень на основі складної ER-діаграми використовується наступний ***алгоритм***:

- розглядається окремо кожен зв'язок із сутностями, що беруть у ньому участь, і з використанням одного із наведених вище правил отримується набір відношень, що включається в загальний набір для даної діаграми;
- розглядається отриманий загальний набір відношень, і з його складу виключаються надлишкові відношення (надлишковим вважається те відношення, множина атрибутів якого є підмножиною атрибутів іншого, або збігається з нею);
- розглядається отриманий загальний набір відношень, і відношення, що відрізняються одним-двома атрибутами, з'єднуються в одне.

Результуючий набір відношень формалізують у вигляді структури реляційної бази даних.

2.3 Проектування структур реляційних баз даних з використанням концепції функціональних залежностей

Концепція функціональних залежностей у реляційних БД. Побудова діаграм ФЗ. Надлишкові ФЗ і одержання мінімального покриття діаграми ФЗ.

Функціональна залежність (ФЗ) - це такий вид зв'язку між атрибутами А й В, коли одному значенню атрибута А під час функціонування ПО відповідає тільки одне значення атрибута В.

Первинний ключ - атрибут чи набір атрибутів, що використовується в даному відношенні для ідентифікації (розрізнення) кортежів (записів). Значення ключа, отже, унікально для кожного запису.

Можливий ключ - атрибут чи набір атрибутів, що може бути використаний для даного відношення як первинний ключ.

Детермінант ФЗ - це атрибут чи набір атрибутів, від якого функціонально залежить інший атрибут чи набір атрибутів, тобто одному значенню якого (яких) під час функціонування ПО відповідає тільки одне значення іншого атрибута (набору атрибутів). Іншими словами, детермінант - ліва частина ФЗ.

Діаграми ФЗ для конкретної ПО будуються в такий спосіб. Складається універсальне відношення (рис. 2.4), що включає в себе всі цікавлячі проектувальника атрибути ПО. Потім усі атрибути перевіряються на існування між ними ФЗ, у загальному випадку, простим перебором. Оскільки такий метод побудови діаграми ФЗ трудомісткий і не наочний, пропонується наступний підхід: використовувати допоміжну ER-діаграму для даної ПО, на основі якої проводити побудову діаграми ФЗ, як це показано на рисунку 2.4. Основна вимога при побудові діаграми ФЗ: необхідно, щоб діаграма була зв’язковою, тобто щоб не були присутні ФЗ, не зв'язані з іншими ФЗ через свої елементи (див. рис. 2.4).

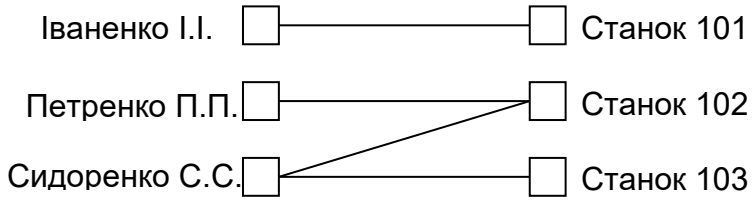
Після побудови діаграми ФЗ необхідно виключити з неї ФЗ, що є надлишковими.

Надлишкова ФЗ несе в собі таку інформацію про ПО, яку можна витягти з інших ФЗ діаграми. Таким чином, при виключенні з діаграми надлишкової ФЗ інформація про ПО загублена не буде. Більш того, надлишкові ФЗ *необхідно виключати* з діаграми, тому що при подальшому проектуванні вони можуть викликати помилки.

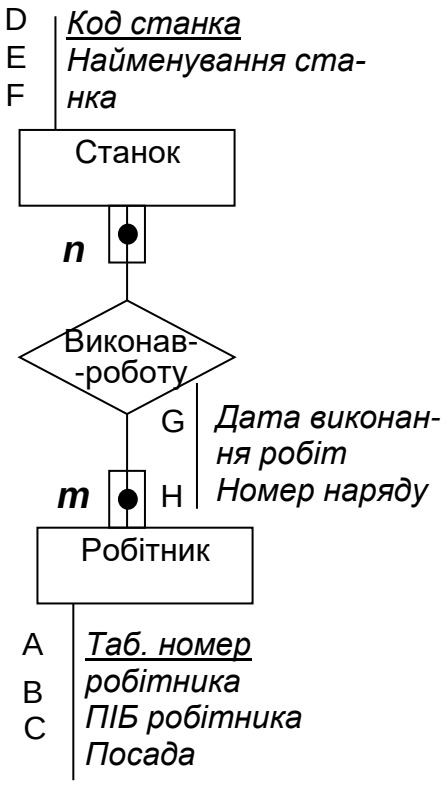
Універсальне відношення для ПО «Облік робіт у цеху»

A	B	C	D	E	F	G	H
Таб. номер робітника	ПІБ Робітника	Посада	Код станка	Найменування станка	Тип станка	Дата виконання робіт	Номер наряду
1	Іваненко І.І.	Токар	101	АІ-1001	Токарний	10.10.2001	505
1	Іваненко І.І.	Токар	102	РІ-2000	Розточувальний	12.10.2001	510
2	Петренко П.П.	Фрезерувальник	102	РІ-2000	Розточувальний	11.10.2001	515
3	Сидоренко С.С.	Токар - фрезерувальник	103	ФІ-300	Фрезерувальний	11.10.2001	520

Діаграма ER-екземплярів для допоміжної ER-діаграми



Допоміжна ER-діаграма для ПО «Облік робіт у цеху»



Діаграма ФЗ для ПО «Облік робіт у цеху»

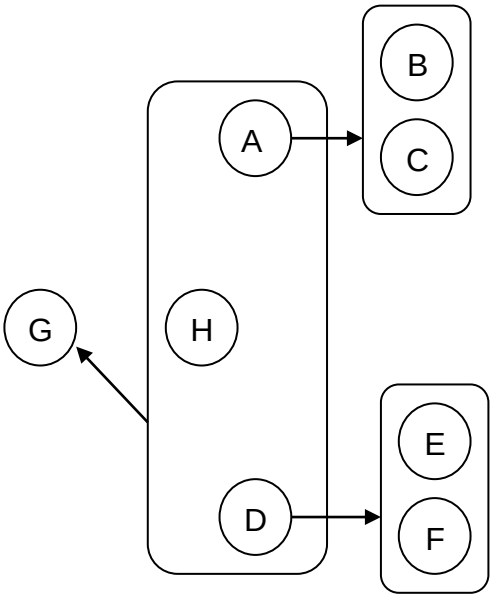


Рисунок 2.4 – Приклад та етапи побудови діаграми ФЗ з використанням допоміжної ER-діаграми

Для визначення надлишкових ФЗ сформульовані *правила виведення ФЗ*, що дозволяють вивести повну множину функціональних залежностей, властивій розглянутій схемі відношення $R(A_1, A_2, ..., A_n)$ на заданій множині атрибутів $U=\{A_1, A_2, ..., A_n\}$ за заданою множиною вхідних функціональних залежностей $F=\{F_1, F_2, ..., F_k\}$. Таким чином, ФЗ, що може бути виведена за зазначеними нижче правилами з уже заданих ФЗ, є надлишковою.

Правило рефлексивності. [$X \in U$; $Y \in U$; $Y \in X$] => [$X \to Y$]. У тому числі, будь-який атрибут (чи група атрибутів) залежить функціонально сам від се-

бе. Залежність буде зберігатися, якщо поповнювати детермінант будь-якою кількістю атрибутів із *U.*

Правило транзитивності. [$X \in U$; $Y \in U$; $Z \in U$; $X \to Y$; $Y \to Z$] => [$X \to Z$].

Правило поповнення. [$X \in U$; $Y \in U$; $Z \in U$; $X \to Y$] => [$X \cup Z \to Y \cup Z$]. Таким чином, поповнюючи тим самим набором атрибутів ліву і праву частину ФЗ, ми можемо одержувати справедливі, але цілком надлишкові ФЗ.

Правило розширення. [$X \in U$; $Y \in U$; $Z \in U$; $X \to Y$;] => [$X \cup Z \to Y$]. Для одержання надлишкових ФЗ можна також розширювати детермінант ФЗ будь-яким набором атрибутів з *U.*

Правило продовження. [$X \in U$; $Y \in U$; $Z \in U$; $W \in U$; $W \in Z$; $X \to Y$] => [$X \cup Z \to Y \cup W$].

Правило псевдотранзитивності. [$X \in U$; $Y \in U$; $Z \in U$; $W \in U$; $X \to Y$; $Y \cup W \to Z$] => [$X \cup W \to Z$].

Правило об'єднання. [$X \in U$; $Y \in U$; $Z \in U$; $X \to Y$; $X \to Z$] => [$X \to Y \cup Z$]. Таким чином можливо об'єднання декількох ФЗ з однаковим детермінантом в одну ФЗ з об'єднаною залежною частиною.

Правило розбивки. [$X \in U$; $Y \in U$; $Z \in U$; $Z \in Y$; $X \to Y$] => [$X \to Z$]. Це правило використовується для доказу оборотності об'єднання ФЗ з однаковими детермінантами.

Після побудови діаграми ФЗ для відношення з'являється можливість за допомогою концепції ФЗ визначити, у якій нормальній формі дане відношення знаходиться. Відповідно, можна також зробити висновок, чи будуть властиві даному відношенню при його використанні в якості БД проблеми вставки, модифікації й видалення кортежів.

Нормальні форми відношень

Нормальні форми відношень – це табличні форми, що відповідають певним обмеженням. Взаємне розташування нормальних форм відношень у просторі табличних форм наведено на рисунку 2.5. Якщо відношення знаходиться в одній (більш строгій) нормальній формі, то воно знаходиться і у всіх інших (менш строгих) формах.

Перша нормальна форма (1НФ).

Відношення знаходиться в 1НФ тоді і тільки тоді, коли значення усіх входячих у неї атрибутів є атомарними. Для роботи мов запитів достатньо, щоб відношення знаходилося в 1НФ. Подальша нормалізація відношень необхідна для реалізації додаткових вимог до БД.

Друга нормальна форма (2НФ).

Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і кожний її неперв-винний атрибут функціонально повно залежить від первинного ключа.

Третя нормальна форма (3НФ).

Відношення знаходиться в 3НФ, якщо воно знаходиться в 2НФ і кожен неперв-винний атрибут нетранзитивно залежить від первинного ключа. Якщо в такім відношенні є інші залежності, крім залежності від ключа, то таке відношення буде мати проблеми при реалізації.

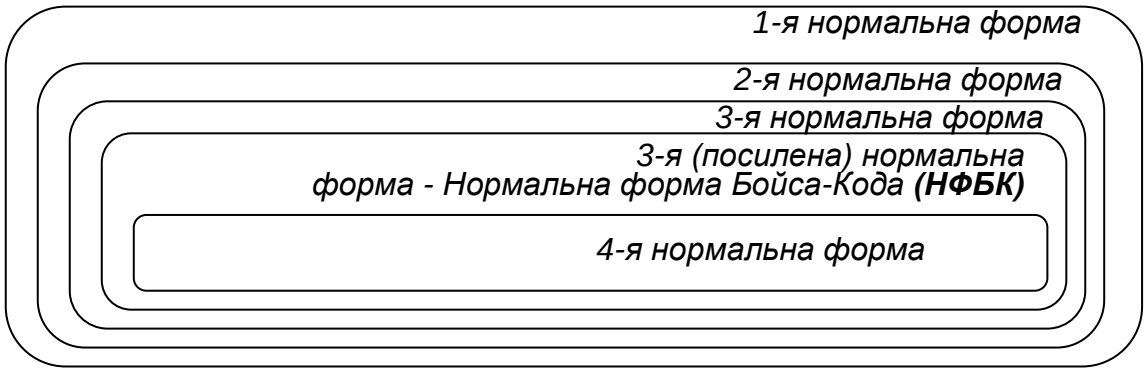


Рисунок 2.5 - Взаємне розташування нормальних форм відношень в просторі табличних форм

Третя посилена нормальна форма (нормальна форма Бойса-Кода - НФБК).

Відношення знаходиться в НФБК, якщо кожен детермінант функціональних залежностей, що є присутнім у діаграмі ФЗ для цього відношення, є можливим ключем. Якщо в такому відношенні відсутні багатозначні залежності, то знімаються всі проблеми при його реалізації.

