Міністерство транспорту та зв'язку України
Державний департамент з питань зв'язку та інформатизації України

ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ЗВ'ЯЗКУ ім. О. С. Попова

Кафедра обчислювальної техніки та мікропроцесорів

Чорний Ю.П.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ ТА СЕРВІСНИХ СЛУЖБ ЗВ'ЯЗКУ

Конспект лекцій

з дисципліни
"Програмне забезпечення інформаційних та сервісних служб зв'язку"
для спеціальності 7.092402

ЗАТВЕРДЖЕНО
методичною радою академії
Протокол № 8
від 9.03.2004 р.

Одеса 2004

УДК 681.3.06 План НМВ 2004 р.

Чорний Ю.П. Програмне забезпечення інформаційних та сервісних служб зв'язку. Конспект лекцій. – Одеса: Вид. центр ОНАЗ ім. О.С.Попова, 2005. – 180 стор.

Викладаються основні питання навчальної дисципліни "Програмне забезпечення інформаційних та сервісних служб зв'язку": принципи побудови програмного забезпечення баз даних, розміщених у локальних, корпоративних та глобальних мережах зв'язку, засоби використання системи Oracle у інформаційних та сервісних службах зв'язку; мови програмування SQL, PL/SQL; керування об'єктами даних, маніпулювання даними, керування транзакціями; ефективне використання запитань, використання можливостей SQL для ефективного пошуку та роботи з базами даних.

Призначений для самостійної роботи студентів, які навчаються за спеціалізацією "Програмне забезпечення мереж зв'язку".

За редакцією – **О.С. Антонова**
Рецензент – **О.В. Онацький**
Редактор **– І.В. Ращупкіна**
Редагування та комп'ютерне редагування – **Т.В. Кірдогло**

СХВАЛЕНО
на засіданні кафедри
обчислювальної техніки
та мікропроцесорів
і рекомендовано до друку
Протокол № 4
від 12 грудня 2003 р.

ЗМІСТ

1 РЕЛЯЦІЙНІ СИСТЕМИ БАЗ ДАНИХ В ІНФОРМАЦІЙНИХ ТА СЕРВІСНИХ СЛУЖБАХ

1.1 Основні поняття та визначення. Oracle – реляційна система керування базами даних

1.1.1 Поняття про базу даних

Минулими роками здійснено чимало спроб дати означення поняття *база даних*. Для нашої мети цілковите задовольнятиме таке: *база даних – це сукупність даних, зорганізована з певною метою.* Тут з*організована* означає, що зазначена сукупність включає дані, які *зберігаються*, мають певний формат *(відформатовані), до* них може бути забезпечено доступ *(доступні)* й вони можуть надаватися споживачеві інформації в припустимому вигляді *(репрезентативні).* У означенні згадка про *мету* підкреслює, що склад даних має відповідати певному завданню: з одного боку, в базі не повинні міститися дані, стосовні не маючи відношення до завдання, а з іншого – в ній має розміщуватися вся інформація, стосовна завдання. За чудовим та зрозумілим прикладом бази даних може слугувати телефонний довідник. Він містить дані, які мають безпосереднє відношення до його призначення (у даному разі – імена абонентів), які відкривають доступ до номера телефону. У довіднику відсутня надлишкова інформація (скажімо, колір апарата абонента), тобто в цій простій базі містяться відомості, безпосередньо пов'язані з метою її створення.

Найчастіше призначення бази даних є пов'язане з діловою сферою, але в ній може міститися наукова, військова й інша інформація, що вона в низці випадків не є стосовна до сфери бізнесу. Існують ділові, наукові, військові й інші бази даних, причому список можливих областей людської діяльності, в яких набувають застосовування дані, зорганізовані в такий спосіб, може бути продовжено практично нескінченно. Окрім поділу баз даних залежно від областей їхнього застосування, можна запропонувати й інший підхід, узявши за підґрунтя формати долученої до них інформації. Окрім текстової й числової інформації, або вміщеної зображення, різного роду графіки й схеми, аудіо- та відеоінформація. Останнім часом все частіше йдеться про *складені документи*, які є зв'язною сукупністю різних видів інформації.

Коли йдеться про бази даних взагалі і проектування баз даних зокрема, узвичаєно посилатися на мету, якою зумовлено їхню організацію й функціонування, незалежно від того, в якій області людської діяльності конкретна база даних використовуватиметься – буде то чи космонавтика, біологія чи будь-що інше. Більш того, насправді виявляється, що власне цільова функція визначає специфіку тієї чи іншої бази даних.

Сучасні бази даних зазвичай, слугують для зберігання та опрацьовування інформації, стосовної різних аспектів діяльності як підрозділів (доволі потужних чи дрібних, що вони складаються всього з кількох співробітників),

так і організацій. Отже, ми можемо використовувати термін *база* даних *рівня підприємства (enterprisewide database),* коли йдеться про інформацію, яка охоплює діяльність підприємства в цілому, або – база даних *рівня підрозділу (departmentide database),* якщо відповідна інформація не виходить за межі його діяльності, або – база даних *рівня робочої групи (workgroup database) для* найдрібніших колективів. Найбільш поширено бази даних двох останніх рівнів – рівня підрозділу й рівня робочої групи.

Іноді можна зустріти бази даних рівня підприємства, наприклад відомості на видання зарплатні чи штатний розпис, але при цьому надто невеликого обсягу. Фактично, коли бази даних окремих підрозділів поєднуються *(інтегруються)* до однієї великої бази, можна говорити про *банк даних (data warehouse – DW).* Бази даних меншого обсягу, які слугують за джерело інформації для потужніших баз даних, останнім часом доволі часто називають *операційними базами* даних (*operational databases).* Проте в цьому немає нічого нового. Те, що сьогодні називається операційною базою, раніше було відоме як *продукована база* даних, оскільки в межах такої бази готувалися дані, які потім долучалися до потужнішої бази даних. У контексті формування банка даних DW можна зустріти згадку про продуковану базу як про операційну, а іноді – як про накопичувач *даних (data store).* З розвиненням Internet мережі бази даних та банки даних все частіше стають прикінцевими пунктами маршруту пошуку інформації, який розпочинається від Web-броузера.

Коли база даних робочої групи інтегрується для обслуговування потреб бази вищого рівня (рівня підрозділу), тоді те, що виходить в внаслідок цього, часто називають *сховищем даних (data mart – DM).* Фактично DM є банком даних рівня підрозділу. У Вас не має спричинювати здивування той факт, що термін *банк даних,* як і термін *база даних,* може мати безліч тлумачень. Коли кілька невеликих за масштабами баз даних об'єднуються до однієї потужної, яка може обслуговувати набагато ширше коло запитів, що забезпечується вищим рівнем організаційної структури, то створена об'єднана база може розглядатися як сховище інформації, якщо в ній розміщено дані, накопичені за доволі тривалий період часу, які можуть слугувати за підґрунтя для прийняття певних управлінських ухвал, а інформаційне середовище дозволяє узагальнювати дані й забезпечує захист первинних даних від спотворювання або модифікування. Таке сховище інформації слугує немовби за резервуар, до якого зливаються потоки від безлічі баз даних, які живлять і наповнюють його.

Можливий є й інший варіант: база даних просто збільшується за обсягом, оскільки в ній накопичуються дані за доволі тривалий час, і внаслідок цього вона стає чимось на зразок історичного архіву. За приклад може слугувати накопичування в електронному вигляді результатів переписів. Причинами винятково кількісного зростання обсягу бази даних можуть слугувати також сам тип накопичуваної інформації (наприклад якщо база даних збирає зображення) або частість її надходження. (Наприклад накопичування даних супутникової телеметрії). За такими базами даних закріпилася назва *надпотужні бази* даних (*very large database – VLDB).*

Вочевидь, що з часом у міру підвищування щільності зберігання інформації, здешевлювання необхідних для цього носіїв, вдосконалювання методів та засобів розпаралелювання опрацьовування на невеликих машинах, розвинення технології RAID (*RAID*– абревіатура від *Redundant Array of Inexpensive Disks, (*надлишковий масив недорогих дисків) та програмних комплексів для роботи з великими масивами даних зазнавали змінювання й уявлення про межу, що вона відокремлює не потужні бази даних від інших. На сьогодні ця межа перевищує 100 Гбайт. База даних більшого обсягу вже може претендувати на те, аби вважатися за надпотужну, а лише кілька років тому здавалося, що уявлювальна межа такого нарощування – 10 Гбайт.

1.1.2 Поняття про систему керування базами даних

Система керування базою даних – це програмний комплекс, що забезпечує функціонування бази даних. Вона зберігає дані, відповідає за їхнє зберігання, безпеку, цілісність, взаємну відповідність і забезпечує доступ користувачів до даних. У віданні СКБД перебуває *словник даних (data dictionary),* який іноді називають *каталогом системи (system catalog).* У цьому словнику розміщено відомості про все, що зберігається в базі даних: найменування, структуру, розміщення й типи. Відомості такого характеру є метаданими (*metadata),* тобто дані *про дані. Весь* життєвий шлях певного фрагмента даних – від створення до знищення – відбувається у словнику даних так само, як і інформація про його логічну й фізичну структуру. Адміністратор бази даних має підтримувати тісний зв'язок зі словником даних, який слугуватиме йому продовж всього часу існування бази.

1.1.3 Безпека даних

Безпека – головна проблема при створюванні, тестуванні й супроводжуванні бази даних. Запитання про те, чи потрібні засоби забезпечення безпеки, саме собою не є коректне. Можна ставити лише запитання, наскільки їх має бути розвинено. Зазвичай, СКБД пропонують кілька специфічних механізмів підвищення рівня безпеки на додання до тих, які вже забезпечуються операційною системою й системою підтримки мережі. Найпоширенішими з них є обов'язкова реєстрація й подальше розпізнавання користувачів для надавання їм доступу до тих чи інших розділів даних.

Окрім цього, практично стандартного для СКБД прийому, існує й безліч інших і поділ користувачів на групи з різними привілеями доступу, підтримка профілів та ін. Всі ці засоби мають на меті одне – забезпечити послідовне втілювання в життя певної політики безпеки в перебігу функціонування бази даних. Наприклад, лише користувач, якого система розпізнала й віднесла до тих користувачів, що вони належать до групи авіаційних фахівців, може дістати доступ до даних, стосовних авіації. Або лише користувач, якого система

розпізнала як оператора, може дістати доступ до засобів створювання резервних копій даних.

1.1.4 Цілісність даних

Під терміном *цілісність (integrity)* розуміється взаємна узгодженість окремих фрагментів даних та їхня коректність. *Узгодженість (consistency)* означає, що всі порції даних в базі даних має бути одноманітно змодельовано й долучено до системи. Кваліфікувати дані як коректні можна лише тоді, коли вони є вірогідні, точні й вагомі.

Один із способів забезпечення цілісності, який взято повсюдно на озброєння в СКБД, – *блокування* стороннього доступу до даних в перебігу їхнього опрацьовування. Зазвичай, дані блокуються на рівні сторінки бази даних чи рядка. Механізм блокування є неодмінним компонентом складнішого механізму забезпечення численної кількості користувачів, про що йтиметься нижче.

Інший прийом, який також сприяє зберіганню цілісності даних, – тиражування (реплікація) змінювання фрагмента даних, який як наслідок зберігається в кількох місцях. І останнє, на що слід звернути увагу, – необхідність тримати під контролем дані, які вводяться до системи або редагуються, і стежити за їхньою відповідністю до вимог специфікацій (наприклад, щоби дані не виходили за межі заданого діапазону).

Якщо слідувати технології моделювання й організації даних, рекомендованій та випробуваній на практиці, СКБД автоматично надійно підтримуватиме цілісність даних впродовж всього життєвого циклу бази даних. Слід пам'ятати, що дані, не пов'язані поміж собою, є просто сміттям: лише поєднані та взаємозв'язані дані утворюють інформацію. Отже, цілісність є неодмінною умовою перетворювання розрізнених даних на дещо таке, що становитиме цінність для споживачів інформації.

Неодмінною вимогою для більшості СКБД є забезпечення розрахованого на численну кількість користувачів доступу до даних. Тут виникає проблема узгодженості операцій, виконуваних кількома користувачами, з базою даних. Проблема полягає в тім, що, якщо два чи більш користувачів звернуться до одного фрагмента даних водночас, СКБД повинна буде гарантувати зберігання цілісності й вірогідності даних. Методи, які при цьому використовуються, є доволі прості, але для їхньої програмної реалізації потрібні певні зусилля. Дійсно, надати можливість двом користувачам водночас переглянути одні й ті самі дані, не змінюючи їх, не є складно. Проте СКБД доведеться серйозно взятися до справи, якщо один користувач змінюватиме фрагмент даних в той самий час, коли інший їх переглядає чи також намагається змінити. В цьому разі СКБД має створити кілька копій фрагмента даних, погодити всі внесені до копій зміни і, врешті-решт занести до бази даних єдиний відкоригований варіант даних, який задовольнятиме всіх учасників операції.

Вище вже згадувалося про один з аспектів технології паралельного доступу – блокування (*locking).* Власне кажучи, чим менше є обсяг фрагмента

даних, який може бути адресовано механізмом блокування, тим оптимальніше забезпечується паралелізм за розрахованого на численну кількість користувачів доступу до бази даних, тобто тим більше користувачів можуть працювати з нею, не заважаючи один одному.

У переважній більшості баз даних щонайменшою порцією даних, яку можна залучити до процедури блокування, є рядок. Більші фрагменти – блок і сторінка бази даних. Блокування на рівні рядка оптимальніше забезпечує доступ з випадковим розподілом по всьому масиву даних, а блокування на рівні сторінки чи блока є більш продуктивне за вельми тривалих операцій з близькими за значенням даними.

Отже, забезпечення паралельного доступу та зберігання цілісності – дві надто тісно пов'язані поміж собою функції СКБД. У підґрунті механізму реалізації цих функцій в СКБД лежить поняття *транзакції (transaction).* Коли користувач намагається переглянути певні дані чи змінити їх, він стосовно до бази даних *виконує транзакцію.*

1.1.5 Поняття транзакції

Один з ключових компонентів СКБД – адміністратор транзакцій (transaction manager), єдиною функцією якого є забезпечення цілісності даних за паралельного доступу до бази даних. У нього доволі багато роботи, оскільки до бази даних водночас звертається велика кількість користувачів, причому вельми часто до одних і тих самих даних, а адміністратор транзакцій повинен повертати дані до бази, у такий спосіб, нібито вони виконуються у кожного моменту часу лише одним користувачем, забезпечуючи тим самим цілісність та коректність даних. Це є відповіддю на запитання, в який спосіб СКБД має забезпечувати взаємозв'язок великої кількості копій первинного фрагмента даних, які різні користувачі змінювали. При опрацюванні транзакцій, які виникли вподовж певного часу, послідовність даних можна гарантувати лише в разі суворого дотримання дисципліни редагування-зберігання-відновлення фрагмента даних, яка набула назви (serialization). Отже, СКБД має зорганізовувати виконання транзакцій, які адресовано одному фрагментові даних такий спосіб, ніби це є послідовні операції змінювання даних файла. Транзакція є елементарною неперервною операцією в механізмі паралельного доступу й у функціонуванні СКБД в цілому. Не може бути нічого дрібнішого в плані виконуваних операцій, а ніж транзакція. Інакше кажучи, жодного стороннього втручання до даних після запуску транзакції на виконання бути не може. Всі транзакції повинні бути *атомарними,* незалежно від того, успішно чи ні завершується та чи інша конкретна процедура. Транзакція, яка спричинилася до очікуваного результату, називається тією, що вона *відбулася, прийнятою (committed),* в іншому разі – *знехтуваною, анульованою (rolled back).*

СКБД організовує оновлювання даних в базі даних, використовуючи транзакції як *логічні одиниці роботи.* Нормальне завершення операції, наявне припинення користувачем розпочатої операції чи її раптове переривання

внаслідок апаратного збою чи з іншої причини, яка виходить за межі нормального функціонування системи – все це змушує СКБД повернутися до численної кількості копій фрагментів даних, які було створено заздалегідь, й або поновити їх, якщо транзакція відбулась, або відтворити дані й анулювати всі незавершені зміни. Для того щоби таке анулювання відбулося й для повторення операції змінювання даних, яку було припинено за екстраординарних обставин, СКБД веде *журнал транзакцій (transaction log).* Фактично операція анулювання *(rolling back)* – це відома процедура *відкочування (undo),* а відтворення процедури після переривання *(rolling forvard)* – операція повторювання *(redo).* До останньої операції слід звертатися тоді, коли останню транзакцію, після апаратного збою, не було завершено перезаписом відновлених даних на диск. За таких обставин СКБД повторює всю транзакцію. З цього випливає, що саме атомарний, неподільний характер транзакції є ключовим моментом її можливого анулювання або відтворення за необхідності.

1.1.6 Реляційна база даних

У реляційній базі даних підтримується єдина логічна структура, а саме відношення (relation), тобто двовимірна структура даних, яку за звичай називають таблицею (table). Атрибути (attributes) або стовпці (columns) містять інформацію про цю структуру. На рис. 1.1 подано таблицю EMPLOYEE, яка містить такі атрибути, як ім'я співробітника, зарплатня, ім'я менеджера тощо.

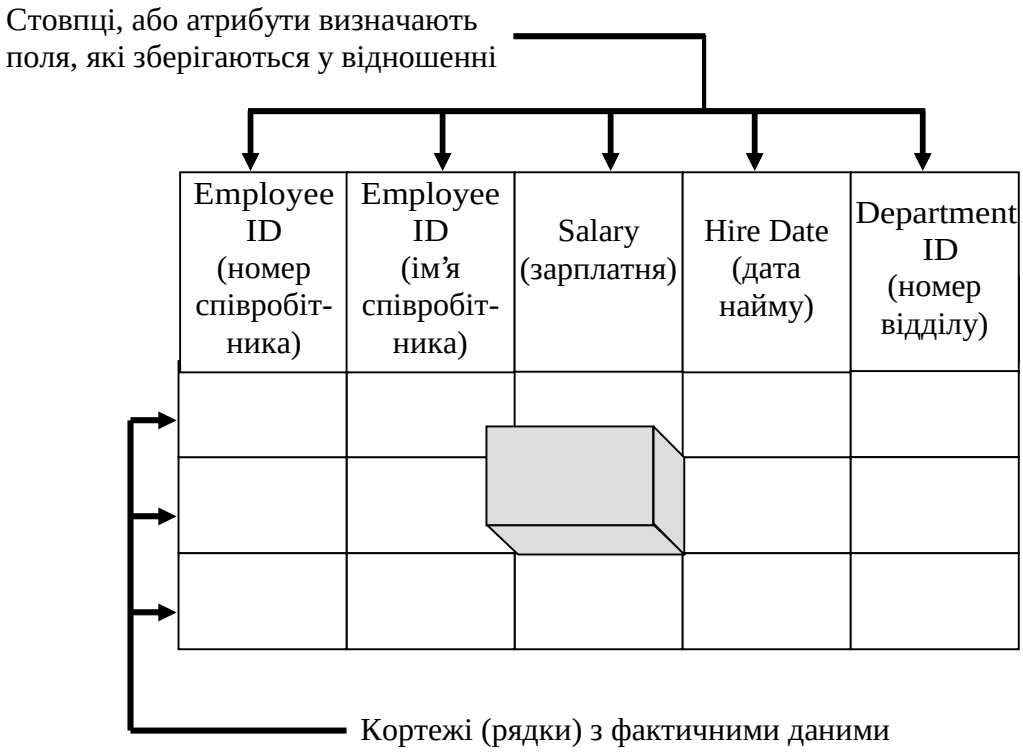

Рисунок 1.1 – Компоненти таблиці Employee.

Фактичні значення даних в таблиці називаються кортежами (tuples) або рядками (rows). Взаємозв'язок може бути задано поміж двома чи більшою кількістю таблиць, які можуть і не містити даних.

Атрибути можуть групуватися на підставі їхнього взаємозв'язку з утворенням складеного ключа (composite key) або навіть складеного первинного ключа. Первинним ключем (primary key) називається атрибут чи то група атрибутів (складений ключ), який у єдиний спосіб ідентифікує рядок в таблиці. У Oracle передбачено автоматичне створювання індексу для первинного ключа. Таблиця може мати лише один первинний ключ, але для забезпечення цілісності даних щодо посилань у кожної таблиці має бути власний первинний ключ. Оскільки первинні ключі використовуються як ідентифікатори, вони повинні містити певний елемент даних, тобто вони не можуть мати непевне значення (null).

Складений ключ складається з одного чи кількох стовпців, розташованих у порядку пріоритету на початку кожної таблиці, значення яких утворюють ключ.

До відношення можна включати додаткові атрибути з унікальними значеннями. На відміну від первинних, унікальні ключі (unique keys) можуть містити непевні значення (null). На практиці унікальні ключі використовуються для того, щоби запобігти дублюванню даних в таблиці, а не для ідентифікування рядків.

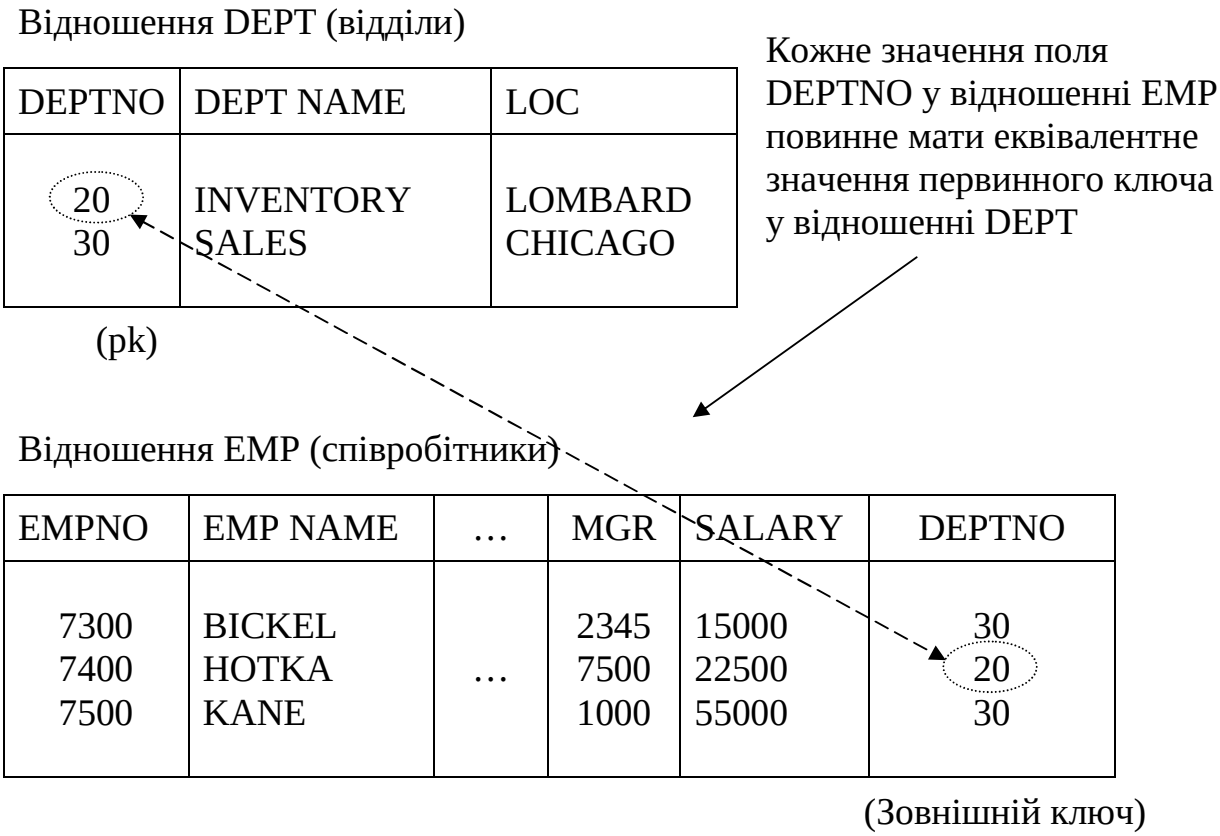

Рисунок 1.2 – Приклад зв'язку поміж первинним і зовнішнім ключами

Цілісність даних на рівні посилань реалізовується за допомогою коду мовою SQL, яка підтримує зв'язок поміж двома чи більш таблицями на підставі значень первинних та зовнішніх ключів. Підтримка цілісності даних спрощує роботу програміста з логічно позв'язаними даними, наприклад з даними про співробітників та відділи. Річ у тому, що при додаванні інформації про нових співробітників програмістові не треба піклуватися про існування відомостей про їхній відділ в таблиці з даними про відділи. У СКРБД Oracle передбачено підтримку обмежень цілісності.

Перевагами використання обмежень цілісності і зв'язку між первинними і зовнішніми ключами можна скористатися за допомогою багатьох інструментів.

У реляційній системі відомості зберігаються в окремих групах даних, які називаються *таблицями (tables).* Серед переваг таблиць слід зазначити простоту створення, додавання, змінювання чи організації зв'язку з іншими таблицями для подання необхідних результатів. Кінцевому користувачеві слід лише розуміти значення даних, а не спосіб доступу до них (див. рис. 1.2). При цьому вибирання відомостей про всіх співробітників відділу 20 виконується так само просто, як і про всіх співробітників, які звітують перед певним менеджером. Користувач може одержувати будь-які відомості про відділи та співробітників і вилучати їх за допомогою запиту на підставі значень будь-яких стовпців, а не лише на підставі певних значень ключа. Вибирання даних з двох чи більш таблиць ґрунтується на типі стовпця (або атрибуту), який є спільним для обох (або всіх) таблиць. Було б помилково встановлювати зв'язок поміж стовпцями з іменами співробітників (EMP NAME) та номерами відділів (DEPTNO). Щоб встановити зв'язок між таблицями, треба пов'язати стовпці з номерами відділів (DEPTNO). На рис. 1.2 показано приклад зв'язку поміж первинним (головною таблицею DEPT) і зовнішнім (підпорядкованою таблицею EMP) ключами.

1.2 Архітектура екземпляра Oracle

1.2.1 Мережні можливості Oracle

Робота бази даних і доступ до неї з боку користувачів перебувають під спостереженням фонових процесів Oracle.

Підґрунтя сучасного обчислювального середовища утворюють об'єднані в мережу комп'ютери. *Мережею (network)* називається система з'єднань комп'ютерів, яка дозволяє одному комп'ютеру взаємодіяти з іншим і спільно використовувати ресурси. Окрім проводів та інших фізичних компонентів, для роботи в мережі треба використовувати спеціальний набір правил передавання даних, або *протокол (protocol).*

СКРБД Oracle підтримує численну кількість типів мереж та протоколів. Щоб комп'ютери з програмним забезпеченням Oracle могли взаємодіяти один з одним, треба встановити на кожному з них компонент Net*8. Цей компонент дозволяє приховувати від кінцевого користувача складність внутрішнього

устрою і навіть тип використовуваної мережі. Оскільки Net*8 самостійно опікується всіма цими складними питаннями, кінцевому користувачеві не треба нічого робити для організації доступу до Oracle з мережі одного типу до іншої. Саме така мережна незалежність дозволяє легко переносити додатки Oracle на фізично різні комп'ютери.

Описана архітектура Oracle є однакова для всіх платформ, на яких використовується Oracle. Не зважаючи на певні незначні відмінності, головна частина архітектури Oracle є однакова для всіх платформ.

1.2.2 Фізична та логічна база даних

Базою даних називається набір пов'язаних даних, які використовуються чи вилучаються для одного чи кількох додатків. Фізичне розташування й реалізовування бази даних не залежать від прикладних програм, а сама фізична база даних може переміщуватися без жодного впливу на програми. У цьому полягає перевага компонента Net*8. Користувачеві слід всього лише ввести рядок підключення (ідентифікатор бази даних Oracle), а Net*8 здатен самостійно віднайти його і підключити до неї користувача.

Фізично база даних Oracle є набором файлів на твердому дискові. Логічно база даних Oracle поділяється на декілька схем. Кожна *схема* пов'язана з обліковим записом, який використовується для створювання об'єктів цієї схеми. Зазвичай, рекомендується створювати обліковий запис з особливими правами (до яких належать привілеї адміністратора бази даних) володіння всіма об'єктами додатка й керування привілеями цього об'єкта. Неможливо дістати доступ до інформації в базі даних без правильного облікового запису, пароля та привілеїв доступу. При цьому обліковий запис і пароль Oracle відрізняються від облікового запису й пароля операційної системи.

Облікові записи Oracle є методом роздільного використовування додатків. Зазвичай один додаток відповідає одній базі даних Oracle. Одне і те ж ім'я таблиці може існувати в двох різних схемах Oracle або різних облікових записах. Таблиці можуть мати однакове ім'я та все ж бути при цьому різними і зберігати різні відомості. Іноді одна й та сама база даних використовується для зберігання різних версій таблиць для розробників, тестерів або користувачів. На рис. 1.3 схематично показана архітектура Oracle.