Якщо у відношенні присутні багатозначні залежності, то відношення, щоб уникнути проблем, повинне знаходиться в четвертій нормальній формі.

Четверта нормальна форма (4НФ).

Відношення *R* знаходиться в 4НФ, якщо всякий раз, коли існує багатозначна залежність $X \to\to Y$ (де *Y* - непорожня множина і вона не є підмножиною *X,* а *XY* складається не з всіх атрибутів *R*), також існує функціональна залежність $X \to A$ для будь-якого атрибута *A* у *R*.

Таким чином, *нормальна форма Бойса-Кода* (НФБК) - така форма відношення, для якого вірно, що кожен детермінант ФЗ у відношенні є можливим ключем.

Для відношення, що знаходиться в НФБК, у більшості випадків відсутні проблеми модифікації, вставки і видалення кортежів.

Нормалізація (приведення до необхідних нормальних форм) виконується шляхом декомпозиції (розбивки на набір відношень).

Одержання наборів відношень із діаграм ФЗ методом декомпозиції.

При проектуванні схеми відношень бази даних методом декомпозиції необхідно виконати наступні етапи.

1 Розробка універсального відношення (УО), що буде містити в собі всі атрибути ПО, значення яких необхідно зберігати в БД.

2 Побудова діаграми ФЗ між атрибутами УО.

3 Одержання мінімального покриття для діаграми ФЗ (видалення надлишкових ФЗ із діаграми за правилами висновку).

4 Для УО, що не знаходиться в НФБК, проведення декомпозиції за наступним алгоритмом.

Алгоритм декомпозиції:

1) для відношення визначається ФЗ, про яке відомо, що воно є перешкодою перебування відношення в НФБК (детермінант ФЗ не є можливим ключем відношення); *при цьому здійснюється вибір ланцюжків ФЗ виду A->B->C, про які відомо, що права частина ланцюжка є перешкодою перебування відношення в НФБК, і*

декомпозицію починають з цієї ФЗ; ця операція над відношенням називається в реляційній алгебрі *проекцією*;

2) створюються два відношення; одне містить всі атрибути визначеного для декомпозиції ФЗ, інше - усі атрибути первісного відношення, крім залежної частини цього ФЗ;

3) перевіряється перебування отриманих відношень у НФБК (шляхом побудови діаграми ФЗ і т.д.); у випадку, якщо одне з відношень не знаходиться в НФБК, для нього повторюють процес декомпозиції за пунктами 1 та 2 алгоритму.

Декомпозиція проводиться ітеративно доти, поки усі відношення в отриманому наборі не будуть знаходитися в НФБК. При проведенні декомпозиції необхідно пам'ятати про необхідність звести кількість відношень до мінімуму.

4) З отриманого набору відношень виключаються відношення, що є надлишковими.

5) Перевіряється наявність у отриманих відношеннях первісного набору ФЗ у мінімальному покритті для УО.

Декомпозиція повинна проводитися *без утрат*, тобто дані й інформація, одержувана з їхньою допомогою, повинні цілком відповідати в початковому і кінцевому наборах відношень. Ця вимога буде виконуватися у випадку, якщо при природному з'єднанні кінцевого набору відношень будуть виходити тільки кортежі початкового набору відношень.

Особливі випадки діаграм ФЗ і декомпозиція без утрат

Інше формулювання обґрунтування вибору ФЗ для декомпозиції виглядає в такий спосіб. *Необхідно при декомпозиції уникати вибору ФЗ, залежна частина якої сама (цілком чи частково) є детермінантом іншій ФЗ.*

Якщо не дотриматися цього правила, то в результаті застосування вищеописаного алгоритму декомпозиції буде загублена ФЗ, тому що вона не буде міститися явно в жодному з отриманих відношень. Це, у свою чергу, призведе до виникнення проблем при їхньому з'єднанні.

Приклад. *{A->B->C} => A->B, A->C.*

Ситуація, коли один атрибут залежить функціонально від декількох, також приводить у результаті застосування вищеописаного алгоритму декомпозиції до утрати ФЗ.

Приклад. *{A->B, C-> B} => A->B, A і C.*

Дана ситуація вимагає застосування алгоритму синтезу, який сформульовано наступним чином. *При декомпозиції необхідно усі ФЗ з однаковими детермінантами виділяти в групи і кожну таку групу виділяти в окреме відношення.*

Алгоритм синтезу може бути використаний як самостійно, так і в сполученні з алгоритмом декомпозиції, як альтернатива при пошуку виходу зі скрутних ситуацій.

Відсутність утрат при декомпозиції відношення R(X,Y,Z) на R1(X,Y) і R2(Y,Z) гарантується за умови, якщо від загального атрибуту двох одержуваних відношень - у даному випадку від Y - залежить хоча б один атрибут із двох, що залишилися (тобто якщо у вхідному відношенні Y->X чи/і Y->Z). У протилежному випадку природне з'єднання приведе до одержання додаткових кортежів, тобто виконання деяких запитів у такій базі даних призведе до одержання невірних даних.

Таким чином, результат проектування БД і з використанням концепції ФЗ, як і при застосуванні ER-діаграм, може привести до різних результатів. Тому невід'ємною частиною процесу проектування є оцінка декількох альтернатив і вибір варіанту, що найбільш відповідає потребам автоматизації ПО.

Порівняльна характеристика двох методів проектування БД і ме-

тодика їхнього спільного застосування.

Для проектування БД *із використанням моделі "сутність-зв'язок"* (ER-діаграм) визначені наступні достоїнства і недоліки.

Переваги:

- інформаційне моделювання, яке здійснюється для ПО й описує процеси, що відбуваються в ній; воно може застосовуватися для системного аналізу взагалі, а не тільки для автоматизації обробки даних;
- моделювання, яке не обумовлює моделей даних, що повинні бути використані в прикладеннях (може бути використана як реляційна, так і не реляційні моделі);
- можливість визначення атрибутів, упущених з уваги, але необхідних для побудови ефективних баз даних;
- швидкість, простота і наочність побудови моделей і одержання з них наборів відношень; отримані відношення вільні від проблем модифікації, вставки і видалення даних.

Недоліки:

- суб'єктивність вибору конструкцій для моделювання одних і тих самих інформаційних компонентів, велика залежність якості проектування від рівня підготовки системотехніка;
- відсутність формалізації і математичного доказу правил одержання наборів відношень;
- неможливість визначення з ER-діаграм деяких обмежень ПО, необхідних для розробки алгоритмів обробки даних.

Для проектування *з використанням концепції функціональних залежностей* (діаграм ФЗ) визначені наступні достоїнства й недоліки.

Переваги:

- строга формалізація і математичний доказ правил одержання мінімального покриття діаграми ФЗ і наборів відношень, вільних від проблем модифікації, вставки й видалення даних.

Недоліки:

- складність і відсутність наочності побудови діаграм ФЗ і одержання набору відношень, високі вимоги до теоретичної підготовки розроблювачів;
- тверда прив'язка процесу проектування до реляційної моделі і задач автоматизації обробки даних;
- громіздкість діаграм і трудомісткість алгоритму для баз даних із великим числом атрибутів (більше двадцяти);
- складність визначення атрибутів, невизначених при аналізі предметної області, але необхідних для побудови ефективних баз даних з раціональною структурою.

Таким чином, два методи проектування структури БД можуть взаємно доповнювати один одного при їхньому спільному використанні. Метод ER-діаграм є слабко формалізованим, використовує творчий потенціал та інтуїцію проектувальника, дозволяє для однієї ПО одержати різні діаграми. Об'єктивного критерію оцінки отриманих діаграм у самому методі немає. Однак цей метод є наочним і доступним неспеціалісту в галузі реляційних БД. Метод діаграм ФЗ цілком формалізований, має чіткі об'єктивні критерії для побудови ефективних діаграм, однак він є складним, ненаочним і малодоступним неспеціалісту в області реляційних БД. Спільне їхнє застосування прискорює процедуру проектування і дозволяє використовувати чіткі критерії оцінки для інформаційних моделей ПО.

Змістовий модуль 2. Теоретичні питання обробки даних та основи розроблення компонентів систем цифрового інтелекту на основі баз даних

Тема 3. Види мов запитів до БД, реляційна алгебра та мова структурованих запитів SQL

3.1 Теоретичні мови запитів і реляційна алгебра

Існують два підходи до обробки даних у БД.

Навігаційна обробка відповідає підходу алгоритмічних мов програмування високого рівня до обробки плоского файлу (набору записів): при її виконанні здійснюється пошук запису за заданим критерієм (наприклад, за фізичним номером), визначається поле (за його номером чи іменем) і обробляється його значення. Результатом такої операції є зчитування в перемінну, запис із перемінної чи модифікація значення поля.

Виконання запитів є підходом, характерним для реляційної моделі даних: відношення розглядаються як множини кортежів і виконуються різні операції над множинами. Операндами, таким чином, є одне чи декілька відношень, а результатом окремої операції є відношення (навіть якщо в цьому відношенні – один стовпець і один рядок).

Існують два шляхи побудови запитів і, відповідно, дві групи мов запитів:

- мови, що описують критерії (властивості), яким повинна задовольняти множина кортежів відношення, отримана в результаті виконання запиту (*мови числення предикатів*); за допомогою таких мов не можна описати набір операцій, які приводять до бажаного результату;
- мови, що описують операції над відношеннями і порядок їх виконання (алгебраїчні мови, представником яких є *реляційна алгебра*); за допомогою таких мов не можна описати критерії (властивості), яким повинна задовольняти одержувана множина кортежів відношення.

Доведено для абстрактних мов запитів, що мови першої й другої груп є однаково виразними.

Розглянемо операції ***реляційної алгебри*** [6]. При визначенні операцій реляційної алгебри передбачається, що порядок стовпців у відношенні фіксований, а самі відношення кінцеві.

Об'єднання відношень R_1 і R_2: $R = R_1 \cup R_2$. Операція застосовується до відношень однієї і тієї ж арності (арність – це кількість доменів, що породжують стовпці). На практиці для об'єднання множин кортежів потрібно також і однакове значеннєве значення стовпців відношень.

Різниця відношень R_1 і R_2: $R = R_1 - R_2$. Різницею $(R_1 - R_2)$ називається множина кортежів, що належать R_1, але не приналежних R_2. Вимоги до арності – такі ж.

Декартовий добуток відношень R_1 і R_2: $R^{(k_1+k_2)} = R_1^{k_1} \times R_2^{k_2}$. Якщо відношення R_1 має арність k_1, а відношення R_2 – арність k_2, то декартовим добутком відношень R_1 і R_2 є множина кортежів арності $(k_1 + k_2)$, причому перші k_1 елементів утворять кортеж з відношення R_1, а останні k_2 елементів – кортеж з відношен-

ня R_2. При цьому для одержання кортежів відношення R виконують усі можливі комбінації кортежів відношень R_1 і R_2 один з одним.

Проекція відношення R_1 на компоненти i_1, i_2, ... , i_r: $R = \pi_{i_1,i_2,...,i_r}(R_1)$.

Операція проекції полягає в тому, що з відношення R_1 вибираються зазначені стовпці і компонуються в зазначеному порядку. Декомпозиція відношень виконується з використанням операції проекції.

Селекція відношення R_1 за формулою F

$$R = \sigma_F(R_1),$$

де F – формула, утворена: 1) операндами, що є номерами чи іменами стовпців; 2) логічними операторами І, ЧИ, НЕ; 3) арифметичними операторами порівняння. Селекція має на увазі добір у результуючий набір кортежів тільки тих кортежів, значення полів у яких задовольняють формулі F.

Наступні операції реляційної алгебри можна одержати за допомогою основних, але вони мають самостійне значення.

Перетинання відношень R_1 і R_2: $R = R_1 \cap R_2 = R_1 - (R_1 - R_2)$.

Частка відношень R_1 і R_2:

$$R = R_1 \div R_2 = \pi_{1,2,...,(n-m)}(R_1) - \pi_{1,2,...,(m-n)}\left((\pi_{1,2,...,(n-m)}(R_1) \times R_2 - R_1)\right),$$

де n – арність відношення R_1, m – арність відношення R_2, $n > m$, $R_2 \neq \varnothing$.

З'єднання відношень R_1 і R_2

$$R = R_1 \underset{i\Theta j}{\triangleright\triangleleft} R_2 = \sigma_{i\Theta(n+j)}(R_1 \times R_2),$$

де Θ – арифметичний оператор порівняння, n – арність відношення R_1, i і j – номера стовпців відповідно у відношеннях R_1 і R_2.

Реальною мовою запитів, що реалізують реляційну алгебру, є мова структурованих запитів SQL.

Розглянуті абстрактні мови запитів є основою реальних мов маніпулювання даними реляційних систем. Кожна з них еквівалентна за своєю виразністю іншим. При цьому мови числення – непроцедурні мови. Їхніми засобами можна виразити усе, що необхідно, але вираз у такій мові описує лише властивості бажаного результату і не вказує, як його одержати. Вирази реляційної алгебри, навпаки, описують конкретний порядок виконання операцій, не описуючи властивостей результату, – вони очікуються.

Для оптимізації запитів у реляційній алгебрі користувач повинний сам оптимізувати порядок і обсяг операцій, що задаються. З іншого боку, якісний компілятор може замінити нераціональний набір операторів, еквівалентних йому, більш оптимальним.

Реальні мови маніпулювання даними повніше абстрактних, тому що вони, крім можливостей абстрактних мов, мають наступні можливості:

- створення таблиць, включення, модифікація й видалення даних;
- арифметичні обчислення й операції порівняння, що входять у вирази реляційної алгебри;
- агрегатні функції (функції, застосовувані до стовпців відношень, у результаті виконання яких обчислюється одна величина, – наприклад, максимальне чи мінімальне значення, сума, середнє);
- команди присвоювання, печаті, настроювання програмного оточення і т.п.

3.2 Основи структурованої мови запитів (SQL) – загальний огляд синтаксису та можливостей

Мова запитів SQL (Structured Query Language) – предметно-орієнтована мова, тобто мова, яка не є мовою програмування загального призначення, а застосовується для виконання завдань у конкретній предметній області, в даному випадку – для формулювання запитів до СУБД з метою виконання операцій над вмістом реляційних баз даних.

Коротка історія розвитку SQL:

- **1970:** Едгар Кодд, науковець з IBM, розробив концепцію реляційної моделі даних, яка стала основою для SQL.
- **1974:** Дональд Чемберлін та Реймонд Бойс з IBM розробили мову SEQUEL (Structured English Query Language) для роботи з реляційними базами даних.
- **1979:** перша комерційна реалізація SQL з'явилася в СУБД Oracle 2.
- **1986:** ANSI (Американський національний інститут стандартів) та ISO (Міжнародна організація зі стандартизації) стандартизували SQL, що сприяло його широкому поширенню.
- **1989:** вийшов стандарт SQL-89 з незначними змінами та доповненнями.
- **1992:** був опублікований стандарт SQL-92, який містив значні розширення та нові функції, такі як підтримка транзакцій та більш складні типи даних.
- **1999:** Вийшов стандарт SQL:1999, який додав підтримку рекурсивних запитів, об'єктно-реляційних можливостей та інших функцій.
- **2003:** стандарт SQL:2003 розширив можливості SQL, додавши підтримку XML та інші функції.
- **2008:** SQL:2008 ввів нові типи даних, такі як DATE та TIME, а також покращив підтримку роботи з часом.
- **2011:** SQL:2011 додав нові функції для роботи з часовими даними та покращив підтримку аналітичних функцій.
- **2016:** SQL:2016 розширив можливості SQL, додавши підтримку поліморфних табличних функцій та інші функції.
- **2023:** SQL:2023 – останній на даний момент стандарт, який продовжує розвивати мову, додаючи нові функції та можливості.

Таким чином, SQL продовжує розвиватися та вдосконалюватися, щоб відповідати сучасним вимогам до обробки та аналізу даних. Ця мова запитів залишається однією з найпопулярніших та найважливіших мов для роботи з базами даних.

Потрібно брати до уваги, що більшість СУБД пропонують використовувати власні діалекти цієї мови (тобто варіанти реалізації синтаксису та набори вбудованих функцій).

Мова складається з декількох складових частин.

Data Control Statements (DCS) складають оператори контролю даних для призначення й перевірки повноважень користувачів, що звертаються до БД – оператори GRANT і REVOKE відповідно.

Data Definition Language (DDL) складають оператори визначення структури й створення об'єктів БД.

CREATE schema – оператор створення схеми БД;
CREATE table – оператор створення таблиці БД;
CREATE view – оператор створення перегляду;
CREATE index – оператор створення індексу таблиці БД;
DROP <об'єкт БД> – оператор видалення зазначеного об'єкту БД;
ALTER table – оператор редагування структури таблиці БД.

Data Manipulation Language (DML) складають оператори пошуку, зміни, видалення, додавання даних, а також виконання операцій реляційної алгебри над таблицями БД.

INSERT – оператор додавання в таблицю запису;

UPDATE – оператор зміни даних у записах таблиці БД;

DELETE – оператор видалення записів із таблиці БД;

SELECT – оператор одержання вибірки даних із зазначених таблиць з необхідними властивостями.

Подальший виклад у даному пункті буде вестися в основному про синтаксис і правила використання оператора SELECT при побудові запитів. Використання інших операторів буде проілюстровано на робочих прикладах.

Особливістю використання SQL при організації прикладень систем БД є залежність його функціональних можливостей від використовуваної СУБД чи формату збереження даних. Відповідно до цього розглядають локальну версію SQL (із скороченими функціональними можливостями) і версію SQL для клієнт-серверних СУБД (повний набір операторів і можливостей). Далі буде розглядатися локальна версія SQL.

Укрупнено текст запиту з використанням оператора SELECT складається з наступних складових частин:

SELECT <перелік стовпців (колонок) результуючої вибірки>
FROM <перелік таблиць-джерел>
WHERE <умова з'єднання таблиць і /чи умова добору записів у вибірку>
ORDER BY <порядок розташування записів вибірки>
GROUP BY <стовпчики, за якими виконується групування>
HAVING <умова добору груп у вибірку>

Розглянемо ***синтаксис оператора SELECT*** (для локальної версії SQL).

```
SELECT [distinct] [<псевдонім>.]<вираз> [as <колонка>] [, [<псевдонім>.]<вираз> [as <колонка>]] [, ...]

FROM [<псевдонім>] <таблиця БД> [, [<псевдонім>] <таблиця БД> ] [, ...]

[WHERE <умова зв'язку> [and <умова зв'язку>] [and/or <умова добору>] [...]]

[GROUP BY <колонка> [, <колонка> [, ...]]]

[HAVING <умова добору груп>]

[ORDER BY <колонка> [ascending/descending] [, ...]]
```

Тут опція distinct не дозволяє включення у результуючу вибірку однакових записів; <псевдонім> – будь-який символьний вираз, що у рамках поточного запиту привласнюється таблиці для зменшення обсягу тексту запиту; <вираз> – поле в таблиці БД, константа, функція (у тому числі агрегатна функція чи функція декількох стовпців); <стовпчик> – стовпець таблиці БД чи вираз, а також символьне ім'я, що привласнюється стовпцю чи виразу.

У якості критеріїв добору записів служать наступні вирази: <умова зв'язку> – застосовується при використанні в запиті декількох таблиць, при цьому реалізується операція *з'єднання* реляційної алгебри (при формулюванні використовуються імена стовпчиків і арифметичних операторів порівняння); <умова добору> – ви-

значає властивості записів, що повинні потрапити в результуючу вибірку, при цьому реалізується операція *селекції* реляційної алгебри.

При формулюванні <умов добору> крім арифметичних операторів порівняння використовуються наступні конструкції:

<вираз> like <шаблон> – дозволяє будувати умову добору по шаблону, якому повинний задовольняти вміст зазначеного поля запису, що відбирається, (у шаблоні символ «_» використовується для позначення одиничного невизначеного символу, символ «%» – для позначення будь-якої їхньої кількості);

<вираз> between <нижня межа значення> and <верхня межа значення> – здійснює перевірку перебування значення зазначеного поля запису, що відбирається, у зазначеному діапазоні;

<вираз1> in (<вираз2>, <вираз3>, ...) – здійснює перевірку входження значення зазначеного поля запису, що відбирається, у зазначений набір значень;

not (<умова добору>) – інвертує умову добору.

Для математичної обробки даних у стовпцях таблиць БД (наприклад, за однойменним полем всіх записів) використовуються ***агрегатні функції***.

count(<вираз>) – визначає число входжень значення <виразу> в усі записи результуючого набору даних;

count(distinct <вираз>) – визначає число входжень значення <виразу> в усі неповторювані записи результуючого набору даних;

avg(<вираз>) – визначає середнє значення <виразу> у всіх записах результуючого набору даних;

avg(distinct <вираз>) – визначає середнє значення <виразу> у всіх неповторюваних записах результуючого набору даних;

sum(<вираз>) – визначає суму числових даних <виразу> у всіх записах результуючого набору даних;

sum(distinct <вираз>) – визначає суму числових даних <виразу> у всіх неповторюваних записах результуючого набору даних;

max(<вираз>) – визначає найбільше значення числових даних <виразу> у всіх записах результуючого набору даних;

min(<вираз>) – визначає найменше значення числових даних <виразу> у всіх записах результуючого набору даних.

Примітка. Якщо в результуючий набір даних виводяться стовпчики з агрегатними функціями разом з неагрегованими колонками (стовпцями таблиць), то в запиті необхідно використовувати опцію GROUP BY для неагрегованих колонок. Імена стовпчиків з агрегатними функціями виглядають у такий спосіб: sum_<вираз>, avg_<вираз> і т.д.

Розглянемо приклади формулювання запитів. Створимо попередньо таблиці БД «Торговий заклад» – таблицю з даними про клієнтів Customers, таблицю з даними про товари Items і таблицю з даними про продажі Sales. Для цього скористаємося операторами розділу DDL мови SQL. Синтаксис оператора для створення таблиці наступний: CREATE table <Ім'я таблиці> (<Ім'я полю> <Тип даних>, ...). Типи даних, оброблювані в SQL: INT – цілочисельні дані; NVARCHAR(<розмір у символах>) – символьні дані; FLOAT – числові дані з крапкою, що плаває; AUTOINC – автоінкрементні (автоматично нарощувані цілочисельні) поля; DATE – поля для збереження дат; BOOL – логічні (бульови) значення; BLOB – великі двійкові об'єкти (зображення й інші дані).

```
CREATE table Customers (CustNumber INT, FirstName
NVARCHAR(30), SurName NVARCHAR(20), Country NVARCHAR(20),
Address NVARCHAR(30), CreditLimit FLOAT);
```

```
CREATE table Items (ItemNumber INT, NameItem NVARCHAR(30),
Unit NVARCHAR(10), Price FLOAT);

CREATE table Sales (CustNumber INT, ItemNumber INT,
SaleDate DATE, Capacity INT).
```

Для заповнення таблиць також скористаємося мовою SQL, наприклад: INSERT into Customers (CustNumber, FirstName, SurName, Country, Address, CreditLimit) values (101, 'Іваненко', 'Іван', 'Україна', 'м. Дніпро, проспект Дм. Яворницького, 112-26', 1500.0) – цей запит додасть у таблицю Customers новий запис і запише у перелічені поля відповідні значення.

Приклад 1. Одержати з таблиці Sales дату, товар і вартість відпущеного товару. При цьому в результуючу вибірку включити тільки записи про продажі, у яких вартість відпущеного товару більше 120 грошових одиниць.

```
SELECT a.SaleDate, b.NameItem, (a.Capacity*b.Price) as
Vartist

FROM Sales a, Items b

WHERE a.ItemNumber=b.ItemNumber and
(a.Capacity*b.Price)>120

ORDER BY a.SaleDate
```

Приклад 2. Одержати загальну вартість кожного з відпущених товарів на кожну дату.

```
SELECT b.ItemNumber, a.ItemName, b.SaleDate,
sum(b.Capacity*a.Price)

FROM Items a, Sales b

WHERE a.ItemNumber=b.ItemNumber

GROUP BY b.ItemNumber, a.ItemName, b.SaleDate
```

Приклад 3. Одержати кількість покупців на кожну дату продажів.

```
SELECT SaleDate, count(distinct ItemNumber)

FROM Sales

GROUP BY SaleDate
```

Примітка. Для випадків, коли потрібно одержати агреговані значення не для всієї таблиці, а для кожної з вхідної в результат групи записів, що характеризуються однаковим значенням якого-небудь поля (групи полів), використовується угруповання записів.