1.2.3 Фонові процеси в Oracle

Для зберігання даних і керування функціями бази даних Oracle використовуються три групи файлів: файли бази даних, керувальні файли і журнали відкочування. Перед доступом до даних ці файли повинні бути створені, відкриті й доступні Oracle. Найважливішими серед них є файли бази даних, в яких, власне, і зберігається інформація додатка. А керувальні файли та журнали відкочування підтримують функціонування архітектури в цілому.

Для доступу до бази даних Oracle8i на комп'ютері слід запустити системні процеси та один чи кілька призначених для користувача процесів. Системні процеси Oracle також називаються *фоновими (background]*. Вони надають необхідні функції для доступу до бази даних з боку призначених для користувача процесів. Ці процеси запускаються за допомогою функції запуску. Також слід окремо відзначити монітор процесів (Process Monitor– PMON), системний монітор (System Monitor – SMON), процес запису до бази даних (Database Writer – DBWR) і процес запису до журналу (Log Writer– LGWR), які для функціонування бази даних слід неодмінно запустити і зробити активними

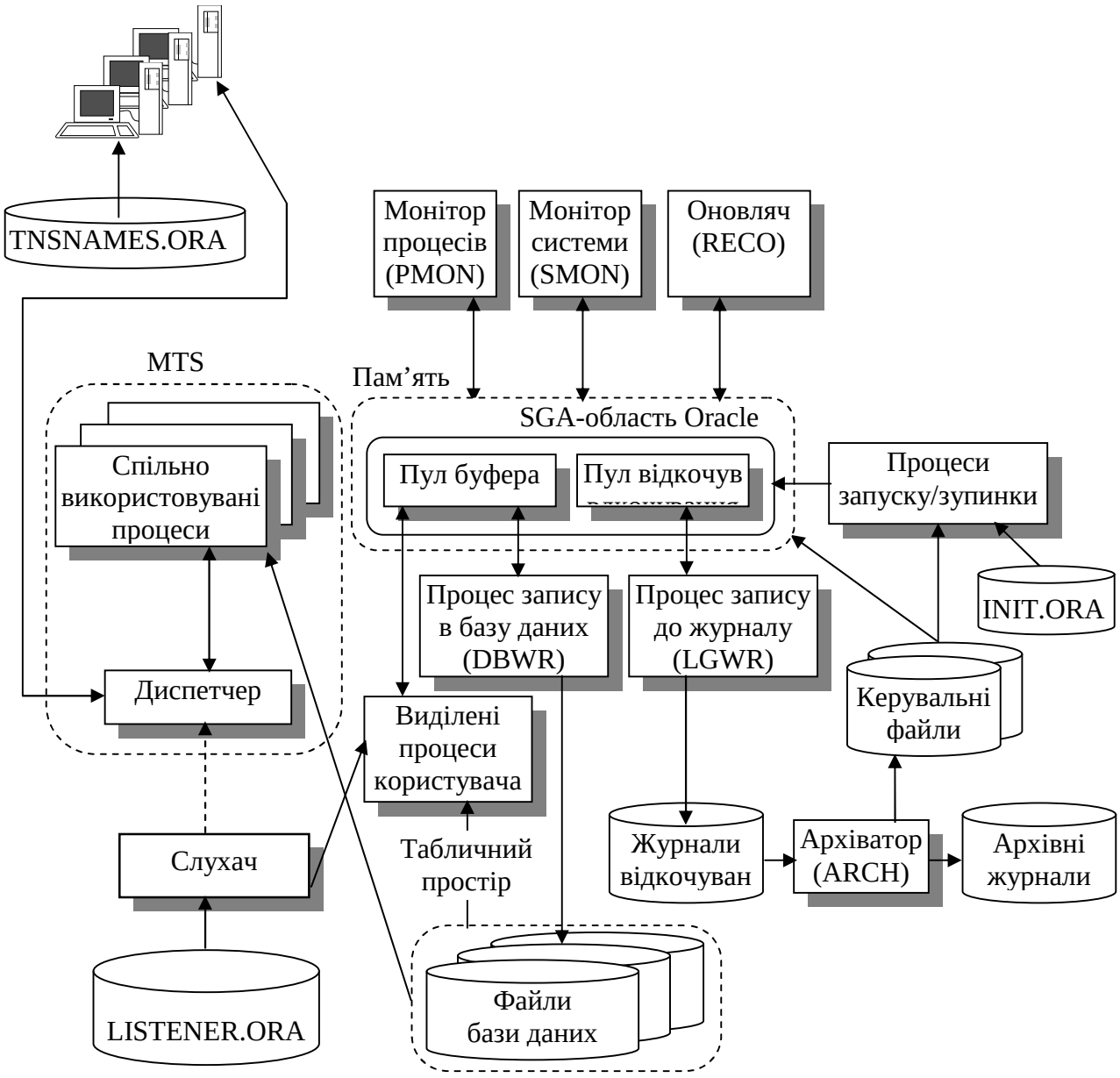

Рисунок 1.3 – Архітектура Oracle8i

Наприклад, для кожного зареєстрованого користувача створюється додатковий фоновий призначений для користувача процес (рис. 1.4).

Для спільного використовування фонових призначених для користувача процесів застосовується багатопотоковий сервер (рис. 1.5). У обох випадках процес-слухач спрямовує мережний трафік користувача до відповідної бази даних Oracle. Після створення фонових процесів процес-слухач (див. рис. 1.4) вже не втручається в роботу окремого користувача. Проте в разі застосування багатопотокового сервера для спільного використання пула виокремлених фонових процесів користувачі взаємодіють з диспетчером, а диспетчер – з програмами, які призначено для користувача.

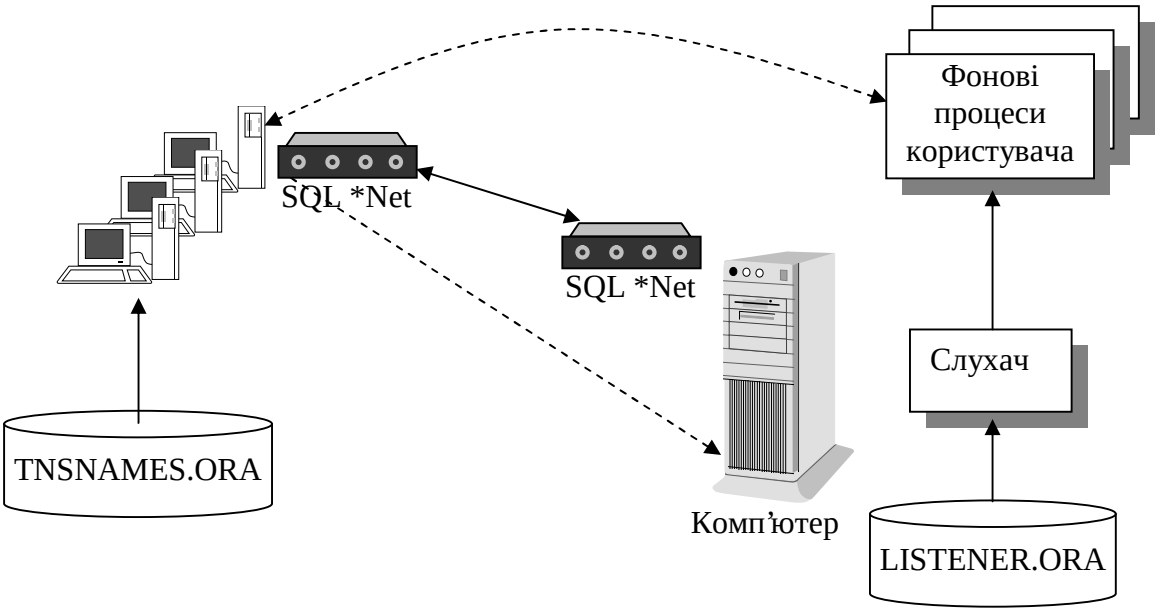

Рисунок 1.4 – Фонові користувацькі процеси

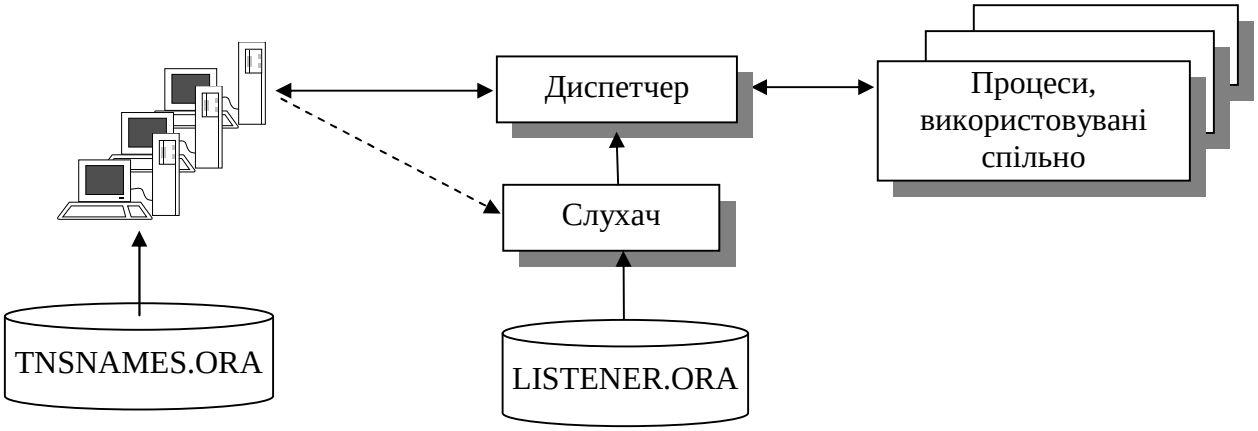

Рисунок 1.5 – Схема взаємодії багатопотокового сервера з фоновими процесами, використовуваними спільно

Фонові користувацькі процеси (user background processes) запускаються після реєстрування користувача в Oracle. Ці процеси використовуються для обміну даними з призначеним для користувача додатком (тобто програмою SQL*Plus, формами або програмою мовою С). Багатопотоковий сервер (multithreaded server – MTS) застосовується для спільного використання набору цих фонових процесів, оскільки для кожного з них треба надати оперативну пам'ять і витратити певний час на опрацьовування. За великої кількості користувачів на певному комп'ютері (наприклад 500 чи більше користувачів, які намагаються здійснити доступ до однієї бази даних Oracle) оперативна пам'ять витрачатиметься ефективніше при використанні MTS. Для цього MTS застосовує механізм організації черг, припускаючи, що не всі користувачі водночас зчитують і записують дані.

1.2.4 Системна глобальна область

Для зберігання інформації в блоках даних і керування ними в Oracle8i використовуються спеціальні структури пам'яті під загальною назвою *системна глобальна область (System Global Area*– SGA). Область SGA створюється відразу після запуску бази даних Oracle8i. Файл INIT.ORA задає розміри цих структур пам'яті і багато інших параметрів бази даних Oracle. Процес запуску бази даних зчитує цей файл і створює необхідні структури пам'яті в області SGA. На рис. 1.6 показана схему структур пам'яті, які утворюють область SGA.

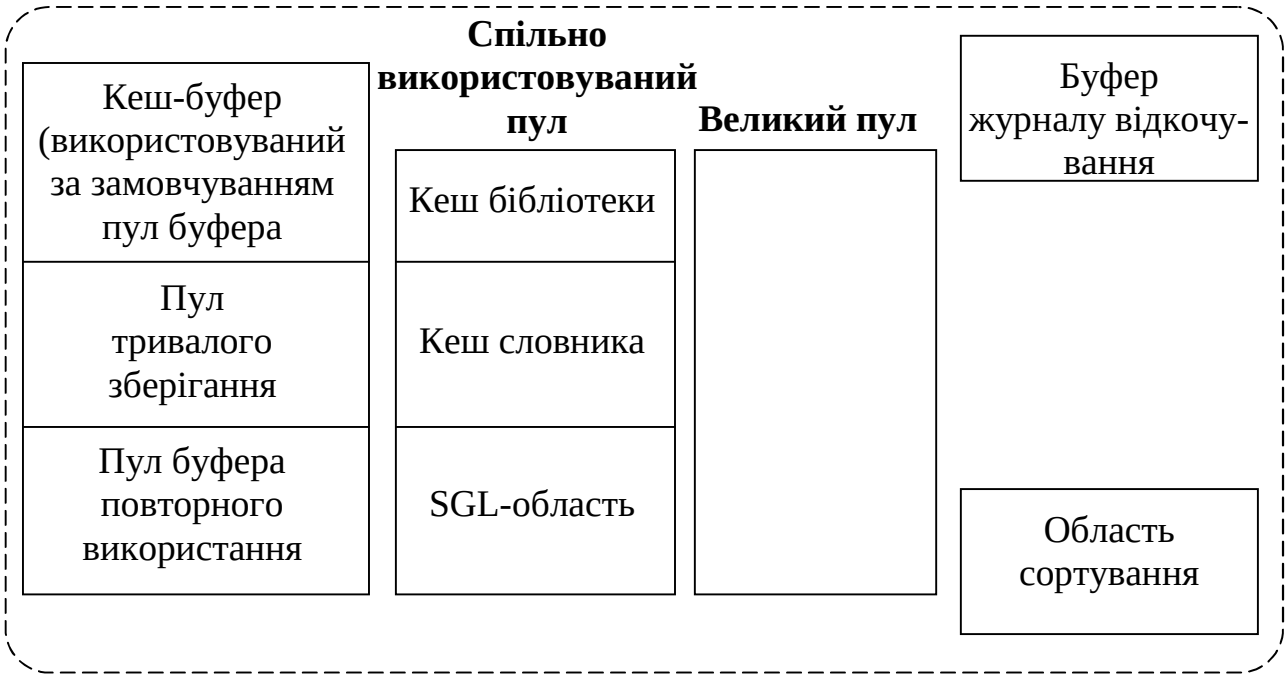

Рисунок 1.6 – Структура системної глобальної області Oracle8i

Область SGA й фонові процеси Oracle утворюють *екземпляр бази* даних Oracle8i. Залежно від використовуваних функцій і заданих параметрів, один і той же набір файлів бази даних може використовуватися різною кількістю екземплярів бази даних. Екземпляр бази даних Oracle містить унікальний набір фонових процесів, пам'ять, яка використовується спільно усіма процесами, і файли бази даних. На одному комп'ютері може бути запущено відразу кілька екземплярів бази даних Oracle.

Під *екземпляром Oracle (Oracle instance)* розуміють спосіб надсилання на унікальне середовище бази даних Oracle незалежно від її фізичного розташування на локальному чи віддаленому комп'ютері.