Приклад 4. Видати загальну кількість відпущеного товару в одиницях виміру з кожного товару, що продається.

```
SELECT b.ItemNumber, b.NameItem, sum(a.Capacity) as Kilkist

FROM Sales a, Items b

WHERE a.ItemNumber=b.ItemNumber

GROUP BY b.ItemNumber, b.NameItem,
```

Приклад 5. Підрахувати загальну кількість покупців, які хоч раз купували товари.

```
SELECT count(distinct CustNumber) as Count_Cust
FROM Sales
```

Приклад 6. Обчислити загальну вартість товарів, відпущених між 10.10.2024 і 10.11.2024.

```
SELECT b.ItemNumber, b.NameItem, sum(a.Capacity*b.Price) as
Vartist
FROM Sales a, Items b
WHERE a.ItemNumber=b.ItemNumber and a.SaleDate between
‘10.10.2024’ and ‘10.11.2024’
```

Приклад 7. Визначити коди покупців, у яких обмеження кредиту менше, ніж у середньому у всіх покупців.

```
SELECT CustNumber
FROM Customers
WHERE CreditLimit< (SELECT avg(CreditLimit) FROM Customer)
```

Примітка. Для випадків, коли необхідно для добору записів використовувати критерій, значення якого динамічно змінюється в залежності від умісту БД, то організовують вкладені запити (підзапити).

Використання опції HAVING.

Якщо в результуючій вибірці необхідно одержати агрегатні дані не з всіх груп, а тільки з тих, що відповідають деяким умовам, після GROUP BY застосовують наступну опцію:

```
HAVING <агрегатна функція> <арифметичний оператор порівняння>
<значення>
```

Необхідно враховувати, що умови WHERE і HAVING мають наступні розходження:

- HAVING виключає з результату групи, не задовольняючі умові, а WHERE виключає з розгляду і формування результату записи, що не задовольняють умові;
- в умові добору WHERE не можна вказувати агрегатні функції.

Приклад 8. Одержати загальну кількість купленого товару для всіх покупців, у яких мінімальна кількість товару, що купується, не менше 100 одиниць.

```
SELECT CustNumber, sum(Capacity)
FROM Sales
GROUP BY CustNumber
HAVING min(Capacity)>=100
```

Приклад 9. Одержати дати відпустки товарів, у які кількість товару, що відпускався, була більше 1000 одиниць. Включити в результуючу вибірку тільки ті групи, за якими таких продажів було більш однієї.

```
SELECT SaleDate, count(*)
FROM Sales
WHERE Capacity>=1000
```

```
GROUP BY SaleDate
HAVING count(*)>1
```

Використання квантора існування

У випадках, коли необхідно відбирати тільки ті записи, для яких підзапит повертає одне чи більш значень, використовується квантор існування exists з наступним синтаксисом:

```
WHERE exists (SELECT ... )
```

Приклад 10. Одержати дані про покупців, хоча б раз придбавших товар у даній організації.

```
SELECT a.CustNumber, a.FirstName
FROM Customers a
WHERE exists (SELECT b.CustNumber FROM sales b WHERE
a.CustNumber=b.CustNumber)
```

Об'єднання результатів виконання декількох запитів виконується за допомогою оператора UNION, при цьому результуючі вибірки повинні мати однакову структуру (склад стовпців). Записи при об'єднанні вибірок не дублюються.

Приклад 11. В одну вибірку вивести дані про товари, що або мають ціну вище 1000 одиниць, або куплені покупцем з кодом 123.

```
SELECT ItemNumber
FROM Items
WHERE price>1000
UNION
SELECT b.ItemNumber
FROM Customers a, Items b, Sales c
WHERE a.CustNumber=c.CustNumber and
c.ItemNumber=b.ItemNumber and a.CustNumber=123
```

Тема 4. Принципи розробки програмних компонентів для обробки даних в БД з використанням об’єктно-орієнтованих засобів розробки.

4.1 Формулювання мети і задач прикладень. Визначення бізнес-логіки й обмежень на дані

Розрізняють прикладення баз даних трьох типів:

- *системи обробки транзакцій*; для роботи з такими прикладеннями необхідний рівень доступу як для читання, так і для запису; системи такого типу відповідають за введення й обробку даних, використовуваних прикладеннями високого рівня;

- *системи прийняття рішень*; припускають рівень доступу – тільки для читання; використовуються керуючим персоналом для огляду деякої частини даних, приготовлених за допомогою низькорівневих систем;

- гібридні системи; з'єднують у собі ознаки двох попередніх типів.

Розробці баз даних і їхніх прикладень передує, як було показано вище, аналіз ПО й інтересів можливих користувачів. Після цього проводиться формулюван-

ня мети і задач програмного комплексу. Мета прикладення (програмного комплексу) необхідно сформулювати наскільки це можливо коротко і лаконічно, одним реченням. Приклад: програмний комплекс повинний забезпечувати облік робіт з обслуговування орендованого нерухомого майна. Другий приклад: прикладення повинне забезпечувати облік і автоматизоване зачислення абітурієнтів у ВУЗ. З урахуванням мети (мета при цьому служить для обмеження при постановці задач) формулюються задачі прикладення (укрупнено). Прикладення БД звичайно вирішують чотири групи задач:

1) забезпечувати введення даних і перевірку їхньої коректності;

2) забезпечувати подання (перегляд) даних;

3) виконувати ряд запитів до БД і подавати відповідним чином їх результати;

4) генерувати звіти (тверді копії документів, використовуваних у паперовій технології). Докладне визначення задач і вимог до функціональних можливостей та інтерфейсу забезпечується шляхом розробки технічного завдання.

Повторимо ***основні етапи проектування і реалізації БД*** у формулюванні, використовуваної для прикладень БД:

- інформаційне і логічне моделювання і проектування бази даних, визначення бізнес-логіки обробки даних;

- розробка схеми структури бази даних з використанням моделі даних обраної СУБД чи формату збереження даних і обраного засобу розробки;

- визначення обмежень на збережені дані (типи збережене даних, входження в діапазон значень, входження в обмежену множину значень, облік залежностей між даними) і внесення їх в опис структури бази даних (якщо обрана СУБД чи формат збереження це припускає);

- формулювання необхідних форм, запитів і звітів; побудова ієрархії класів (якщо це можливо і необхідно);

- реалізація таблиць БД і коду прикладення за допомогою ЯОД і ЯМД обраних засобів реалізації відповідно;

- розробка і підключення довідкової підсистеми, розробка посібника користувача, розрахунково-пояснювальної записки проекту.

Забезпечення надійності функціонування прикладень баз даних

При розробці програмного забезпечення необхідно реалізовувати захід щодо забезпечення надійності спільного функціонування прикладень і користувача, а також цілісності даних. Потрібно дотримувати наступних принципів:

1 Усі меню і всі діалогові вікна реалізовувати у відповідності зі стандартами Common User Access, при цьому діалогові елементи повинні бути чіткими і недвозначними, що відповідають ергономічним вимогам до типів і розмірів шрифтів, до колірних гамів; реалізовувати інтуїтивно зрозумілу для користувача реакцію на його дії (бажано дотримувати угод операційної системи).

2 Контролювати послідовність взаємодії користувача з елементами інтерфейсу, у загальному випадку створювати контекстно-залежний інтерфейс, а також створювати і використовувати контекстно-залежну допомогу.

3 Забезпечувати відмовлення користувача від виконуваних дій у будь-який момент часу, на будь-якому етапі їхнього виконання, з відновленням вхідного стану БД чи даних користувача.

4 Передбачати в програмі постійний вхідний контроль вводу даних (контроль типів даних і логічний контроль, контроль входження в діапазон, контроль за допомогою шаблонів і т.д.).

5 Підтримувати унікальність записів і посилальну цілісність (для систем які вимагають програмної підтримки цих режимів).

6 Потенційно небезпечні дії супроводжувати попереджуючими повідомленнями і діалоговими вікнами для вибору варіантів реакції.

7 Забезпечити введення тих самих даних однократно, при цьому бажано організовувати введення первинних даних і обов'язково введення повторюваних даних шляхом вибору з меню, списків, що розгортаються, та ін.

8 Організовувати добре прооблену обробку помилок для продовження коректної роботи прикладення у випадку виникнення виняткових ситуацій (руйнування індексів, відсутності файлів і каталогів, спроб доступу до уже відкритих файлів, обробки некоректних даних, нестачі місця на нагромаджувачах і ін.).

9 Організовувати періодичне (чи за запитом) страхове копіювання даних і їх архівацію.

4.2 Засоби розробки і СУБД, використовувані при організації БД

Обґрунтування вибору СУБД чи засобу розробки для реалізації прикладення баз даних. СУБД чи засіб розробки обирається за наступними ознаками:

- можливість забезпечити підтримку обраної при проектуванні БД моделі даних (практично в кожному випадку – реляційної моделі);
- підтримка посилальної цілісності даних і унікальності ключів (індексів);
- підтримка явних обмежень на дані, що вводяться, включаючи перевірку обмежень і обробку виняткових ситуацій (або видачу попереджуючих повідомлень і запобігання запису даних, не задовольняючих обмеженням);
- наявність діалекту мови запитів SQL, що дозволяє здійснювати ефективну обробку даних за запитами користувачів;
- розвинута мова маніпулювання даними для організації навігаційної обробки даних;
- розвинута мова опису даних з великим словником даних для опису необхідних схем структур даних;
- підтримка засобів візуального програмування, використання різних генераторів і майстрів для автоматизації програмування;
- підтримка сумісності форматів файлів даних або сучасних технологій доступу до даних.

Локальні бази даних, не підтримуючі транзакції, паралельна обробка даних і видалений доступ у даний час використовуються порівняно мало. Розробка і створення таких баз даних проводяться при автоматизації малих офісів, для реалізації малих Internet-магазинів чи рішення невеликих технічних задач. Ринок СУБД, що підтримують локальні БД, досить вузький. Усі ці СУБД підтримують реляційну модель даних, мають убудовані мови програмування користувальницького інтерфейсу і навігаційної обробки даних, а також версії так називаної локальної SQL (діалекту мови SQL з істотно урізаними функціональними можливостями). Вони відрізняються лише форматом збереження даних, синтаксисом мови маніпулювання даними, методами організації користувальницького інтерфейсу. Формати збереження даних локальних СУБД використовуються засобами розробки (Visual Studio, VS Code, Embarcadero) за допомогою драйверів ODBC (Open DataBase Connectivity) для організації локальних баз даних. Потужні інтегровані середовища програмування на одній з об’єктно-орієнтованих мов (C#, Python, Visual Basic, C++, Object Pascal) використовують бібліотеки низькорівневого доступу до БД, що замінюють, власне, СУБД, а також ієрархії класів, які дозволяють організувати прикладення з розвинутим користувальницьким інтерфейсом і можливостями обробки даних

Пріоритетний розвиток одержують великі бази даних масштабу підприємства з централізованим збереженням і паралельним доступом до даних. Зараз на ринку подані шість основних виробників ***промислових*** систем управління базами даних.

Сервери баз дані фірми ORACLE. У СУБД використовується мова опису й маніпулювання даними PL/SQL, що дозволяє також визначати тригери і збережені процедури, викликувані сервером автоматично при виникненні відповідних подій. Використовується ряд архітектурних рішень, спрямованих на підвищення ефективності сервера: буферизація відкомпільованих SQL-операторів на сервері баз даних, використання статистичної інформації для оптимізації запитів та ін.. У восьмій версії системи убудовані засоби для використання в Internet/Intranet, підтримка збереження мультимедійної інформації, підтримка об'єктно-реляційної схеми. Основними достоїнствами серверів ORACLE є висока надійність, здатність до масштабування, розвинуті програмні засоби і засоби адміністрування. ORACLE займає близько 60% ринку.

Сервери баз дані фірми Microsoft. Останні версії Microsoft SQL Server мають могутні програмні й адміністративні засоби (підтримуються ANSI SQL і власні процедурні мовні засоби Transact SQL), розвинуті засоби проектування схем баз даних і убудованих засобів OLAP (On-Line Analisis Processing). За продуктивністю СУБД не уступає серверам ORACLE.

Сервер *Interbase компанії Inprise* має відкриту архітектуру, БД переносні практично на всі платформи. Область застосування – організація так званих убудованих систем, що не вимагають спеціального адміністрування і підтримки.

4.3 Основи розробки систем баз даних для багатьох користувачів

Принципи паралельної обробки даних у базах даних

Транзакція – логічна одиниця роботи програмного забезпечення над БД, що переводить БД з одного цілісного стану в інше. СУБД та інші програмні комплекси, що підтримують механізм транзакцій, забезпечують гарантію того, що, якщо в ході відновлень, зроблених транзакцією, відбулася помилка, то усі відновлення будуть анульовані. Визначено наступні властивості транзакцій: 1) атомарність (усе чи нічого); 2) погодженість (транзакції переводять БД з одного цілісного стану в інший без збереження проміжних); 3) ізоляція (паралельні транзакції не впливають на роботу одне одного); 4) довговічність (відновлення, зроблені транзакцією, зберігаються).

У системі, що підтримує механізм транзакцій, є присутнім відповідний компонент, називаний адміністратором транзакцій і здійснюючий їхнє виконання за наступною схемою (наведено псевдокод):

```
...
   begin transaction;
   ...
   insert ({code_customer: 100, first_name: 'Іванов'}) into
Customer;
   ...
   if <виникнення помилки> then goto Undo;
   commit transation;
   goto Finish
Undo:
   rollback transaction;
Finish:
   return;
...
```

де псевдооператор begin повідомляє про початок транзакції, commit – про успішне закінчення (так називана крапка фіксації, або кінець логічної одиниці роботи), а rollback – про невдале завершення транзакції і про необхідність скасування

всіх зроблених у ході поточної транзакції відновлень. Відновлення цілісності даних при цьому забезпечується в такий спосіб. Періодично (у черговій за часом контрольній точці) програмна система (частіше СУБД) записує стан БД і дані про транзакції у журнал транзакцій. При відмовленні системи здійснюється аналіз проведених транзакцій, визначаються потребуючі відкочування (скасування внесених змін) і виконується їхнє відкочування, визначаються транзакції, що вимагають перезапуску, і виконується їхній перезапуск.

Технології «файл-сервер» і «клієнт-сервер». Реалізація багаторівневих прикладень

Бази даних на персональних комп'ютерах розвивалися за напрямком від настільних (desktop), чи локальних прикладень, коли реально з БД могло працювати одне прикладення, до систем колективного доступу до БД.

Локальне прикладення установлювалося на одиничному персональному комп'ютері; там же розташовувався і БД, з яким працювало дане прикладення. Однак необхідність колективної роботи з однієї і тієї ж БД спричинила за собою перенос БД на мережний сервер. Прикладення, що працювало з БД, розташовувалося також на сервері. Були випущені відповідні версії локальних СУБД, що дозволяли створювати прикладення, що одночасно працюють з однієї БД на файловому сервері. Основною проблемою була явна чи неявна обробка транзакцій і неминуче встає при колективному доступі проблема забезпечення значеннєвої і посилальної цілісності БД при одночасній зміні тих самих даних.

У ході експлуатації таких систем були виявлені загальні недоліки ***файл-серверного підходу*** при забезпеченні паралельного доступу багатьох користувачів до БД. Вони полягають в наступному:

- уся вага обчислювального навантаження при доступі до БД лягає на прикладення клієнта (при видачі запиту на вибірку інформації з таблиці *вся* таблиця БД копіюється на клієнтське місце, і вибірка здійснюється на клієнтському місці);
- локальні СУБД використовують навігаційний підхід до обробки даних, орієнтований на роботу з окремими записами;
- не оптимально витрачаються ресурси клієнтського комп'ютера і мережі (зростає мережний трафік і збільшуються вимоги до апаратних потужностей користувальницького комп'ютера);
- оскільки під БД розуміється набір окремих таблиць, що співіснують у єдиному каталозі на диску, це дає низький рівень безпеки – як з погляду розкрадання і нанесення шкоди, так і з погляду внесення помилкових змін;
- бізнес-правила в системах "файл-сервер" реалізуються в прикладенні, що дозволяє в різних прикладеннях, що працюють з однієї БД, проектувати взаємовиключні бізнес-правила; значеннєва цілісність інформації при цьому може порушуватися;
- недостатньо розвинутий апарат транзакцій для локальних СУБД служить потенційним джерелом помилок.

Наведені недоліки зважуються при перекладі прикладень з архітектури *"файл-сервер"* в ***архітектуру "клієнт-сервер"***, що знаменує собою наступний етап у розвитку СУБД. Характерною рисою архітектури "клієнт-сервер" є перенос обчислювального навантаження на *сервер БД (SQL-сервер)* і максимальне розвантаження прикладення клієнта від обчислювальної роботи, а також істотне зміцнення безпеки даних.

БД у цьому випадку міститься на мережному сервері, як і в архітектурі "файл-сервер", однак прямого доступу до БД із прикладень не відбувається. Функції прямого звертання до БД здійснює спеціальна керуюча програма – сервер БД (SQL-сервер), що поставляється розроблювачем СУБД.

Таким чином, SQL-сервер – це програма, розташована на комп'ютері мережного сервера. SQL-сервер повинен бути завантажений на момент прийняття запиту від клієнта. Функціями сервера БД є: 1) прийом запитів від прикладень-клієнтів, інтерпретація запитів, виконання запитів у БД, відправлення результату виконання запиту прикладенню-клієнту; 2) керування цілісністю БД, забезпечення системи безпеки, блокування невірних дій прикладень-клієнтів; 3) збереження бізнес-правил, часто використовуваних запитів у вже інтерпретованому вигляді; 4) забезпечення одночасної безпечної й відмовостійкої роботи багатьох користувачів з тими самими даними.

Взаємодія сервера БД і прикладення-клієнта відбувається в такий спосіб: 1) клієнт формує SQL-запит і відсилає його серверу; 2) сервер, прийнявши запит, виконує його і результат повертає клієнту; 3) у клієнтському прикладенню здійснюється інтерпретація отриманих від сервера даних, реалізація інтерфейсу з користувачем і введення даних, а також реалізація частини бізнес-правил. Схема архі-

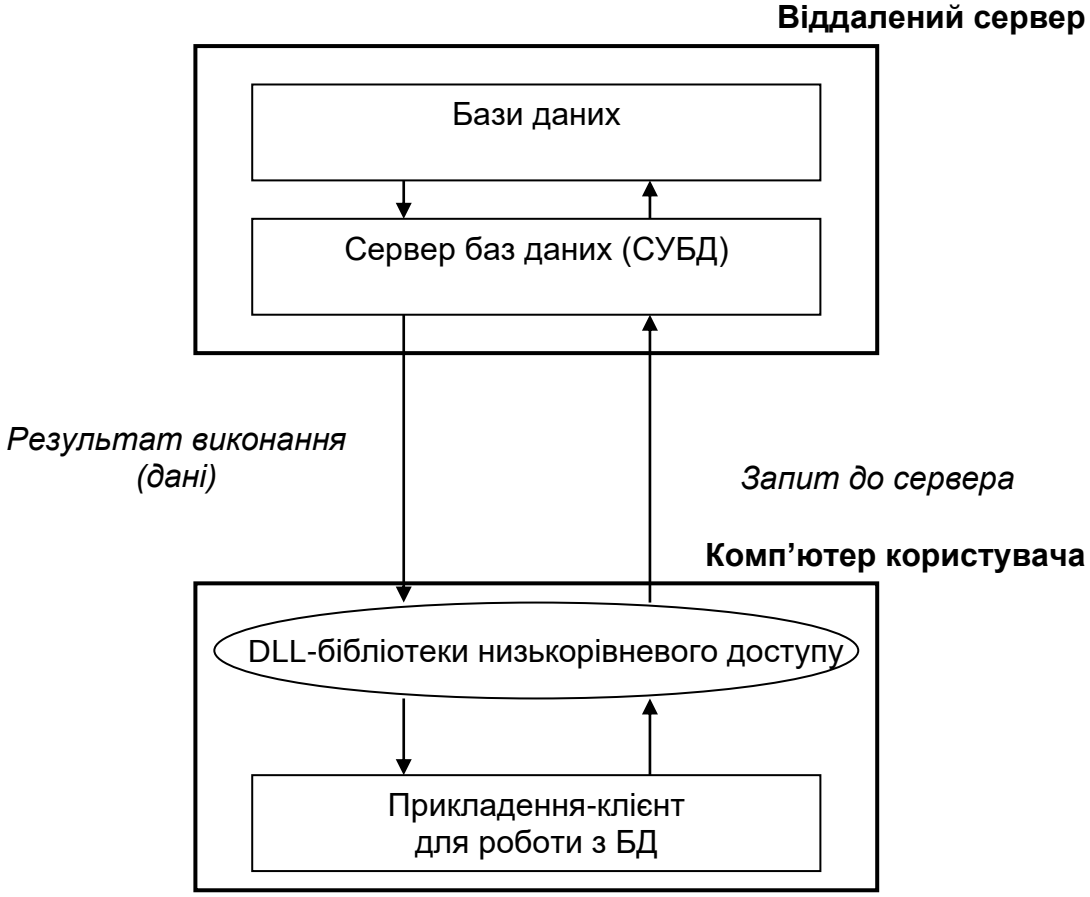


Рисунок 4.1 - Структурна схема багатоланкової архітектури "клієнт-сервер"

тектури наведена на рисунку 4.1.

Переваги архітектури "клієнт-сервер":

- більшість обчислювальних процесів відбувається на сервері: у такий спосіб знижуються вимоги до обчислювальних потужностей комп'ютерів клієнтів;

- спрощується нарощування обчислювальних потужностей в умовах розвитку програмного забезпечення і зростання обсягів оброблюваних даних;

- знижується мережний трафік за рахунок посилки сервером клієнту тільки тих даних, що він запитував;

- підвищується безпека даних, забезпечується посилальна й значеннєва цілісність

інформації, тому що БД на сервері являє собою, як правило, єдиний файл, у якому містяться таблиці БД, обмеження цілісності й інші компоненти БД; доступ до нього можливий тільки через СУБД;
- безпека системи зростає за рахунок переносу більшої частини бізнес-правил на сервер; падає питома вага суперечних один одному бізнес-правил у клієнтських прикладеннях, що виконують різні дії над БД;
- сервер реалізує керування транзакціями і запобігає спроби одночасної зміни тих самих даних; різні рівні ізоляції транзакцій дозволяють визначити поводження сервера при виникненні ситуацій одночасної зміни даних.

Для реалізації архітектури застосовують промислові СУБД, такі як InterBase, Oracle, Informix, Sybase, DB2, MS SQL Server. Як правило, SQL-сервер керується окремим співробітником чи групою співробітників (адміністратори SQL-сервера). Вони керують фізичними характеристиками баз даних, роблять оптимізацію, настроювання і перевизначення різних компонентів БД, створюють нові БД, змінюють існуючі й т.д., а також видають привілею (дозволу на доступ визначеного рівня до конкретних БД, SQL-серверу) різним користувачам.

Крім цього, існує окрема категорія співробітників, називаних адміністраторами баз даних. Як правило, це адміністратори сервера, розроблювачі БД чи користувачі, що мають привілеї на створення, зміну, настроювання оптимальних параметрів окремих серверних БД. Адміністратори БД також відповідають за надання прав на різнорівневий доступ до супроводжуваного ними БД для інших користувачів.

При переносі прикладень, раніше розроблених для застосування в архітектурі "файл-сервер", потрібно не тільки частково чи цілком переписувати прикладення клієнтів, але і перетворювати локальну БД у серверну. Для цього під керуванням серверної СУБД (наприклад, MS SQL Server) створюють БД на сервері, куди потім переносять дані з локальних СУБД, реалізованих, наприклад, за допомогою MS Access. Основна проблема, що встає в цьому випадку – несумісність деяких форматів даних чи їхня відсутність. При виникненні подібних проблем варто вивчити питання сумісності типів даних локальної СУБД і обраної серверний СУБД.

Багатоланкова архітектура "клієнт-сервер"

Розвиток ідеї архітектури "клієнт-сервер" привело до появи багатоланкової архітектури доступу до баз даним (у літературі її також називають трьох-ланковою архітектурою, N-tier чи multi-tier архітектурою).

Архітектура "клієнт-сервер" є двох-ланковою. Першою ланкою є прикладення клієнта, другим - сервер БД і сама БД. У трьох-ланкової архітектурі набори даних, колишні раніше "власністю" клієнтських прикладень, виділяються в окрему ланку, називана сервером прикладень (див. рисунок 4.2).