1.2.5 Табличні простори, керувальний файл, журнали відкочування

Під час інсталяції Oracle8i створює декілька використовуваних за умовчуванням файлів бази даних, які відповідають певним табличним просторам. Таблиці й інші об'єкти можуть зберігатися разом в одному табличному просторі. Табличний *простір (tablespace) є* свого роду спосіб організації даних, що зберігаються, в СКРБД Oracle, який можна порівняти зі звичними каталогами або папками комп'ютера. Для кожного табличного простору створюються файли бази даних, причому і має мати як мінімум один такий файл. Середовище Oracle має мати щонайменш два спеціальні табличні простори: SYSTEM і TEMP. Табличний простір SYSTEM містить всі об'єкти (таблиці, індекси, кластери тощо), необхідні для керування роботою самої бази даних Oracle8i. Ці об'єкти утворюють словник *даних (data dictionary). Табличний* простір TEMP є робочою областю, в якій виконуються різноманітні команди SQL, наприклад з'єднання даних з кількох таблиць, зазначених у реченні FROM, або сортування даних.

У файлі, якій здійснює керування, записано ім'я бази даних, дату й час її створення, розташування журналів бази даних та журналів відкочування, а також необхідну інформацію для запуску саме тих файлів (з тією самою позначкою часу), які використовувалися перед попереднім закриттям даного екземпляра Oracle. По суті, ця інформація використовується для відновлення резервної копії Oracle. Керувальний файл є важливою складовою частиною процесу відновлення, тому що саме в ньому можна віднайти майже всю необхідну інформацію про відновлення даних. За втрати чи ушкодження курувального файла екземпляр Oracle не можна буде запустити. У файлі ініціалізації INIT.ORA можна зазначити відразу кілька копій керувального файла, які автоматично використовуватимуться за втрати чи ушкодження початкового керувального файла.

Журнали відкочування (redo logs або *online redo logs)* містять інформацію про всі дії, виконувані в базі даних, тобто всі зміни об'єктів, призначених для користувача та системних об'єктів. За виникнення будь-якого збою, наприклад за втрати одного чи кількох файлів бази даних, записані в журналах

відкочування дані використовуються для відновлення бази даних Oracle без втрати всіх зафіксованих транзакцій. За виникнення будь-яких проблем (приміром, за раптового вимикання електроживлення) після перезавантаження комп'ютера Oracle автоматично відновить дані на підставі записів в журналах відкочування. Фоновий процес SMON автоматично врахує всі зафіксовані зміни з журналів відкочування для відновлення стану файлів бази даних.

База даних Oracle8i повинна мати щонайменше два журнали відкочування. Вони мають фіксовану довжину і ніколи не змінюють заданого розміру.

У Oracle журнали відкочування можуть зберігатися в архівних журналах у такий спосіб, щоби згодом ними можна було скористатися для відновлювання даних. Імена архівних журналів зберігаються в керувальних файлах, а сам процес називається *режимом архівування (archive log mode).* Параметрами режиму архівування може керувати оператор за допомогою кількох спеціальних команд або автоматично, задаючи необхідні параметри у файлі ініціалізації INIT.ORA. Архівні журнали є точними копіями вже заповнених журналів відкочування, які створюються відразу після перемикання процесу запису змін до іншого журналу відкочування Oracle.

1.2.6 Фонові процеси DBWR, LGWR, SMON, PMON. Кеш-буфер

Для організації роботи з користувачами в Oracle завжди повинно бути запущено чотири фонові процеси: DBWR, LGWR, SMON та PMON.

Процес DBWR переписує зміни блоків бази даних в області SGA (при вставлянні, оновлянні чи знищуванні рядків) до відповідних файлів бази даних. Блок даних в області SGA зі зміною даних, яку ще не записано на твердий диск, називається *брудним (dirty)* блоком. Процес DBWR записує дані лише з брудних блоків, починаючи з найстарішого (або щонайдовшого) блока в області SGA.

Робота з давно створеним *(least-recently-used – LRU)* блоком означає в Oracle процес роботи насамперед із найстарішим блоком.

Процес LGWR записує інформацію про відкочування транзакцій з буферів відкочування області SGA до журналів відкочування. Якщо користувач або транзакція фіксують зміни бази даних Oracle, то саме процес LGWR записує вміст буферів відкочування до журналів відкочування перед потвердженням успішно виконаної фіксації. У разі виникнення якихось збоїв комп'ютера це гарантує відновлення всіх зафіксованих транзакцій.

Процес SMON виконує безліч різних завдань, включаючи моніторинг спроб змінювання одного об'єкта з боку кількох користувачів, що може призвести до виникнення взаємного блокування (*deadlock). Процес* SMON стежить за кожною транзакцією й виконує відкочування з видачею повідомлення про помилку для транзакції з гіршим часом виконання. Він також керує розподілом простору в табличних просторах і використовується для роботи з транзакціями в журналах відкочування, якщо роботу з базою даних Oracle не було завершено у нормальний спосіб (включаючи припинення роботи

з примусовим перериванням транзакцій або за раптового збою апаратного забезпечення, який призвів до зупинки процесів Oracle).

Взаємне блокування (deadlock) виникає в тому разі, коли транзакції намагаються виконати численні оновлення даних в одному й тому самому об'єкті. Наприклад, одна транзакція намагається заблокувати об'єкт, який вже заблоковано іншою транзакцією, причому друга транзакція, у своєю чергою, намагається накласти блокування на інший об'єкт, який вже заблоковано першою транзакцією. У такій ситуації транзакції перебуватимуть в стані очікування непевний час, тому що ці блокування не може бути знято.

Процес PMON стежить за призначеними для користувача процесами, диспетчером і спільно використовуваними фоновими процесами багатопотокового сервера. При збої фонових процесів (спільно використовуваних або виділених) призначених для користувача, PMON автоматично виконає відкочування всієї виконаної роботи з момента початку останньої транзакції (тобто з момента останньої команди фіксації COMMIT або відкочування ROLLBACK транзакції) і звільнить всі блокування, накладені цією транзакцією.

У деяких режимах роботи бази даних Oracle можуть використовуватися додаткові фонові процеси. Наприклад, процес ARCH використовується в режимі архівування для створення архівних журналів на підставі журналів відкочування. А процес RECO використовується при роботі з реплікатором Oracle Replication для транзакцій, які охоплюють два чи більш екземплярів Oracle. При збої транзакції процес RECO зафіксує (або виконає відкочування) тих частин транзакції, які спричинили цей збій. Врешті, процес LCK використовується сервером паралельних обчислень Oracle Parallel Server.

Процес-слухач, по суті, нагадує регулювальника обміну даними поміж користувачами екземпляра Oracle й самим екземпляром Oracle. Процес-слухач не пов'язаний ані з яким окремим екземпляром Oracle і для вибору потрібного екземпляра використовує параметри, зазначені у файлі LISTENER.ORA. Процес-слухач існує для кожного типу мережного підмикання, використовуваного даним комп'ютером. У середовищі без сервера MTS процес-слухач запускає фоновий процес на тому ж самому комп'ютері, де запущено екземпляр Oracle, реєструє користувача і продовжує стежити за спробами підмикання до Oracle з боку інших користувачів. Далі користувачі можуть безпосередньо взаємодіяти з цим фоновим процесом впродовж всього сеансу свого підмикання до екземпляра Oracle. У середовищі з MTS процес-слухач передає запити користувача доступному диспетчерові. Цей диспетчер розподіляє запити користувача поміж існуючими фоновими процесами, дозволяючи багатьом користувачам спільно використовувати кілька фонових процесів. Користувачі взаємодіють з диспетчером, а диспетчер потім надсилає їхні запити вільному фоновому процесові (або запускає додаткові фонові процеси, якщо вимоги користувача є надто високі). MTS є вельми ефективний при роботі з сотнями або навіть тисячами користувачів, яких підімкнено до одного й того ж екземпляра Oracle. Кожен користувач має доступ до файлу TNSNAMES. ORA, що він містить інформацію про тип мережі, та імені екземпляра Oracle, до якого треба підімкнутися.

Кожен екземпляр Oracle має власну область SGA зі структурою пам'яті, яка створюється при запуску, а звільнюється за припинення роботи даного екземпляра. Область SGA складається з кількох різних частин, які подано на рис. 1.6.

Кеш-буфер (buffer cache) є робочим місцем для розміщення блоків даних у відповідь на запити з боку призначених для користувача процесів, що стосуються вилучення даних з табличних просторів. У Oracle перед виконанням операції читання даних, що вилучаються, завжди відбувається перевірка існування запитаного блока даних. Співвідношенням *влучень і промахів (hit ratio) кеш-буфера* називається відношення кількості читань блоків даних з кеш-буфера (влучень) до кількості всіх операцій читання блоків даних з табличного простору (сума промахів та влучень). Взагалі, правильне окреслення співвідношення влучень та промахів може значною мірою вплинути на продуктивність екземпляра Oracle. Для різних додатків його оптимальне значення може виявитися різним, але для більшості додатків воно не повинне бути менше за 90%.

Величину співвідношень влучень та промахів можна контролювати за допомогою параметра DB_BLOCK_BUFFERS у файлі INIT.ORA.

Щоби оптимізувати керування величиною співвідношення влучень та промахів, пул буфера в Oracle можна розбити на три додаткові області. *Пул тривалого зберігання (keep pool)* призначено для тих об'єктів, які можуть багаторазово прочитуватися, але в них рідко вставляються нові дані. Пул повторного виконання (recycle pool) використовується для тих об'єктів, в яких часто виконуються операції вставляння й оновлення даних, але які не надто часто прочитуються. Врешті, використовуваний замовчуванням пул *(default* pool*)* або просто пул *буфера (buffer pool)* використовується для всіх інших об'єктів. Відстежуючи роботу об'єктів і розміщуючи їх в різних пулах, можна відділити об'єкти, які часто й рідко зчитуються, що дозволяє розміщувати в області SGA оперативної пам'яті лише часто використовувані об'єкти й не застосовувати для них LRU-опрацювання.

Спільно використовуваний пул містить кеш бібліотеки, кеш словника й область SQL. *Область SQL* містить текст команд SQL. Кеш *бібліотеки (library cache)* призначено для команд SQL, які використовуються для роботи з додатком бази даних, а кеш *словника даних (dictionary cache)* – для команд SQL, які використовуються для роботи із словником даних.

Розмір спільно використовуваного пула визначається параметром SHARED_POOL_SIZE в файлі INIT.ORA.

У Oracle8i вводиться поняття великого пула, призначеного для поліпшеного опрацьовування мережного трафіка і конфліктів у спільно використовуваному пулі. Багатопотоковий сервер і процеси резервного копіювання Oracle використовують саме цю область пам'яті, а не спільно використовуваний пул. Характеристики великого пула визначаються за допомогою параметрів LARGE_POOL_SIZE та LARGE_POOL_MIN_ALLOC файлу INIT.ORA.

2 ПРОГРАМУВАННЯ НА МОВІ SQL В СУРБД ORACLE

2.1 Керування програмою SQL*Plus, та робота з нею

*2.1.1 Призначення SQL*Plus*

Утиліта SQL*Plus дозволяє виконувати команди SQL та блоки PL/SQL, а також розв'язувати низку інших завдань. За допомогою SQL*Plus можна:

- уводити, редагувати, запам'ятовувати, завантажувати і виконувати команди SQL та блоки PL/SQL;
- форматувати, створювати, зберігати, друкувати й публікувати у Web результати виконання запитів (звіти);
- одержувати опис (імена й типи стовпців) будь-якої таблиці й подання;
- звертатися до віддалених баз даних і копіювати з них дані;
- надсилати та приймати повідомлення від кінцевих користувачів;
- адмініструвати базу даних.

Базові поняття

При роботі з SQL*Plus використовуються базові поняття, наведені у табл. 2.1.

Таблиця 2.1 – Базові поняття SQL*Plus

Команда	Команда SQL*Plus або оператор SQL Oracle
Блок PL/SQL	Група взаємозалежних операторів PL/SQL, оформлена у вигляді анонімного блока
Таблиця	Базова одиниця зберігання даних у Oracle.
Запит	SQL-оператор SELECT, що він обирає інформацію з однієї чи кількох таблиць
Результати запиту	Дані, повернуті запитом
Звіт	Результати запиту, сформатовані за допомогою команд SQL*Plus

*2.1.2 Запуск SQL*Plus і вихід з нього*

Для запуску утиліти SQL*Plus використовується команда операційної системи, зазвичай – **sqlplus**. Запуск виконується з командного інтерпретатора або вікна командного рядка.

У версіях для операційних систем Microsoft використовувалися інші імена команди для виклику SQL*Plus, наприклад **sqlplus30**. Сьогодні (розпочинаючи з версії 8.1.5) у цих операційних системах наявні дві версії цієї

утиліти: **sqlplus**, що вона працює у вікні командного рядка, й віконна версія, **sqlplusw**.

Опції командного рядка

Як випливає із синтаксису, при виклику SQL*Plus можна зазначити низку опцій. Вони коротко описані в табл. 2.2.

Таблиця 2.2 – Опції командного рядка утиліти SQL*Plus

Опція	Призначення
HELP	Видає інформацію про використання й синтаксис виклику команди **sqlplus** і завершує роботу
VERSION	Видає інформацію про версії утиліти SQL*Plus і завершує роботу
MARKUP	Дозволяє генерувати повну HTML-сторінку за результатами запиту або сценарію
RESTRICT	Дозволяє вимкнути низку команд, взаємодіючих з операційною системою. Якщо зазначено рівень обмеження 3, сценарій **login.sql** у поточному каталозі не читається. Стандартний сценарій запуску **glogin.sql** читається, але обмежені команди в ньому не працюють
SILENT	Придушує видачу всіх інформаційних повідомлень і запрошень SQL*Plus, включаючи запрошення командного рядка, коди команд і початкову інформацію, за звичай видавану при входженні до SQL*Plus. Використовується за виклику SQL*Plus з іншої програми, коли його бажано приховати, і разом з опцією **MARKUP** при створюванні Web-звітів

Реєстраційна інформація

При запуску SQL*Plus треба ввести реєстраційну інформацію. Якщо цю інформацію не зазначено, але зазначено початковий сценарій, передбачається, що реєстраційну інформацію зазначено в першому рядку сценарію. Якщо реєстраційну інформацію не зазначено ані в командному рядку, ані в початковому сценарії, SQL*Plus запрошує її.

Якщо зазначено ім'я користувача й не зазначений пароль, SQL*Plus запрошує лише пароль. При вставлянні опції **SILENT** запрошення для введення імені користувача й пароля не видаються, введене ім'я користувача відбивається, а пароль, як зазвичай-ні.