Використання контейнерів *Data Module* при цьому має та перевага, що компоненти типу *"набір даних",* один раз розміщені в *Data Module,* можуть використовуватися у всіх формах прикладення. При цьому компоненти типу *"набір даних"* (*TTable, TQuery, TStoredProc*), розміщені в *Data Module,* мають єдине поводження

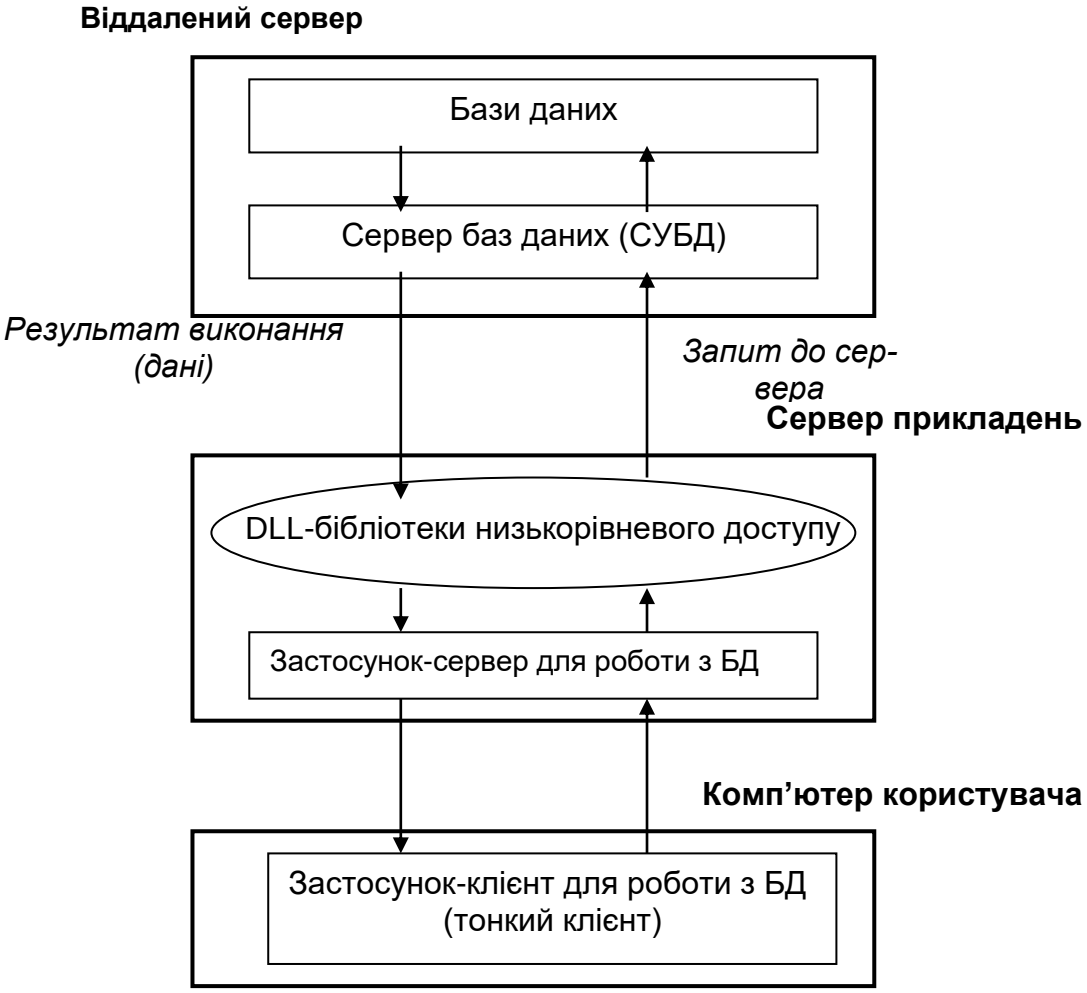


Рисунок 4.2 - Структурна схема багатоланкової архітектури "клієнт-сервер"

для всього прикладення. Це є особливо важливим при реалізації бізнес-правил, що зберігаються в прикладеннях клієнта.

Крім того, модулі даних, експортовані в репозиторій, можуть використовуватися при розробці інших прикладень. Це забезпечує повторне використання один раз розробленого програмного коду та гарантує однакове поводження наборів даних у всіх прикладеннях, що використовують той самий модуль даних з репозиторію.

Отже, модулі даних у трьох-ланкової архітектурі "клієнт-сервер" виділяються в окремий *"сервер прикладення".* Разом з ним розташовується BDE. Тепер, при зміні бізнес-правил немає необхідності змінювати прикладення клієнтів і обновляти їх у всіх користувачів-клієнтів, як це було раніше, коли частина бізнес-правил зберігалася в прикладенні клієнта.

Сервер прикладень розділяється декількома клієнтами. Він формує запит до вилученого БД (тобто до SQL-сервера). На ньому розташовані реальні набори да-

них (у двох-ланковій архітектурі "клієнт-сервер", що розташовувалися в прикладенні клієнта). У клієнтському прикладенні розміщається "клієнтський набір даних" (компонент *TClientDataSet*). Він являє собою локальну копію даних із сервера прикладень. Таким чином, усі зміни, внесені користувачем у дані за допомогою клієнтського прикладення, вносяться в локальну копію НД. При відновленні видаленого НД клієнтське прикладення посилає серверу прикладень тільки змінилися записи. Сервер прикладень, у свою чергу, відсилає ці зміни SQL-серверу, що вносить їх у вилучену БД.

Взаємодія клієнтського прикладення (у багатоланковому прикладенні називаного "тонким клієнтом", thin client) із сервером прикладень здійснюється за допомогою так званих "брокерів даних" (наприклад, компоненти *TRemoteServer*, що розташовуються в прикладенні тонкого клієнта, і *TProvider*, що розташовуються на сервері прикладень).

4.4 Основи розробки програмних компонентів систем баз даних з використанням технології ADO.NET

Більшості розробників та користувачів може знадобитись виконувати роботу з базами даних (БД), розташованими або локально на клієнтських машинах або на віддалених серверах. Для цих цілей у складі бібліотеки FCL є набір класів простору імен `System.Data` (і пов'язаних з ним), який називається технологією ADO.NET. Дана технологія надає прості в застосуванні, але потужні засоби доступу до даних, за допомогою яких можна максимально повно використовувати ресурси системи.

Дана технологія дозволяє реалізувати два режими роботи з даними: приєднаний і від'єднаний режими. У приєднаному режимі додаток відкриває з'єднання з БД і не закриває його до завершення роботи. Однак, такий режим постійного з'єднання з СУБД, є незручним з наступних причин:

- підтримання з'єднання з СУБД вимагає використання системних ресурсів: чим більше відкритих з'єднань доводиться підтримувати, тим нижче продуктивність системи;
- додатки, що використовують доступ до даних через постійне з'єднання, дуже погано масштабуються. Такий додаток добре обслуговує з'єднання з двома клієнтами, насилу справляється з 10 і зовсім не працює з 100 клієнтами.

В ADO.NET ці проблеми вирішуються за допомогою від’єднаного режиму роботи з БД. В цьому режимі з'єднання з джерелом даних відкрито тільки до завершення необхідних дій над даними. Наприклад, якщо додаток запитує дані з БД, з'єднання встановлюється тільки на час завантаження даних, після чого відразу ж закривається. Аналогічно при оновленні БД з'єднання відкривається на час виконання команди `UPDATE`, А потім закривається. Підтримуючи з'єднання відкритими, протягом мінімально необхідного часу, ADO.NET економно використовує системні ресурси і дозволяє масштабувати інфраструктуру доступу до даних - при цьому продуктивність знижується незначно.

Архітектура ADO.Net показана на рис. 4.3. Тут під провайдером даних розуміється набір класів, що виконують функції посередника при взаємодії програми і бази даних та ефективно реалізують основні операції роботи з БД.

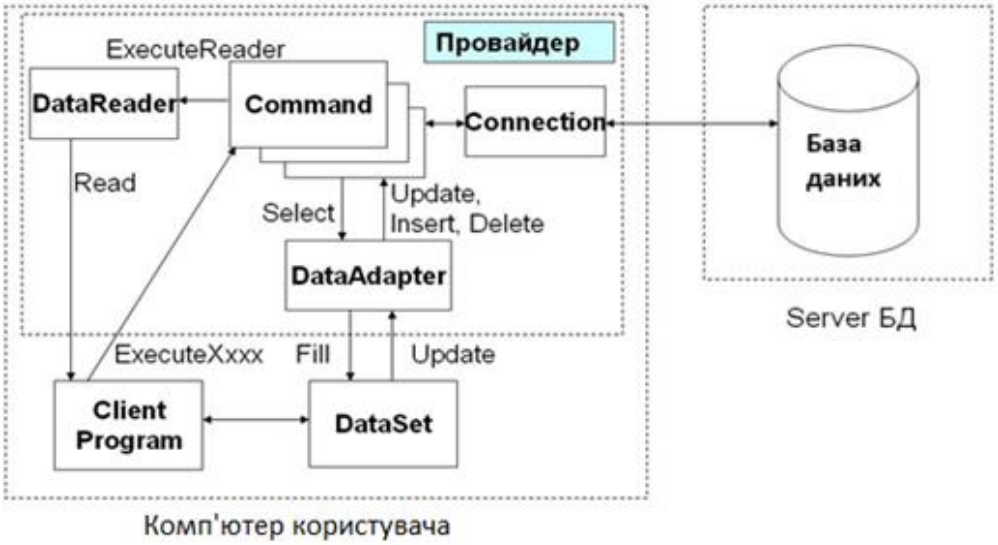


Рисунок. 4.3. Основні класи ADO.Net.

Нині з .NET Framework поставляються 4 провайдера даних:

- SQL Server .NET Data Provider - використовується для роботи з SQL Server версії 7.0 і вище (`System.Data.SqlClient`);
- OleDb .NET Data Provider - використовується для роботи з БД за допомогою COM технології OLEDB (`System.Data.OleDb`);
- Odbc Data Provider - використовується для роботи з БД за допомогою старої технології ODBC, заснованої на драйвери баз даних (`System.Data.Odbc`);
- Oracle Data Provider - використовується для роботи з базами даних Oracle (`System.Data.OracleClient`).

Будь-який провайдер включає власні реалізації наступних універсальних класів (рис. 4.3):

- `Connection` - забезпечує підключення до БД;
- `Command` - застосовується для управління джерелом даних; дозволяє виконувати команди, що не повертають даних, наприклад `INSERT`, `UPDATE` і `DELETE`, або команди, повертають об'єкт `DataReader` (такі, як `SELECT`);
- `DataReader` - надає доступний тільки для односпрямованого читання набір записів, підключений до джерела даних;
- `DataAdapter` - заповнює від'єднаний об'єкт `DataSet` або `DataTable` і оновлює його вміст.

У різних провайдерів до назв цих класів додаються префікси, такі як `Sql`, `OleDb`, `Odbc`, `Oracle` (для перерахованих вище провайдерів). Якщо назва класу провайдера наводиться без префіксу, що вказує його версію, то сказане в рівній мірі відноситься до всіх версій провайдерів даних. наприклад, `Command` означає як `OleDbCommand`, так і `SqlCommand`.

Приєднаний режим роботи з БД реалізується за допомогою тільки провайдерів даних (Data Provider). Доступ до даних здійснюється наступним чином, об'єкт `Connection` встановлює з'єднання між додатком і БД. За допомогою об'єкта `Connection` безпосередньо взаємодія з БД виконують об'єкти `Command`. Даний об'єкт дозволяє виконувати команди до БД. Якщо виконана команда повертає кілька записів, то `Command` відкриває до них доступ через

об'єкт DataReader. використовуючи об'єкт DataReader можна послідовно прочитати і зберегти в програмі всі отримані з БД записи.

Від'єднаний режим роботи з базою даних реалізується за допомогою класу DataSet. Об'єкти даного класу містять вибірки даних з різних таблиць бази даних в оперативній пам'яті комп'ютера, ізольовано від реальної БД. Такі об'єкти можна також розглядати як локальні копії фрагментів БД.

Дані в об'єкт DataSet найпростіше завантажити за допомогою об'єкта DataAdapter, який керує набором команд (запитів до БД) для виконання вибірки, зміни, видалення і додавання нових даних. об'єкт DataSet зберігається в пам'яті, з його вмістом можна виконувати різні дії (читати, змінювати, видаляти, додавати) незалежно від БД. Всі зміни зроблені локально в об'єкт класу DataSet може передати в серверну БД за допомогою об'єктів DataAdapter.