Якщо замість імені користувача й пароля зазначена косу (/), передбачається стандартна реєстрація від імені користувача операційної системи (аутентифікація операційною системою). При цьому не можна зазначати рядок зв'язку. Сервер реєструє користувача **OPS$*name***, де ***name*** – реєстраційне ім'я поточного користувача операційної системи. Префікс "**OPS$**" можна замінити на будь-який інший за допомогою відповідного параметра ініціалізування сервера (**OS_AUTHENT_PREFIX**).

Настроювання програми SQL*Plus

Для адміністрування SQL*Plus призначено два файли – **glogin.sql** та **login.sql**.

Файл **glogin.sql** визначає глобальні параметри настроювання, а файл **login.sql** – параметри настроювання для окремих користувачів. Ці файли включають команди SQL*Plus та вирази SQL, які використовуються кожного разу, коли користувач Oracle викликає SQL*Plus. Спочатку зчитується та виконується файл **glogin.sql**, а потім – файл **login.sql** відповідного користувача. Файл glogin.sql розміщується в каталозі **$ORACLE_HOME/sqlplus/admin**.

Вихід з SQL*Plus

Для виходу з SQL*Plus необхідно ввести команду EXIT. Буде видано інформацію про версії сервера, до якого було підімкнено утиліту, та про версії SQL*Plus. Потім робота програми завершується й або закривається вікно, або ви одержуєте запрошення командного рядка операційної системи.

За замовчуванням при виконанні команди **EXIT** SQL*Plus фіксує поточну транзакцію і завершує роботу з кодом повертання 0 (**SUCCESS**).

2.1.3 Підмикання до стандартної бази даних

Для доступу до даних необхідно спочатку підімкнутися до бази даних. При запуску SQL*Plus зазвичай відбувається підмикання до *стандартної бази даних* із зазначеним ім'ям користувача й паролем. Після запуску й успішної реєстрації можна в будь-якого моменту підімкнутися до схеми іншого користувача або до іншої бази даних. Для цього використовується команда **CONNECT**.

Стандартна база даних конфігурується на рівні операційної системи шляхом встановлення змінного середовища або редагування файла конфігурації Oracle.

Використання команди CONNECT

Команда **CONNECT** має такий синтаксис:

<команда CONNECT> ::=

CONN[**ECT**] [<користувач і база даних>] [**AS** <роль користувача>]

Інформація про користувача й базу даних, до яких треба підімкнутися, вводиться в тому самому форматі, що й при запуску (див. синтаксис виклику утиліти SQL*Plus вище). При підмиканні, як і при запуску, можна зазначати роль користувача, від імені якого воно виконується.

Якщо пароль користувача застарів, при підмиканні доведеться ввести нового пароля. Якщо ж обліковий запис заблоковано, то видається

повідомлення про це й підмикання не відбувається, доки адміністратор не розблокує обліковий запис.

У разі якщо підмикання з будь-якої причини не вдалося, ви залишаєтеся відімкненими від бази даних, про що буде повідомлено. Можна також явно відімкнуся від схеми поточного користувача за допомогою команди **DISCONNECT**.

Бази даних на інших комп'ютерах які просто відрізнюються від стандартної називаються віддаленими. До віддаленої бази даних можна підімкнутися, якщо для неї сконфігуровано мережні служби Oracle Net і драйвери мережних служб віддаленої бази даних і локального клієнта SQL*Plus сумісні.

CONNECT scott/tiger@training

2.1.4 Головні команди

Після запуску утиліта SQL*Plus видає запрошення командного рядка (зазвичай це рядок "**SQL>** "), після чого можна вводити команди. Команди складаються з одного чи кількох слів, розділених довільною кількістю прогалин чи символів табуляції. Регістр символів у командах зазвичай не має значення. При натисканні клавіші Enter команда (при виконанні певних умов по її завершенню) виконується. Після виконання команди знову видається запрошення.

Певні команди, наприклад, блоки PL/SQL, можуть посідати кілька рядків. Тоді SQL*Plus видає додаткові запрошення з номером рядка, наприклад, "**2>**".

У командному рядку SQL*Plus можна вводити команди трьох видів:

- SQL-оператори для роботи з інформацією в базі даних;
- блоки PL/SQL для роботи з інформацією в базі даних;
- команди SQL*Plus для форматування результатів запиту, установлення опцій, редагування команд тощо.

Від виду команди залежить спосіб продовження її на наступному рядку, завершення й виконання.

Виконання команд

Буфер SQL

Останню введену команду (SQL-оператор або блок PL/SQL) SQL*Plus зберігає в області, називаємій **буфер SQL**. Оператор або блок залишається в цьому буфері, доки не буде введено нового. До буфера SQL не потребляє крапка з комою або скісна риса (/), що вона завершує оператор або блок, відповідно. Вміст буфера SQL можна редагувати й виконувати повторно.

Для повторного виконання вмісту буфера SQL використовується команда **RUN** або скісна риса (/). Команда **RUN** видає вміст буфера до стандартного вихідного потоку, а потім виконує. Скісна риса спричинює просте виконання команди з буфера.

Виконання операторів SQL

Оператор SQL можна завершити трьома способами:

- крапкою з комою (;);
- рядком, що він містить лише скісну рису (/);
- порожнім рядком.

Крапка з комою наприкінці рядка означає виконання оператора. Утиліта SQL*Plus виконує оператор і запам'ятовує його в буфері SQL. Цей символ є ознакою завершення команди, і після нього не можна вводити коментар.

Скісна риса в якості першого і єдиного символу чергового рядка означає, що оператор треба виконати. Утиліта SQL*Plus виконує оператор і запам'ятовує його в буфері SQL.

Порожній рядок в операторі SQL або сценарії SQL*Plus зазвичай означає, що введення команди завершено, але виконувати її покищо не треба (це поводження можна змінити за допомогою команди SQL*Plus **SET SQLBLANKLINES**). Команда міститься в буфер SQL і залишається там, доки не буде введено іншу команду.

Виконання блоків PL/SQL

Для введення блоків PL/SQL слід працювати в режимі PL/SQL. Утиліта SQL*Plus переходить до цього режиму, якщо:

- у командному рядку введено ключове слово **DECLARE** або **BEGIN**;
- введено одну з команд SQL, що вона створює зберігану програмну, одиницю, наприклад, **CREATE PROCEDURE**.

Утиліта SQL*Plus опрацьовує блоки PL/SQL так само, як і оператори SQL, за винятком крапки з комою і порожнього рядка. У режимі PL/SQL їхнє введення не спричинює виконання або завершення команди. Щоби завершити й виконати блок PL/SQL, треба ввести рядок з єдиним символом – скісною рисою (/). Щоби завершити введення блока і просто помістити його до буфера SQL, не виконуючи, уведіть рядок з єдиним символом крапка (**.**).

Виконання команд SQL*Plus

Команди SQL*Plus виконуються відразу й до буфера SQL не потрапляють. Завершувати команди SQL*Plus крапкою з комою можна, але не обов'язково. Більшість команд SQL*Plus можна скорочувати до однієї або кількох перших літер, достатніх для однозначного визначення команди.

Якщо треба ввести довгу команду SQL*Plus, її можна перенести на наступний рядок. Для цього наприкінці рядка слід ввести дефіс (-) і лише потім натиснути клавішу Enter. Буде видано запрошення, і введення команди можна буде продовжити.

У такий самий спосіб інтерпретується дефіс і в операторах SQL – якщо це останній не пробільний символ у рядку, він не потрапляє до буфера і просто вважається за ознаку продовження команди на наступному рядку.

Виконання команд базової операційної системи

З командного рядка SQL*Plus можна виконувати будь-які команди базової операційної системи. Для цього використовується команда HOST, за якою йде командний рядок для операційної системи.

Команду **HOST** можна скорочувати до перших двох літер. Якщо команду введено без параметрів, відкривається сеанс стандартного командного інтерпретатора операційної системи. У ньому можна виконувати будь-яку кількість команд. Для виходу з командного інтерпретатора UNIX досить виконати команду **EXIT** або ввести **Ctrl+D**.

Доступ до команди **HOST** можна заборонити, запускаючи утиліту SQL*Plus з будь-яким рівнем обмеження.

Команда DESCRIBE

Команда **DESCRIBE** дозволяє одержати опис таблиці або представлення, а також специфікацію типу, пакета, функції або процедури:

<команда DESCRIBE> ::=
DESC[**RIBE**] [[<схема>**.**]<об'єкт>[**@**<рядок зв'язку>]]

Опис таблиці, представлення чи синоніма містить таку інформацію:

- ім'я кожного стовпця;
- чи припускаються в стовпці значення **NULL**;
- типи даних стовпців;
- довжина або точність (та масштаб, для числових) стовпців.

Команда **DESCRIBE** дозволяє описувати об'єкти рекурсивно, причому глибина задається командою **SET DESCRIBE**. Можна також видавати номер рядка і зсовувати ім'я стовпця або атрибута, якщо об'єкт містить кілька об'єктних типів.

Довжину видаваного рядка опису можна контролювати за допомогою команди **SET LINESIZE**.

Опис функцій і процедур містить таку інформацію:

- тип програмної одиниці (процедура або функція);
- ім'я функції або процедури;
- тип значення, що повертається, для функцій;
- імена, типи, режим передавання і стандартні значення аргументів.

*2.1.5 Редагування команд SQL. Команди SQL*Plus для роботи з SQL буфером*

Команди SQL*Plus не потрапляють до буфера, тому редагувати їх можна лише в перебігу введення за допомогою клавіші **Backspace**.

Утиліта SQL*Plus підтримує кілька команд редагування тексту в буфері. Їх подано в табл. 2.3.

Таблиця 2.3 – Команди SQL*Plus для роботи з SQL-буфером

Команда	Призначення
A[**PPEND**] <текст>	Додає <текст> наприкінці рядка
C[**HANGE**] /<старий>[/<новий>[/]]	Замінює в рядку підрядок <старий> на рядок <новий>. Якщо <новий> не зазначено, просто вилучає підрядок <старий>
CL[**EAR**] **BUF**[**FER**]	Вилучає всі рядки з буфера
DEL [*]	Вилучає поточний рядок
DEL <n> [* | <m> | **LAST**]	Вилучає рядок з номером <n> або з рядка <n> до поточного, до рядка <m> або до останнього рядка, відповідно
DEL * [<n> | **LAST**]	Вилучає з поточного рядка по рядок <n> або до закінчення, відповідно
DEL LAST	Вилучає останній рядок
ED[**IT**] [<ім'я файла>[<розширення>]]	Викликає текстовий редактор операційної системи і завантажує до нього буфер або зазначений файл. Якщо редагується буфер, то в поточному каталозі створюється файл **afiedt.buf**, у якому й міститься результат редагування. Завантаження змін до буфера виконуються автоматично після виходу з редактора. Стандартний редактор в ОС UNIX задається змінною середовища **_EDITOR**. За замовчуванням в ОС UNIX використовується редактор **ed**, а у Windows – **Notepad**.
I[**NPUT**] [<текст>]	Вставляє рядок або <текст>, у тому числі з кількох рядків, до буфера після поточного рядка
LIST [<n> | * | **LAST**]	Видає, відповідно, усі рядки в буфері SQL, рядок із зазначеним номером, поточний або останній рядок
LIST <n> [* | <m> | **LAST**]	Видає рядок з номером <n>, усі рядки, розпочинаючи з рядка з номером <n>, до поточного, рядка з номером <m> або до закінчення буфера, відповідно.
LIST * [<n> | **LAST**]	Видає всі рядки з поточного рядка до рядка з номером <n> або до останнього рядка, відповідно.

2.1.6 Команди для роботи з файлами

Утиліта SQL*Plus дозволяє запам'ятовувати команди, оператори SQL і блоки PL/SQL у **командних файлах** (або **сценаріях**). Надалі ці командні файли

– текстові файли базової операційної системи – можна виконувати, завантажувати і редагувати.

Створення командних файлів

Створювати командні файли можна поза середовищем SQL*Plus за допомогою будь-якого текстового редактора або в середовищі SQL*Plus за допомогою поданих вище команд редагування або виклику зовнішнього редактора. Розглянемо засоби SQL*Plus – зберігання буфера SQL у файлі – команду **SAVE**, і виклик зовнішнього редактора для редагування буфера SQL– команду **EDIT**.

Команда SAVE

*Команда **SAVE*** дозволяє зберегти у файлі вміст SQL-буфера і має наступний синтаксис:

<команда SAVE> ::=
SAV[**E**] <ім'я файла>[**.**<розширення>] [<режим запису>]
<режим запису> ::=
CRE[**ATE**] | **REP**[**LACE**] | **APP**[**END**]

*Команда **SAVE*** за замовчуванням (або в режимі запису **CREATE**) створює файл із зазначеним ім'ям і записує до нього вміст буфера. У режимі **REPLACE** вміст існуючого файлу заміняється вмістом буфера або файл створюється. У режимі **APPEND** уміст буфера дописується в кінець зазначеного файла.

Якщо розширення не зазначено, передбачається стандартне розширення (**SQL** або задане командою **SET SUFFIX**). Якщо ім'я файла збігається з режимом запису, розширення слід зазначати обов'язково.

Команда **SAVE** додає до командного файла рядок, що він містить символ скісної риси (/).

Команда EDIT

*Команда **EDIT*** дозволяє викликати текстовий редактор базової операційної системи для редагування зазначеного файла або вмісту SQL-буфера. Вона має такий синтаксис:

<команда EDIT> ::=
ED[**IT**] [<ім'я файла>[**.**<розширення>]]

Якщо розширення не зазначено, передбачається стандартне розширення (**SQL** або задане командою **SET SUFFIX**). Файл для редагування відшукується в поточному робочому каталозі. Якщо в ньому такий файл не віднайдено, його створюють. За виклику без параметрів вміст буфера вміщується до файла **afiedt.buf** у поточному робочому каталозі, а потім цей файл завантажується до

текстового редактора. Це стандартне ім'я можна перевизначити за допомогою команди **SET EDITFILE**. Якщо команду викликано без параметрів, а SQL-буфер є порожній, видається таке повідомлення про помилку:

SP2-0107: Немає нічого для зберігання.

Якщо редагувався SQL-буфер, після завершення роботи редактора вміст відповідного файлу автоматично завантажується до буфера. При цьому останній символ останнього рядка (якщо він не є порожній) зрізається.

Ім'я викликуваного текстового редактора утримується в користувацькій змінній SQL*Plus **_EDITOR**. Значення цієї змінної можна задати за допомогою команди **DEFINE**. Якщо її значення не задано, використовується стандартний редактор операційної системи (Notepad у Windows, що задається змінною середовища **EDITOR** або **ed** в UNIX).