4.4.1 Опис класів провайдерів даних

Провайдери даних включають такі основні класи:

Клас Connection - задає і реалізує з'єднання з БД. Всі дані, необхідні для відкриття каналу зв'язку з БД, зберігаються у властивості ConnectionString, в якій записується рядок підключення до базе даних. Основними методами даного класу є: Open() – відкриття з'єднання і Close() – закриття з'єднання.

Клас Command - реалізує виконання команд до БД, використовуючи для обміну даними відкрите з'єднання. За допомогою об'єктів Command можна виконувати команди SQL, збережені процедури (Stored procedure), а також оператори, які повертають окремі таблиці. об'єкт Command підтримує три методи:

- ExecuteNonQuery()- виконує команди, що не повертають дані, наприклад INSERT, UPDATE і DELETE;
- ExecuteScalar()- виконує запити до БД, повертають єдине значення;
- ExecuteReader()- повертає результуючий набір через об'єкт DataReader.

Клас DataReader - надає доступ до вибірці записів БД, доступний тільки для односпрямованого читання. На відміну від інших компонентів провайдера даних, створювати екземпляри DataReader безпосередньо не можна, їх можна отримувати тільки за допомогою методу ExecuteReader() об'єкта Command. Якщо записувати вміст об'єкта DataReader на диск не потрібно, рядки з даними можна передавати з додатком безпосередньо, оскільки в будь-який момент часу в пам'яті знаходиться тільки один рядок, використання об'єкта DataReader майже не знижує продуктивність системи, але вимагає монопольного доступу до відкритого об'єкту Connection протягом часу життя об'єкта DataReader.

Клас DataAdapter - основний клас, що забезпечує формування відокремлених даних. По суті, він виконує функції посередника у взаємодії між БД і об'єктом DataSet. При виклику методуFill() об'єкт DataAdapter заповнює об'єкт DataTable в DataSet даними, отриманими з БД. Після обробки даних, завантажених в пам'ять, можна перенести змінені записи в БД, викликавши метод Update() об'єкта DataAdapter. У класу DataAdapter є чотири властивості, які представляють команди БД:

- SelectCommand - містить текст або об'єкт команди, здійснює вибірку даних з БД; при виклику методуFill() ця команда виконується і заповнює об'єкт DataTable або об'єкт DataSet;
- InsertCommand - містить текст або об'єкт команди, здійснює вставку рядків в таблицю;
- DeleteCommand - містить текст або об'єкт команди, здійснює видалення рядка з таблиці;
- UpdateCommand - містить текст або об'єкт команди, здійснює оновлення значень в БД.

При виклику методу Update() всі змінені дані переносяться з об'єкта DataSet в БД з виконанням відповідної команди InsertCommand, DeleteCommand або UpdateCommand.

4.4.2 З'єднання з базою даних

Для з'єднання з базою даних потрібно створити об'єкт відповідного класу Connection. Задати рядок підключення до БД – властивість ConnectionString. А потім викликати метод Open() - для виконання з'єднання. Після закінчення використання даного з'єднання потрібно викликати його метод Close()для звільнення ресурсів сервера. наприклад:

```
//Оголосити клас OleDbConnection і створити екземпляр.
OleDbConnection myCon = new OleDbConnection ();
//Створити рядок підключення, в якій заданий тип БД.
//БД Microsoft Access і вказано шлях до файлу БД.
myCon.ConnectionString =
@ "Provider = Microsoft.Jet.OLEDB.4.0; DataSource =" + @ "З: \
Northwind \ Northwind.mdb";
myCon.Open();
//. . .
myCon.Close();
```

Слід зазначити, що набір параметрів задаються в рядку підключення може змінюватися в залежності від типу і конфігурації провайдера даних, тому рекомендується по можливості створювати з'єднання за допомогою графічних інструментів Visual Studio .NET.

Наприклад, для з'єднання з базою даних Northwind СУБД SQL Express, яка встановлена на комп'ютері Master рядок з'єднання може виглядати наступним чином:

```
@ "Data Source = Master \ SQLEXPRESS; Initial Catalog =
Northwind; Integrated Security = True"
```

В цьому рядку вказується, що використовується єдина обліковий запис і в ОС і в базі даних - Integrated Security = True.

4.4.3 Виконання команд для роботи з даними

Для виконання команд в СУБД використовуються об'єкти класу Command. Ці об'єкти містять посилання на оператор SQL або збережену процедуру БД і можуть виконати ці команди в СУБД, використовуючи заданий відкрите з'єднан-

ня (об'єкт Connection). об'єкти Command дозволяють дуже швидко і ефективно виконувати наступні види взаємодії з різними БД:

- команди SQL SELECT (вибірка записів), і процедури, які повертають набір записів (вибірку);
- команди SQL, що не повертають значення, наприклад INSERT (Вставка записів), UPDATE (Оновлення записів) і DELETE (Видалення записів);
- команди SQL, які повертають єдине значення;
- команди мови зміни структури бази даних (Database Definition Language, DDL), наприклад CREATE TABLE або ALTER.

Створення команди

Для виконання команди до БД необхідно створити екземпляр об'єкта Command відповідного типу і задати значення його властивостям. Властивість CommandType визначає тип команди, що міститься у властивості CommandText, Воно може приймати одне з наступних значень перерахування CommandType:

- Text - в цьому випадку властивості CommandText, повинен бути заданий текст допустимої команди SQL;
- StoredProcedure - в цьому випадку в властивість CommandText має бути занесено ім'я процедури, яка буде викликатися на виконання;
- TableDirect - в цьому випадку в властивість CommandText має бути задано ім'я таблиці БД і при виконанні даної команди будуть повернуті всі стовпці і рядки цієї таблиці.

Властивості Connection слід задати посилання на відповідний об'єкт класу Connection.

Виконання команд

Для виконання команди можна використовувати три методи класу Command: ExecuteNonQuery(), ExecuteScalar() і ExecuteReader(). Всі ці методи виконують команду, задану в властивості CommandType і CommandText. Відрізняються ці методи тільки повертається значенням.

Метод ExecuteNonQuery () - найпростіший з методів виконання команди, він повертає тільки кількість оброблених записів БД. Даний метод застосовують для виклику команд SQL і збережених процедур, таких, як INSERT, UPDATE або DELETE, а також команд зміни схеми БД - DDL.

Метод ExecuteScalar () повертає тільки значення першого поля першого рядка отриманої вибірки, незалежно від того, скільки рядків було отримано.

Метод ExecuteReader () повертає об'єкт DataReader, який дозволяє виконувати послідовне читання даних від початку до кінця вибірки. Якщо не потрібно модифікувати вміст БД, то цей спосіб отримання даних є найшвидшим і ефективним.

Формування запитів SQL під час виконання

Іноді зміст оператора запиту SQL стає відомим тільки в період виконання. Наприклад, запит може використовувати рядок пошуку, введену користувачем, або повертати стовпці або таблиці, певні програмно в період виконання. Для вирішення цих завдань відповідні команди SQL повинні створюватися, налаштовуватися і виконуватися динамічно, під час виконання програми.

Для такого динамічного формування SQL команд описують заготовку тексту команди, і додають до неї рядкові змінні, які заміщуються відповідними ряд-

ками в період виконання програми. Для об'єднання рядків використовуйте оператор конкатенації "+", наприклад:

```
string Cmd = "SELECT * FROM Employees WHERE Name = '" +
     aString + "'";
```

Відзначимо, що будь-які рядкові змінні, що передаються СУБД за допомогою конструкції `WHERE` оператора `SELECT`, необхідно укладати в одинарні лапки ('). Якщо одинарні лапки знаходяться всередині такої змінної, слід замінити їх парою одинарних лапок (''), В іншому випадку, виконання запиту закінчиться невдачею.

Наступний приклад демонструє динамічне формування команди `DELETE`, в яку вставляється значення поля `Name` для визначення записів, що видаляються:

```
// Цей приклад передбачає наявність об'єкта myConnection
// рядок aStr містить значення поля Name public
void DeleteRecord (string aStr)
{
  string Cmd;
  Cmd = "DELETE * FROM Employees WHERE Name = '" +
    aStr + "'";
  OleDbCommand myCommand =
    new OleDbCommand(Cmd, myConnection);
// залишилося відкрити з'єднання і виконати команду,
myConnection.Open(); myCommand.ExecuteNonQuery();

// Не забудьте закрити з'єднання!
myConnection.Close();
}
```

Робота з параметрами команди

При виконанні команди до бази даних часто потрібно задавати параметри, при цьому значення деяких параметрів часто стають відомими тільки в період виконання. Наприклад, в додатку для обліку товарів в книжковому магазині слід передбачити функцію пошуку книг за назвою, яку можна реалізувати за допомогою запиту до БД з використанням наступного оператора SQL:

```
SELECT * FROM Books WHERE (Title LIKE [value]).
```

При цьому значення `value` має задаватися користувачем і заздалегідь воно не відомо. У зв'язку з цим необхідно мати певний спосіб передачі введених значень оператору SQL і збереженим процедурам під час виконання програми.

Параметри - це значення, якими заповнюють поля підстановки, введені в текст команди під час розробки. Параметри є об'єктами класу `Parameter`, які повинні додаватися до властивості (колекції) `Parameters` об'єкта `Command`. У період виконання значення параметрів зчитуються з цієї колекції і підставляються в SQL-оператор, або передаються збереженій процедурі. У колекцію `Parameters` повинні заноситися об'єкти класу `Parameter` відповідного типу (наприклад, `SqlParameter` і `OleDbParameter`). На об'єкти колекції `Parameters` можна посилатися за індексом, або за ім'ям, заданому властивіс-

тю ParameterName. Нижче наведено два способи установки значення першого за рахунком параметра з ім'ям myParameter.

```
// встановлюємо значення параметра за індексом
DleDbCommand1.Parameters [0] .Value = "Іваненко A.I.";
// встановлюємо значення параметра за ім'ям
OleObCommand1.Parameters [ "myParameter"]. Value =
     "Група 8551";
```

В класі OleDbParameter є взаємопов'язані властивості DbType і OleDbType. Перше представляє тип параметра в платформі .Net, а друге - як він представлений в БД; це необхідно, оскільки не всі БД сумісні з типами платформи. Об'єкт Parameter виконує перетворення параметрів з типу, використовуваного в додатку, в тип, який використовується в БД. Оскільки ці властивості взаємопов'язані, при зміні значення одного з них, значення іншого автоматично змінюється і перетворюється у відповідний підтримуваний тип. Аналогічним чином пов'язані властивості DbType і SqlType об'єктів SqlParameter, властивість SqlType вказує тип БД SQL, представлений цим параметром.

Властивість Direction класу Parameter вказує, чи є цей параметр вхідним або вихідним. Можливими значеннями цієї властивості є: Input (вхідні), Output (вихідні), InputOutput (вхідні та вихідні) або ReturnValue (повертається в якості результату).

Властивості Precision, Scale і Size - визначають точність і розмір значення параметрів. Властивості Precision і Scale застосовуються з числовими і десятковими параметрами і визначають розрядність і довжину дробової частини значення властивості Value, відповідно, а властивість Size застосовується з двійковими і строковими параметрами і представляє максимальну довжину такого поля. Властивість параметра Value містить значення параметра.

Якщо властивість CommandType об'єкта Command встановлено в Text, необхідно передбачити поля підстановки для всіх параметрів оператора SQL.

У об'єктів класу OleDbCommand поля підстановки позначаються символами «?», наприклад:

```
SELECT Empld, Title, FirstName, LastName FROM
Employees WHERE (Title =?)
```

Можна вказувати декілька параметрів:

```
SELECT Empld, Title, FirstName, LastName
FROM Employees WHERE (FirstName =?) AND (LastName =?)
```

В цьому випадку, порядок заповнення параметрів визначається порядком елементів колекції Parameters.