Завантаження командних файлів

Будь-якого моменту у ході роботи з SQL*Plus можна завантажити вміст будь-якого текстового файлу до буфера SQL. Для цього використовується команда **GET** з таким синтаксисом:

<команда GET> ::=
GET <ім'я файлу>[**.**<розширення>] [<режим видачі>]
<режим видавання> ::=
LIS[**T**] | **NOL**[**IST**]

Якщо розширення не зазначено, передбачається стандартне розширення (**SQL** або задане командою **SET SUFFIX**).

Файл, що завантажується, повинен містити один оператор SQL або блок PL/SQL. SQL-оператор не повинен завершуватися крапкою з комою (**;**). Зазвичай завантажуються командні файли, створені за допомогою команди **SAVE** – вони автоматично задовольняють даним умовам.

Врахуйте, що команди власне SQL*Plus до SQL-буфера **не потрапляють**, і якщо вони виявляться у файлі, завантаженому за допомогою команди **PUT**, то при виконанні вмісту буфера буде видано повідомлення про помилки – **команди SQL*Plus не є операторами SQL!** Також помилка видається, якщо файл містить кілька операторів SQL або PL/SQL-блоків.

За замовчуванням і в режимі **LIST** вміст завантаженого файла видається на екран. Припинити видавання вмісту командного файлу дозволяє режим **NOLIST**.

Виконання командних файлів

Хоча команда **GET** і дозволяє завантажити уміст файлу до SQL-буфера, звідки воно надалі може бути виконане за допомогою команди /, цей спосіб не є універсальним, тому що накладає істотні обмеження на вміст командного файла (див. вище). Для завантаження й виконання командних файлів, що

містять будь-яку кількість команд SQL*Plus, SQL-операторів та PL/SQL блоків, використовується команда **START** та її скорочені варіанти, **@** та **@@**.

Команда **START** має такий синтаксис:
<команда START> ::=
STA[**RT**] <ім'я або посилання> [<аргумент>{ <аргумент>}]
<ім'я або посилання> ::=
<ім'я файла>[**.**<розширення>] | <URI>

Файл, переданий команді **START** з іменем (або за посиланням на Web-сайт у Oracle9i на платформі Windows), може містити будь-які команди, які можна вводити в діалоговому режимі. Якщо розширення в імені файла не зазначено, передбачається стандартне розширення (SQL або задане командою **SET SUFFIX**).

Зазначений за коротким іменем файл буде спочатку відшукуватися в поточному каталозі, потім – у каталогах, що вони входять до стандартного шляху пошуку (зазвичай задається змінною середовища **SQLPATH**).

Передавання параметрів командних файлів при викликанні

При викликанні командного файла можна передавати параметри у вигляді аргументів командного рядка. Утиліта SQL*Plus підставляє значення аргументів командного рядка замість **позиційних параметрів** командного файла (**&1**, **&2** і т.д.). Перший аргумент підставляється замість параметра **&1**, другий – замість **&2** і т.д.

Скорочені форми команди START (@, @@)

Утиліта SQL*Plus підтримує дві скорочені форми команди **START**: **@** та **@@**. Команда **@** функціонально є аналогічна до команди **START**:

Примітка

Команда **@** видаляє завершальний символ **SQLTERMINATOR** (за замовчуванням – крапка з комою; див. **SET SQLTERMINATOR**) з останньої команди. Якщо цей символ потрібен, слід його подвоїти.

Команда **@@** працює так само, як і **@**, але додатково відшукує командний файл у тім самім каталозі, що й командний файл, в якому вона викликана. Тому дана команда використовується для вкладених викликів командних файлів.

Запам'ятовування установок середовища SQL*Plus

Утиліта SQL*Plus підтримує численні установки, які має сенс запам'ятовувати поміж сеансами. Для цього використовується команда **STORE**:

<команда STORE> ::=
STORE SET <ім'я файла>[**.**<розширення>] [<режим запису>]

Ця команда записує значення змінних середовища SQL*Plus до командного файла базової операційної системи:

SQL> **store set f:\env**
Створено file f:\env

Режим запису вказує, чи буде файл просто створено (**CREATE**, використовується за замовчуванням), переписано, якщо існує (**REPLACE**), або ж значення установок середовища SQL*Plus буде додано в кінець існуючого файла (**APPEND**). Отриманий командний файл може бути виконаний командою **START** або її скороченими формами (**@**, **@@**).

Запис до файла і друкування результатів виконання запитів

Утиліта SQL*Plus дозволяє скидати результати виконання команд до файла і роздруковувати їх на стандартному принтері. Таке скидання називають **спулінгом**. Для цього використовується команда **SPOOL** (керуюча **спулінгом**) з таким синтаксисом:

<команда SPOOL> ::=
SPO[**OL**] [<файл або команда>]
<файл або команда> ::=
<ім'я файла>[**.**<розширення>] | **OFF** | **OUT**

Команда **SPOOL** видає результати виконання команд SQL*Plus до зазначеного файла і, можливо, на стандартний принтер, незалежно від їхнього відображення на екрані. При викликанні без параметрів команда видає стан спулінга. Якщо не зазначене розширення імені файла, використовується стандартне розширення (зазвичай, **LST** або **LIS**).

Команда **OFF** припиняє спулінг. Команда **OUT** припиняє спулінг і посилає файл на стандартний принтер базової операційної системи.

Щоби результати, скидувані до файла, не видавалися на екран, треба виконати команду **SET TERMOUT OFF**.

2.2 Адміністрування в СУРБД Oracle

2.1.1 Користувацькі об'єкти бази даних

Користувацькі об'єкти – це об'єкти системи, що вони не перебувають у винятковому віданні СКРБД Oracle. Усі вони, за винятком файлів даних, є логічними об'єктами бази даних. Це означає, що їхні форма і призначення мають сенс лише в контексті сервера Oracle. Файли даних, на відміну від решти, є реальними файлами операційної системи, а тому їх називають фізичними об'єктами бази даних.

До числа об'єктів користувача належать файли даних, розширювання (extend), табличні простори (tablespace) й сегменти (segments) бази даних.

Файли даних

Файли даних у СКРБД Oracle є єдиними фізичними об'єктами. Кожен файл даних розміщено на одному табличному просторі і він зберігає поточне значення цього простору. Файл даних являє собою реальний файл операційної системи, і з ним можна поводитись, як з будь-яким її файлом. Дані зберігаються в бінарному форматі у файлі Oracle, й, отже, їх можна прочитати лише за допомогою засобів СКРБД Oracle.

Файли даних формуються командами SQL CREATE TABLESPACE та ALTER TABLESPACE. Розмір файла даних установлюється відповідно до параметрів оператора CREATE і не є пов'язаний безпосередньо з даними, що у ньому містяться.

Одержати інформацію про реально існуючі в базі даних файли даних можна за допомогою подавань DBA_DATA_FILES та v$datafile.

Табличні простори

Табличний простір **(Tablespace)** – це логічна модель, що вона використовується для групування даних з однотипними методами доступу. До табличного простору належать один чи кілька файлів даних. При формуванні об'єктів бази даних слід зазначати табличний простір, якому вони належатимуть. Після цього дані (які, власне, й утворять об'єкт) зберігатимуться у файлі даних, який належить до зазначеного табличного простору.

Табличні простори використовуються для поділу операцій уведення-виведення, які виконуються в перебігу доступу до даних. Наприклад. один табличний простір може бути сформовано для зберігання об'єктів даних, а інший – для зберігання об'єктів індексів. Якщо виокремити для цих табличних просторів файли даних на різних фізичних дисках, можна бути впевненим, що звертання до об'єктів індексів не буде перетинатися зі звертанням до даних, на які ці індекси зазначають.

Інформація про табличні простори міститься в поданні DBA_TABLESPSCES.

Блоки Oracle

Блоки Oracle є елементами нижчого рівня ієрархії зберігання даних у системі, являючи собою найменші адресовані сервером одиниці зберігання. Їх не слід плутати з блоками операційної системи; хоча блоки Oracle і формуються з блоків операційної системи, це не є те саме.

Доступ до даних виконується в термінах блоків Oracle. При виконанні операції введення-виведення СКРБД Oracle зчитує стільки байтів даних, скільки входить до одного блока Oracle. Обсяги об'єктів бази даних і розміри блоків у кеш-буфері також встановлюються відповідно до розміру блока Oracle.

Розмір блока Oracle встановлюється при створюванні бази даних і не може бути змінений надалі.

У кожнім блоці Oracle передбачено місце для його заголовка, майбутніх відновлень даних у блоці і реально існуючих рядків даних. У кожнім блоці виділяється місце для подальшого відновлення даних у його рядках. Це місце буде використано в тому різі, якщо в перебігу відновлення даних якого-небудь рядка вона збільшується за розміром.

Можливість долучати нові рядки до складу блока контролюється параметрами PCTFREE і системи зберігання інформації.

Параметр PCTFREE контролює виділення пам'яті (у відсотках) для подальшого відновлення даних у блоці. Наприклад, якщо значення параметра PCTFREE дорівнює 30 %, тобто 70 % обсягу блока буде використано для нових рядків. Коли ці 70 % буде заповнено, Oracle викреслить блок зі списку вільних (доступних для нових рядків), а решту 30% зберігатиме для можливих надалі відновлень уже долучених у нього рядків.

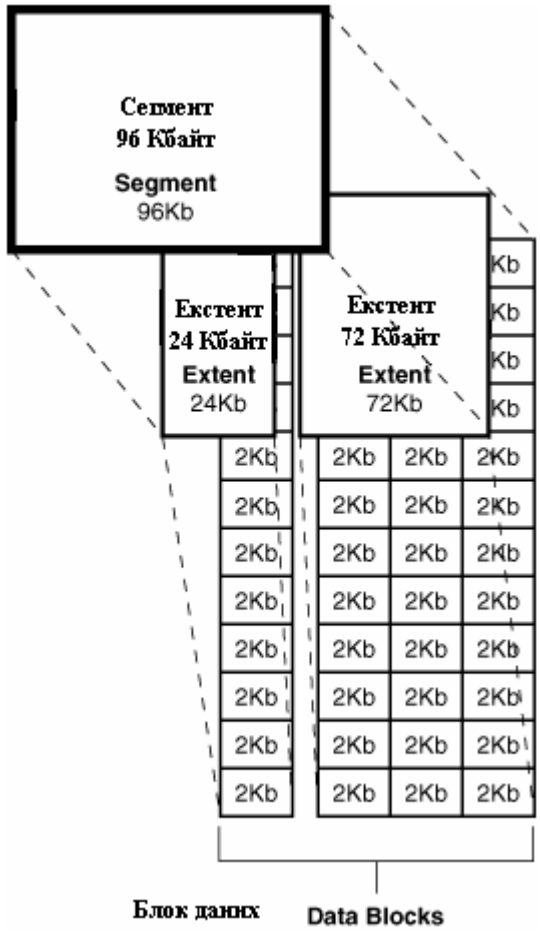

Рисунок 2.1 – Логічна структура табличного простору бази даних

Параметр PCTUSED задає частину обсягу блока, яка має звільнитись, перш ніж блок знову буде долучено до списку доступних для введення нових рядків. Звільнення простору блока є можливе внаслідок вилучання рядків чи

скорочування обсягу інформації, що зберігається в них, після відновлення даних.

Спільне настроювання параметрів PCTFREE та PCTUSED має забезпечувати як можливість безперешкодного відновлення інформації в рядках, долучених до блока, так і його раціональне використання за зменшення обсягу даних, які зберігаються.

Розширювання (Extents)-екстенти

Розширювання (extents) – це об'єкти інформаційної структури зберігання даних, які утворюються неперервними блоками Oracle (одним чи кількома).

Простір для розширювання надається екстентові при створюванні об'єкта відповідно до параметрів команди CREATE. Коли виявляється, що нові дані не може бути розміщено у вже створені розширювання (extents), має бути виокремлено простір для нового розширювання.

Сегмент (Segment)

Сегмент (segment) – це набір екстентів, відведених для зберігання логічної структури. Сегмент – це частина простору, відведена для зберігання даних таблиці, індексу, сегмента відкочування чи тимчасового об'єкта. Коли об'єктові бази даних не вистачає місця в його сегменті для введення нових даних, Oracle надає йому додатковий простір у вигляді екстента.

Кожен сегмент бази даних складається з одного чи кількох розширювань. Розмір розширювань (екстентів) в сегменті бази даних може бути таким самим , як і решти, чи відрізнюватись від них.

Сегменти – це створені користувачами об'єкти, що вони зберігаються в базі даних.

На додаток до створених користувачем сегментів даних та індексів, два типи сегментів розглядаються як системні (чи створювані адміністратором бази даних). Це – тимчасовий сегмент та сегмент відкочування (rollback). Зазвичай ці сегменти створюються адміністратором і потім передаються до загального користування.

Сегменти та екстенти складаються з блоків даних, а ті, своєю чергою, в сукупності утворюють **файл даних (date file).** Для параметра ініціалізації DB_BLOCK_SIZE зазначається певне значення в байтах. Це розмір кожного блока Oracle. Зазвичай розмір блока є кратний розмірові блока операційної системи. Блоки Oracle становлять, як правило, 2K, 4K, 8K и іноді 16K.На рис. 2.1 показано логічну структуру табличного простору бази даних.

2.2.2 Керування блоками Oracle

Найменшою адресованою сервером Oracle одиницею є блок. Усі сегменти (segment) бази даних складаються з екстентів, які своєю чергою складаються з блоків, а структура блока завжди є однакова, незалежно від того, є сегмент таблицею, індексом, кластером чи то іншим об'єктом.

Проектування бази даних з оптимальними характеристиками розпочинається з підбору конфігурації й адміністрування блоків Oracle.

Кожен блок складається з п'яти розділів:

- Заголовок (Header). Містить загальну інформацію, таку як адреса блока й тип сегмента.
- Каталог таблиць (Table directory). Містить інформацію про таблиці, які мають дані в блоці даних.
- Каталог рядків (Row directory). Містить інформацію про рядки в блоці даних.
- Вільний простір (Free space). Простір відведено для відновляння існуючих рядків і введення нових. Два параметри керування простором – PCTFREE та PCTUSED – забезпечують керування використанням вільного простору.

У заголовку блока міститься інформація про нього: сегмент якого типу зберігається в блоці, дані яких сегментів входять до цього блока, адреса блока і покажчики на реальні рядки, зберіганні в ньому.

Заголовок блока складається з фіксованої та змінної частин і посідає від 85 до 100 байт.

Керування областю зберігання даних та областями вільного простору в межах блока є тісно пов'язані. Рядки, що зберігаються в блоці, становлять область даних. Резервна вільна область являє собою фрагмент простору блока, зарезервованого для майбутніх відновлень рядків, зберіганих у блоці. Розмір резервної вільної області визначається у відсотках від розміру блока.

Параметри PCTFREE та PCTUSED

Параметри PCTFREE та PCTUSED – це параметри налаштовування підсистеми зберігання.

Якщо інформація має бути записана до блока бази даних, то СКРБД, аналізує значення параметрів PCTFREE та PCTUSED й поточні характеристики блока, та з'ясовує, чи можливе є додавання нової інформації до блока.

Якщо відносний розмір простору в резервній області є більше, аніж значення параметра PCTFREE, блок може бути використано для запису нової інформації, інакше блок буде вважатися заповненим і нові дані до нього додаватись не будуть.

Лише-но відносний обсяг використовуваного простору в блоці стане менше за значення параметра PCTUSED, блок знову використовуватиметься СКРБД для додавання даних.

Цей метод дозволяє СКРБД Oracle зберігати достатній обсяг додаткового простору для додавання інформації в межах блока, а компактне розміщення рядків у блоці, своєю чергою, допомагає забезпечувати високий рівень продуктивності СКРБД.

Якщо вільний простір у блоці є менше, аніж це окреслено параметром PCTFREE, блок вилучається зі списку доступних. Це триватиме доти, поки межа вільного простору не перейде оцінку, задану параметром PCTUSED внаслідок виконання операцій вилучання рядків, зберіганих у блоці, чи відновлення, що

зменшує довжину рядків у блоці. Параметр PCTUSED визначає відносний розмір вільного простору (у відсотках), по досягненні якого до блока можуть додаватися нові рядки.

Значення параметрів PCTFREE та PCTUSED ніколи не повинні сягати 100 %. Окрім того, за близьких значень цих параметрів блок буде постійно вилучатися зі списку вільних блоків і долучатися до нього за найменшої зміни вмісту.

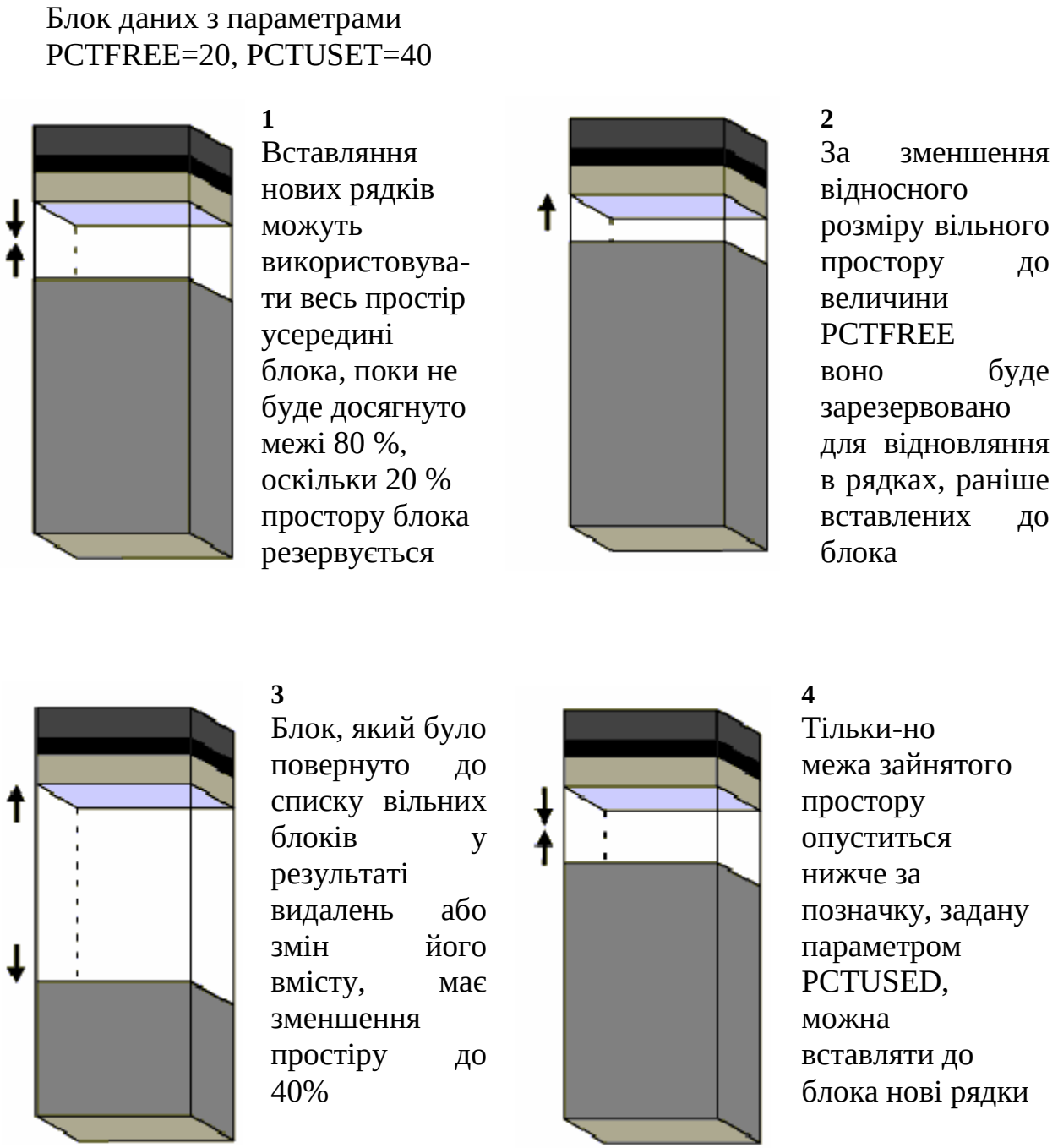

Рисунок 2.2 – Взаємодія параметрів PCTFREE та PCTUSED при керуванні використанням простору всередині блока

Цього можна уникнути, якщо залишити поміж значеннями параметрів PCTFREE та PCTUSED зазор принаймні у 20 %, значення параметрів за замовчуванням дорівнюватимуть 10 % для параметра PCTFREE й 40 % – для параметра PCTUSED.

Віднайти поточне значення для параметрів PCTFREE та PCTUSED можна, використовуючи подавання словника даних dba_tables, dba_clusters та dba_index.

За допомогою параметрів PCTFREE та PCTUSED можна точно набудовувати специфічні параметри зберігання окремих таблиць.

На рис. 2.2 подано взаємодію параметрів PCTFREE та PCTUSED при керуванні використанням простору всередині блока. У цьому прикладі значення параметра PCTFREE дорівнює 20 %, а значення параметра PCTUSED 40 %.

На схемі 1 зазначено, що вставки можуть використовувати весь простір всередині блока, поки не буде досягнуто межі 80 %, тому 20 % простору блока резервується для подальших відновлень, що зазначається параметром PCTFREE.

На схемі 2 зазначено, що за зменшення відносного розміру вільного простору до величини параметра PCTFREE його буде зарезервовано для відновлень у рядках, раніше вставлених до блока, а можливість додавання нових до цього блока не припускається.

На схемі 3 подано блок, що його буде повернуто до списку вільних блоків внаслідок вилучань чи змін, а використовуваний простір у блоці зменшиться на 40 % (що зумовлено значенням параметра PCTUSED для цієї таблиці).

На схемі 4 зазначено, що, тільки-но границя зайнятого простору опуститься нижче за оцінку, задану параметром PCTUSED, можна вставляти в блок нові рядки. В міру вставляння нових рядків використовуваний простір буде збільшуватися, допоки його границя не сягне верхньої оцінки, заданої параметром PCTFREE, – і цикл розпочнеться спочатку.

2.3.3 Встановлення параметрів пам'яті

Параметри пам'яті можна встановлювати для таких типів структур логічної пам'яті:

- табличних просторів (що вони діють на кожен сегмент у табличному просторі);
- таблиць, кластерів, знімків та журналів знімків (сегментів даних);
- індексів;
- сегментів відкочування.

Кожна база даних має значення, за замовчуванням для параметрів пам'яті. Параметри пам'яті, які можна встановлювати, перелічено нижче.

Параметр **INITIAL** – визначає розмір (у байтах) першого екстента, що розподіляється при створюванні сегмента.

Параметр **NEXT** – задає розмір у (байтах) наступного (інкрементального) екстента, що він розподіляється для сегмента. Другий екстент сегмента дорівнюватиме заданому значенню NEXT. Після цього кожне чергове значення

NEXT обчислюватиметься як попереднє значення NEXT, помножене на значення (1+PCTINCEASE/100).

Параметр **MAXEXTENTS** – визначає загальну кількість екстентів, включаючи початкового, котру буде розподілено для сегмента.

Параметр **MINEXTENTS** – визначає загальну кількість екстентів, що їх буде розподілено для сегмента при його створюванні. Це дозволяє розподілити великий обсяг пам'яті під час створювання, навіть якщо неперервного простору немає.

Якщо **MINEXTENTS> 1**, то під час створювання сегмента йому надається необхідна кількість екстентів з використанням значень INITIAL, NEXT та PCTINCEASE.

Значення **PCTINCEASE** визначає відсоток, на який збільшується кожен наступний інкрементальний екстент стосовно попереднього, наданого для сегмента. Якщо PCTINCEASE =0, то всі інкрементальні сегменти будуть однакового розміру. Якщо PCTINCEASE >0, то кожне чергове значення NEXT збільшується на PCTINCEASE відсотків. Значення PCTINCEASE не може бути від'ємним.

Значення NEXT обчислюється як попереднє значення NEXT, помножене на (1+PCTINCEASE/100) і округлене у великий бік до найближчого кратного за розміром блока даних.

Для сегментів відкочування PCTINCEASE завжди має нульове значення, що не може бути змінено.

За правильного призначення PCTINCEASE може зменшити фрагментацію сегмента, збільшуючи розміри інкрементальних екстентів і скорочуючи загальну кількість екстентів, призначених для сегмента.

Якщо змінити значення PCTINCEASE для сегмента, то поточне значення NEXT для цього сегмента не зміниться; нове змінювання вплине лише на наступні значення NEXT.

2.4.4 Створювання табличних просторів та керування ними

Для створювання нового табличного простору використовується оператор мови SQL CREATE TABLESPACE.

Синтаксис оператора CREATE TABLESPACE має бути таким:

```
CREATE TABLESPACE tablespace_name
DATAFILE Datafile_Options Storage_Options ;
Datafile_Options:

[AUTOEXTEND OFF]
[AUTOEXTEND ON [NEXT int K | M] [MAXSIZE int K | M]]

Storage_Options:
```

DEFAULT STORAGE
MINIMUM EXTENT int {K|M}
LOGGING | NOLOGGING
ONLINE | OFFLINE
PERMANENT | TEMPORARY
EXTENT MANAGEMENT {DICTIONARY |
LOCAL {AUTOALLOCATE | UNIFORM [SIZE int K | M]} }

CREATE TABLESPACE – оператор створювання табличного простору; **tablespace_name** – найменування табличного простору; DATAFILE – шлях, місце розташування файла бази даних; **Datafile_Options** – опції файла бази даних.

Datafile_Options:

- [AUTOEXTEND OFF] автоматичне розширювання для екстентів є вимкнено.
- [AUTOEXTEND ON [NEXT int K | M] автоматичне розширення для екстентів увімкнено. Наступне (початкове, вихідне, ціле кількість збільшується на розмір у кілобайтах чи в мегабайтах.
- [MAXSIZE int K | M]] увімкнено збільшення розміру файла бази даних у кілобайтах чи в мегабайтах.

Storage_Options – опції зберігання; **DEFAULT STORAGE** – опції зберігання за замовчуванням.

DEFAULT STORAGE:

- INITIAL – визначає розмір першого екстента;
- NEXT – визначає розмір другого екстента;
- MINEXTENTS – визначає мінімальну кількість екстентів, що їх може бути виділено об'єктові в табличній області;
- MAXEXTENTS – визначає максимальну кількість екстентів, що їх може бути виділено об'єктові в табличній області;
- PCTINCREASE – зазначає у відсотках збільшення обсягу простору, виділеному наступному екстентові об'єкта.

MINIMUM EXTENT int {K|M} – забезпечує кратність розміру кожного екстента табличної області зазначеному цілому значенню. Цей параметр перевизначати не можна.

ONLINE – увімкнено (оперативний режим). OFFLINE – вимкнено.

TEMPORARY – тимчасовий табличний простір. PERMANENT – постійний, незмінний табличний простір.

Вилучання табличних просторів

Для вилучання табличних областей використовується команда SQL DROP TABLESPACE. Якщо табличний простір містить таблиці з первинними

чи унікальними ключами, на які посилаються зовнішні ключі таблиць з інших табличних просторів, то слід зазначити каскадне вилучання обмежень CASCADE CONSTRAINTS.

2.2.5 Створювання й керування обліковими записами користувача

Керування обліковими записами користувача (створювання, змінювання і вилучання) зазвичай виконує адміністратор бази даних.

Для створювання облікового запису користувача використовується команда мови SQL CREATE USER.

Для створювання облікового запису користувача треба мати системний привілей CREATE USER.

Роль адміністратора бази даних (DBA) призначається користувачеві за допомогою облікового запису SYSTEM. Користувач з цим обліковим записом стає власником додатка, створюючи об'єкти й облікові записи інших користувачів.

Для створення облікового запису користувача треба підімкнутися до бази даних (до СКРБД Oracle) за допомогою програми SQL*PLUS і видати таку команду SQL:

```
SQL> DROP USER leader14;

        User dropped.

SQL> CREATE USER leader
        2  IDENTIFIED by leader
        3  DEFAULT tablespace users
        4  TEMPORARY tablespace temp
        5  Profile default;

        User created.
```

При створюванні облікового запису користувача зазначається його пароль, надається використовуване за замовчуванням табличний простір та тимчасовий табличний простір.

У даному коді створювання облікового запису користувача фраза IDENTIFIED by свідчить про те, що впізнання користувача відбувається за паролем. При використовуванні впізнання за паролем слідом за IDENTIFIED by має бути наведено пароль (відбивається на екрані), який зазначає зазнає, який пароль має увести користувач для одержання доступу до бази даних.

Цей метод відрізняється від пізнавання операційною системою, за якого користувач орієнтується поза базою даних (IDENTIFIED EXTERNALLY).

Використовуваний за замовчуванням табличний простір (DEFAULT tablespace) застосовується для розміщування створюваних користувачем об'єктів, якщо для них явно не зазначено іншого табличного простору.