У об'єктів класу SqlCommand застосовують іменовані параметри. Для створення поля підстановки іменованого параметра, необхідно вказати ім'я параметра (так як воно задано властивістю ParameterName), випередивши його символом «@». Нижче показаний приклад оператора SQL, в якому оголошено поле підстановки для іменованого параметра Title:

```
SELECT EmpId, Title, FirstName, LastName FROM Employees WHERE
(Title = @Title)
SqlParameter parameter = new SqlParameter ();
parameter.ParameterName = "@Title";
parameter.SqlDbType = SqlDbType.NVarChar;
parameter.Direction = ParameterDirection.Input; ...
```

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Базові

1. Сагайда П. І., Зорі А. А., Тарасов О. Ф. Організація комп'ютерних систем для інтелектуальної обробки даних на основі опрацювання формалізованих знань : монографія. Краматорськ : ДДМА, 2020. 191 с.
2. Bagui S. S., Earp R. W. Database Design Using Entity-Relationship Diagrams. Third Edition. CRC Press, 2023. 388 p.
3. Foster E. C., Godbole S. V. Database Systems. A Pragmatic Approach. Third Edition. CRC Press, 2023. 622 p.
4. Coronel C., Morris S. Database Systems: Design, Implementation, and Management. 14th Edition. Cengage Learning, 2024. 818 p.
5. Byrne G. Target C#: Simple Hands-On Programming with Visual Studio. Apress, 2022.

Додаткові

6. Сагайда П. І., Зорі А. А. Компоненти комп'ютерних систем інтелектуальної обробки даних на основі категоріально-онтологічних моделей : монографія. Краматорськ : ДДМА, 2019. 159 с.
7. Харів Н. О. Бази даних та інформаційні системи : навчальний посібник. Рівне : НУВГП, 2018. 127 с.
8. Лосєв М. Ю., Федько В. В. Бази даних : навчально-практичний посібник для самостійної роботи студентів. Харків : ХНЕУ ім. С. Кузнеця, 2018. 233 с.
9. Бардус І. О., Лазарєв М. І., Ніценко А. О. Бази даних у схемах (на основі фундаменталізованого підходу). Харків : Вид-во "Диса плюс", 2017. 133 с.
10. Ben-Gan I. T-SQL Fundamentals. Microsoft Press, 2023.
11. Edet T. ADO.NET: Building Secure and Scalable Data Access. Independently Published, 2024. 275 p.
12. Fuzzy Logic in Intelligent System Design / P. Melin et al. Springer, 2018.
13. Dimitrakas N. Introduction to MS Access 2016. University of Stockholm & Royal Technical University, 2016.

Web-ресурси

14. Introduction to Databases : Coursera : веб-сайт. URL : https://www.coursera.org/learn/introduction-to-databases (дата звернення: 10.10.2024).
15. Проектування реляційних баз даних : Coursera : веб-сайт. URL: https://www.coursera.org/learn/relational-database-design (дата звернення: 10.10.2024).
16. Введення в мову структурованих запитів (SQL) : Coursera : веб-сайт. URL: https://www.coursera.org/learn/intro-sql (дата звернення: 10.10.2024).
17. SQL: практичний вступ до запитів до баз даних : Coursera : веб-сайт. URL: https://www.coursera.org/learn/sql-practical-introduction-for-querying-databases (дата звернення: 10.10.2024).
18. Мова структурованих запитів (SQL) : Coursera : веб-сайт. URL: https://www.coursera.org/learn/the-structured-query-language-sql (дата звернення: 10.10.2024).
19. SQL Tutorial : веб-сайт. URL: https://www.w3schools.com/sql/default.asp (дата звернення: 10.10.2024).
20. Free courses, tutorials, videos, and more about learning the programming language C# : веб-сайт. URL: https://dotnet.microsoft.com/en-us/learn/csharp (дата звернення: 10.10.2024).
21. C# Tutorial : веб-сайт. URL: https://www.w3schools.com/cs/index.php (дата звернення: 10.10.2024).

ДОДАТОК А. Перелік питань і завдань до іспиту (заліку) з дисципліни «Бази даних»

1 Роль інформації в сучасному світі. Поняття "інформація" і "дані". Семантика даних і проблеми інтерпретації даних у БД.
2 Поняття бази даних, бази знань і банку даних, формула, що ілюструє їхній взаємозв'язок. Поняття системи управління базами даних (СУБД). Основні компоненти архітектури СУБД і взаємовідносини запитів користувачів із БД.
3 Поняття моделі даних. Модель даних алгоритмічних мов і мов СУБД. Перелічіть використовувані в даний час при проектуванні БД моделі даних.
4 Коротка характеристика реляційної моделі даних. Особливості (обмеження) і достоїнства реляційної моделі даних. Сучасний стан ринку СУБД.
5 Рівні абстракції в БД і розходження сприйняття БД на різних рівнях абстракції. Етапи проектування БД.
6 Суттєвість інформаційного підходу до проектування БД. Три сфери при інформаційному підході до проектування БД. Елементарна ситуація, елементарне повідомлення.
7 Модель "сутність-зв'язок" і її основне призначення. Загальний підхід до побудови моделей предметної області, поняття "сутність", "атрибут", "зв'язок".
8 Ступінь зв'язку, клас приналежності зв'язку. Діаграми ER-екземплярів і ER-типу. Навести приклади ER- діаграм.
9 Локальні зображення предметної області. Моделювання локальних зображень. Загальні вимоги до складання проектних схем (угоди про графічні зображення, загальні правила використання конструкцій і т.д.). Навести приклад.
10 Вибір ідентифікуючих атрибутів для кожної сутності; ключі та вторинні ключі.
11 Об'єднання моделей локальних представлень за допомогою похідних від зображень, застосованих у локальних конструкціях; мета, яка досягається при цьому.
12 Концепція агрегації, використовувана при об'єднанні локальних зображень. Навести приклади.
13 Концепція узагальнення, використовувана при об'єднанні локальних зображень. Навести приклади.
14 Структури даних реляційної моделі: поняття відношення, домену, кортежу. Навести приклади.
15 Структури даних реляційної моделі: поняття декартового добутку доменів; арності кортежів і відношень; реляційної бази даних.
16 Реляційна база даних як сукупність відношень. Первинні ключі, вторинні ключі й індексовані файли.
17 Мети проектування БД. Дубльовані й надлишково дубльовані дані. Проблеми видалення, вставки і модифікації. Навести приклад відповідних відношень.
18 Функціональні залежності (ФЗ) як один з видів залежностей між атрибутами. Графічне зображення ФЗ. Навести приклади ФЗ.
19 Ключі сутностей і відношень. Можливі ключі та детермінант функціональної залежності.
20 Нормальні форми відношень. 1-я нормальна форма (1НФ), 3-я посилена нормальна форма (інакше нормальна форма Бойса-Кодда - НФБК) і обмеження, що накладаються на відповідні відношення. Причина, за якою ця форма використовується при проектуванні БД.

21 Загальний підхід (алгоритм) побудови БД із використанням ER-методу й одержання набору попередніх відношень безпосередньо з ER-діаграм. Правила одержання попередніх відношень для бінарних зв'язків "один-до-одного".
22 Алгоритм побудови БД із використанням ER-методу й одержання набору попередніх відношень безпосередньо з ER-діаграм. Правила одержання попередніх відношень для бінарних зв'язків "один-до-багатьох" і "багато-до-багатьтох".
23 Використання при побудові моделей «сутність-зв'язок» зв'язків більш високого порядку, чим бінарні (на прикладі потрійних зв'язків - зв'язків між трьома сутностями) і ролей. Ситуації, коли необхідно застосувати ці конструкції. Правила одержання попередніх відношень для таких конструкцій з ER-діаграм.
24 Поняття нормалізації і декомпозиції. Поняття проекції відношення - формулювання і математична формалізація. Правило одержання проекції при виключенні функціональної залежності, яка порушує обмеження для НФБК.
25 Альтернативний метод проектування БД - метод декомпозиції. Загальний підхід (алгоритм) до проектування методом декомпозиції. Умови, при яких можливе застосування цього методу для проектування.
26 Правило вибору функціональної залежності (ФЗ) для здійснення декомпозиції. Проблеми, що виникають при утраті ФЗ при здійсненні декомпозиції, і ситуації, коли це відбувається. Метод синтезу, який виключає виникнення подібних проблем.
27 Надлишкові ФЗ. Мінімальне покриття в діаграмах ФЗ і спосіб його одержання. Правила висновку (властивості) рефлексивності, транзитивності, псевдотранзитивності, декомпозиції. Навести приклади висновку (виключення надлишкових ФЗ) за цими правилами.
28 Надлишкові ФЗ. Мінімальне покриття в діаграмах ФЗ і спосіб його одержання. Правила висновку (властивості) поповнення, розширення, продовження. Навести приклади висновку (виключення надлишкових ФЗ) за цими правилами.
29 Надлишкові ФЗ. Мінімальне покриття в діаграмах ФЗ і спосіб його одержання. Правила висновку (властивості) поповнення, розширення, продовження, адитивності (об'єднання) і декомпозиції. Привести приклади висновку (виключення надлишкових ФЗ) за цими правилами.
30 Запити до БД. Види теоретичних мов запитів до БД. Реляційна алгебра і її призначення. Операції реляційної алгебри: різниця, об'єднання,
31 Запити - довільні функції над відношеннями. Характеристики теоретичних мов запитів. Порівняння мов числення й алгебраїчних мов (реляційної алгебри). Додаткові можливості мов маніпулювання даними в реляційних СУБД у порівнянні з теоретичними мовами.
32 Операції реляційної алгебри: об'єднання, різниця і декартовий добуток. Математична формалізація цих операцій і приклад їхньої реалізації над відношеннями.
33 Операції реляційної алгебри: проекція, селекція і перетинання. Математична формалізація цих операцій і приклад їхньої реалізації над відношеннями.
34 Операції реляційної алгебри: частка, з'єднання відношень і природне з'єднання. Математична формалізація цих операцій і приклад їхньої реалізації над відношеннями.

35 Вимоги до прикладень систем баз даних, реалізація яких забезпечує надійність їхньої роботи і збереження цілісності даних.
36 Забезпечення захисту даних у системах БД. Безпека даних - керування доступом, ідентифікація користувача і т.д.
37 Забезпечення захисту даних у системах БД. Таємність даних - питання шифрування і т.д.
38 Забезпечення захисту даних у системах БД. Цілісність даних - види і приклади обмежень цілісності і т.д.
39 Забезпечення захисту даних у системах БД. Відновлення даних - поняття транзакції, адміністратор транзакцій, анулювання результатів виконання транзакцій при рівнобіжній обробці даних у БД, проблеми при рівнобіжному виконанні транзакцій, рівні ізоляції транзакцій.
40 Організація паралельних процесів обробки даних. Поняття транзакції; елементи, на які розділяється БД і необхідність їхнього блокування для збереження цілісності даних; вибір розміру елементів з погляду продуктивності роботи БД;
41 Нескінченні чекання і тупики при паралельній обробці даних у БД: умови виникнення цих станів і методи боротьби з ними.

ДОДАТОК Б. Види практичних завдань до іспиту (заліку) з дисципліни «Бази даних»

1 Розробіть інформаційну модель і побудуйте діаграму ER-типу для зазначеної предметної області. Зв'язки між сутностями повинні бути цілком специфіковані (визначені ступені і класи приналежностей). Кожній сутності повинне бути привласнено не менш трьох-чотирьох атрибутів, один із яких повинний бути ідентифікуючим (ключовим). Перевірте отримані діаграми на предмет відсутності залежностей між неідентифікуючими атрибутами.

2 Одержіть набір відношень реляційної бази даних з даної діаграми ER-типу. Зробіть по кілька записів (кортежів) в отриманих схемах відношень. Перевірте отримані відношення на відповідність нормальній формі Бойса-Кодда (на відсутність проблем видалення, вставки і модифікації).

3 Розгляньте дану у якості індивідуального завдання таблицю. Чи є ця табличная форма відношенням з погляду реляційної моделі даних? Яким перетворенням її необхідно піддати, щоб одержати відношення в 1-й нормальній формі? Одержіть відношення, визначить функціональні залежності між атрибутами в отриманому відношенні і перевірте це відношення на відповідність нормальній формі Бойса-Кодда. Проведіть при необхідності декомпозицію відношення шляхом одержання проекцій.

4 Одержіть мінімальне покриття для даної діаграми функціональних залежностей шляхом виключення надлишкових ФЗ, визначених за допомогою правил висновку. Укажіть, якими правилами ви користалися.

5 Напишіть SQL-запит для одержання з набору таблиць, поданих в індивідуальному завданні, зазначених даних.

Навчально-методичне видання

Павло Іванович Сагайда
Олександр Анатолійович Костіков

Конспект лекцій з дисципліни «Бази даних»

Самостійне електронне мережеве видання

Публікується в авторський редакції