Тимчасовий табличний простір (TEMPORARY tablespace) необхідний при використанні команд SQL, яким слід надати певний тимчасовий сегмент для сортування в перебігу звичайної SQL команди – опрацьовування даних.

При використовуванні компонента SQL*Plus пароль задається за допомогою ключових слів IDENTIFIED BY.

При створюванні нового користувача для нього можна специфікувати квоти для будь-яких табличних просторів.

Кожен користувач має табличний простір за замовчуванням, що зазначається при його створюванні фразою **DEFAULT TABLESPACE**. Коли користувач створює об'єкт і не зазначає табличний простір, СКРБД Oracle зберігає за замовчуванням цей об'єкт у табличному просторі даного користувача.

Вирішуючи, який табличний простір призначити користувачеві, пропонується враховувати такі рекомендації:

- установлювати табличний простір за замовчуванням лише в тому разі, якщо користувач має привілеї на створювання об'єктів (таких як таблиці, подання, кластери) і має на нього квоту;
- якщо можливо, табличний простір за замовчуванням має бути відмінним від SYSTEM.

При виконуванні користувачем пропозиції SQL, котра вимагає тимчасового сегмента, Oracle зберігає цей сегмент у тимчасовому табличному просторі даного користувача.

У СКРБД Oracle в якості тимчасового табличного простору для користувача за замовчуванням призначається табличний простір TEMP. Установити тимчасовий табличний простір можна під час створювання користувача і змінити його пізніше.

При створюванні користувача можна призначити для нього квоту для будь-якого табличного простору. Призначення квоти переслідує дві мети:

- якщо користувач має привілеї створювати об'єкти певного типу, він може створювати об'єкти в зазначеному табличному просторі;
- СКРБД Oracle обмежує обсяг пам'яті, що її може бути розподілено поміж об'єктам користувача в надаваному табличному просторі. Це обмеження визначається квотою.

У наведеному вище прикладі створювання облікового запису користувача leader йому привласнено табличний простір за замовчуванням – USERS та тимчасовий табличний простір – TEMP. Ці, задані за замовчуванням, табличні простори USERS та TEMP **не створюються окремо**. Вони створюються при інсталяції СКРБД Oracle, у перебігу створювання завантажувальної бази даних.

2.2.6 Надавання та скасування об'єктних привілеїв

На виконання кожної дозволеної в Oracle дії існує спеціальний привілей. У Oracle привілеї бувають двох типів: **системні привілеї** (system privileges) – для виконання адміністративних завдань, наприклад для створювання більшості об'єктів, та **об'єктні привілеї** (object privileges) – для звертання до цих об'єктів із запитами, для введення до них даних, для змінювання і вилучання даних у них. Всі привілеї надаються командою **GRANT**. Щоби надати об'єктний привілей, треба самому мати привілей, наданий з параметром with grant option, чи володіти об'єктом. Той, хто надає об'єктний привілей, повинен визначити рівень доступу до об'єкта, необхідний користувачеві. Після цього можна надавати привілей.

Доступні об'єктні привілеї

Після створювання об'єкта в базі даних Oracle його привілеями розпоряджається його власник. Суть керування цим об**'**єктом полягає в надаванні привілеїв, які надавали б користувачеві можливість маніпулювати об'єктом: додавати до нього дані, змінювати, вилучати і переглядати їх. До об'єктних привілеїв належать:

- **SELECT –** дозволяє звертатися до даних таблиці, послідовності, подання чи моментального знімка.
- **INSERT –** дозволяє вводити дані до таблиці, а іноді й до подавань.
- **UPDATE –** дозволяє оновлювати дані в таблиці чи в поданні.
- **DELITE –** дозволяє вилучати дані з таблиці чи подання.
- **ALTER –** дозволяє змінювати опис лише таблиці чи послідовності; привілеї alter на решту об'єктів бази даних вважаються за системні.
- **INDEX –** дозволяє створювати індекс для раніше описаної таблиці.
- **REFERENSES –** дозволяє створювати й змінювати таблицю, встановлюючи обмеження зовнішнього ключа для даних таблиці, на яку здійснюється посилання.
- **EXECUTE –** дозволяє виконувати процедуру, що вона зберігається, чи функцію.

Об'єктні привілеї на будь-який об**'**єкт бази даних належать користувачеві, що він створив об**'**єкт. Їх можна надавати іншим користувачам, щоби дозволити їм звертатися до об**'**єкта й маніпулювати ним або керувати привілеями для решти користувачів. Можливість виконувати певні адміністративні функції над об'єктними привілеями надається вказівкою спеціального параметра WITH GRANT OPTION.

Надавання об'єктних привілеїв в Oracle

Привілей, наданий користувачеві, надає йому можливість розпоряджатися ним, якщо зазначено параметр **with grant option.** До адміністративних функцій, виконуваних над об'єктним привілеєм, належать надавання привілею чи його скасовування, а також надавання об'єктного привілею іншому користувачеві з можливістю виконання ним адміністративних функцій над цим привілеєм.

GRANT select, update, insert ON employee TO leader14_01;
GRANT select, update, insert ON employee TO leader14_02 WITH GRANT OPTION;

Об'єктні привілеї скасовуються командою **revoke**. Якщо користувачеві надано привілей з параметром **with grant option**, то команда revoke позбавляє його можливості виконувати дії, контрольовані цим привілеєм, і можливості розпоряджатися ним. Але, окрім позбавлення користувача можливості застосовувати привілей, зі скасовуванням об'єктних привілеїв пов'язано ціла низка питань, стосовних каскадування. Припустімо, що користувач leader14_01 створює таблицю EMPLOYEE і вводить до неї кілька рядків. Потім він надає привілей select з параметром with grant option на таблицю EMPLOYEE користувачеві leader14_02. Після цього користувач leader14_02 надає те ж саме користувачеві leader14_03. Якщо потім користувач leader14_01 скасує привілей користувача leader14_02, то скасовується і привілей користувача leader14_03. Окрім того, за вилучення таблиці буде вилучено і всі об'єктні привілеї, надані користувачам на цю таблицю.

2.2.7 Керування привілеями і ролями

Привілеї розроблювача програмного забезпечення

Наведений у табл. 2.4 набір привілеїв стосується розроблювачів програмного забезпечення, котрі можуть працювати з СКРБД Oracle.

Таблиця 2.4 – Привілеї розроблювача програмного забезпечення

Привілей	Опис
1	2
CREATE CLUSTER	Створює кластер з таблиць, що вони належать розроблювачеві. Розроблювачі можуть також вилучати належні їм кластери
CREATE PROCEDURE	Створює зберіганні процедури, пакети й функції, що вони належать розроблювачеві. Розроблювачі можуть також вилучати кожний з об'єктів, якими вони володіють

Закінчення таблиці 2.4

Привілей	Опис
CREATE DATABASE LINK	Визначає канал зв'язку бази даних. Цей засіб є подібний до синоніма, оскільки він являє собою іменований покажчик на іншу базу даних. Ключова розбіжність полягає в тому, що канал зв'язку зберігає ідентифікатор Oracle й пароль для вилученої системи, визначаючи тим самим привілеї доступу до цієї системи. Розроблювачі можуть також вилучати будь-які створені ними канали зв'язку бази даних
CREATE PUBLIC SYNONYM	Створює альтернативне ім'я для надсилання на такий об'єкт бази даних, як таблиця чи подання. Цим ім'ям для звертання до даного об'єкта може користуватися будь-який користувач екземпляра. Однак користувач для доступу до об'єкта все одно повинен мати привілеї на цей об'єкт
DROP PUBLIC SYNONYM	Вилучає альтернативне ім'я, застосовуване для надсилання на об'єкт бази даних і доступне для всіх користувачів у цьому екземплярі
CREATE SEQUENCE	Створює послідовність певному розроблювачеві. Розроблювачі можуть також вилучати будь-які створені ними послідовності.
CREATE SNAPSHOT	Створює локальну копію таблиці, розташованої в іншому екземплярі Oracle. Розроблювачі можуть також вилучати будь-які приналежні їм знімки
CREATE SYNONIM	Створює приватний синонім (лише для розроблювача). Розроблювачі можуть також вилучати будь-які приналежні їм приватні синоніми
CREATE TABLE	Створює таблицю. Розроблювачі можуть також створювати для них індекси й вилучати приналежні їм таблиці
CREATE TRIGGER	Створює тригер (команду, виконувану за виникнення певної події), що він належить розроблювачеві. Розроблювачі можуть також вилучати приналежні їм тригери
CREATE VIEW	Створює подання, що воно належить розроблювачеві. Розроблювачі можуть також вилучати приналежні їм подання
CREATE TYPE	Створює новий об'єктний тип
CREATE LIBRARY	Створює нову об'єктну бібліотеку

Керування ролями

Роль – це група системних та об'єктних привілеїв, що їх може бути призначено користувачеві бази даних. Ролі надаються необхідні привілеї, наприклад CREATE PROCEDURE, CREATE TRIGGER, CREATE TABLE, CREATE TYPE. Потім ця роль призначається користувачеві, й він здобуває всі привілеї, надані цій ролі. Не існує обмежень на кількість людей, яким може бути призначено цю роль, а змінювання можна поширити на всіх користувачів бази даних за внесення змін до прав ролі.

Створювання ролей

Для створювання ролі використовується команда SQL CREATE ROLE.

```
SQL> CREATE role DEVELOPER;
Role created.
```

Було створено роль для розроблювача програмного забезпечення (DEVELOPER). Щоби роль була здатна працювати, треба надати цій ролі привілей і призначити її (роль) користувачам. Після створення ролі їй надаються привілеї з використанням команди GRANT, потім надають користувачам доступ до ролі з використанням тієї самої команди GRANT (де як привілей зазначено ім'я ролі).

Надавання привілеїв ролі

Набір привілеїв, перелічених в таблиці 2.4 для розроблювачів програмного забезпечення, було надано ролі DEVELOPER, для того щоби ця роль була здатна працювати.

```
SQL> GRANT CREATE CLUSTER, CREATE PROCEDURE,
  2  CREATE DATABASE LINK,CREATE PUBLIC SYNONYM,
  3  CREATE SEQUENCE, CREATE SNAPSHOT,
  4  CREATE SYNONYM, CREATE TABLE,
  5  CREATE TRIGGER, CREATE VIEW,
  6  CREATE TYPE, CREATE LIBRARY
  7  TO DEVELOPER;

Grant succeeded.
```

Призначання ролі розроблювача програмного забезпечення користувачеві leader14

```
SQL> GRANT DEVELOPER TO leader14;
Grant succeeded.
```

Користувачеві leader14 було призначено роль розроблювача програмного забезпечення DEVELOPER.

Вилучання ролей

Адміністратор бази даних може вилучати роль з бази даних. Ця операція вилучає інформацію, стосовну ролі та всіх існуючих користувачів і ролей. Як і при скасуванні системних привілеїв, вилучання ролі не впливає на існуючі об'єкти бази даних. Для вилучання ролі використовується команда SQL DROP ROLE.

2.2.8 Сеанс доступу до бази даних

Користувацькі привілеї

Ці привілеї належать звичайним користувачам бази даних. Навіть в разі, коли ці користувачі застосовують звичайні й відносно прості функції (читання та запис в існуючих об'єктах), а Oracle передбачено кілька варіантів привілеїв доступу, що дозволяють організаціям зміцнити захист у максимально можливою мірою. В таблиці 2.5 подано користувацькі привілеї.

Таблиця 2.5 – Користувацькі привілеї

Привілей	Опис
CREATE SESSION	Дозволяє користувачеві долучитися до бази даних Oracle. Без неї користувач не має жодного доступу до Oracle
ALTER SESSION	Дозволяє користувачеві видавати команду alter session, що надає можливість виконувати установлення параметрів підтримки національної мови (NLS), керувати каналами зв'язку бази даних і використовувати засоби трасування
FORSE TRANSACTION	Дозволяє користувачеві виконувати фіксацію або відкочування розподіленої транзакції в локальній базі даних

Ролі доступу до СКРБД Oracle.

- **CONNECT**. Користувачі можуть звертатися до даних Oracle відповідно до привласнених їм привілеїв на об'єкти. Вони не можуть створювати об'єкти бази даних чи виконувати певні адміністративні функції.
- **RESOURSE**. Користувачі можуть звертатися до даних Oracle відповідно до привласнених їм привілеїв на об'єкти, а також створювати такі об'єкти бази даних, як таблиці та подання. Вони не можуть виконувати певні адміністративні функції бази даних.

Надавання привілеїв доступу до СКРБД Oracle

Створимо привілеї для підмикання користувача, наприклад з ім'ям leader14 до СКРБД Personal Oracle 81798.

```
SQL> GRANT CREATE SESSION, ALTER SESSION to leader14;
Grant succeeded.
SQL> GRANT CONNECT, RESOURCE to leader14;
Grant succeeded.
```

Для підмикання користувача до СКРБД Personal Oracle 81798 в програмі SQL*Plus використовується команда **CONNECT.**

Для перевірки можливості підмикання користувача leader14 до СКРБД Personal Oracle 81798 виконаємо команду **CONNECT:**

```
SQL> connect leader14/alex14;
Connected.
```

Сеанс доступу до СКРБД Personal Oracle 81798 відбувся!

Для відключення користувача з бази даних використовується команда **DISCONNECT.**

Після вимкнення користувача з бази даних, ви ще можете користуватися програмними засобами зв'язку з базою даних, але сам зв'язок з базою даних буде припинено.

При використанні для розриву зв'язку оператора EXIT припиняється не лише ваш сеанс SQL, але й закривається програма (SQL*Plus), за допомогою якої здійснювався доступ до бази даних.

2.3 Створювання таблиць та обмежень

2.3.1 Створювання таблиць. Оператор CREATE TABLE

Оператор CREATE TABLE.

Призначення. Створює таблицю – головну структуру, призначену для зберігання користувацьких даних. Синтаксис оператора **CREATE TABLE** має такий вигляд:

```
CREATE TABLE [ схема. ] таблиця
( ( стовпець тип_даних [ DEFAULT вираз]
[ обмеження_стовпця]...
| обмеження_таблиці)...
[, ( стовпець тип_даних [ DEFAULT вираз]
[ обмеження_стовпця]...
```

| обмеження_таблиці)]...)
[[PCTFREE ціле] [RCTUSED ціле]
[INITRANS ціле] [MAXTRANS ціле]
[STORAGE storage_фраза]
| [CLUSTER кластер (стовпець [, стовпець]...)]]
[ENABLE enable_фраза
| DISABLE disable_фраза]...
[AS підзапит]

Оператор CREATE TABLE визначає таку інформацію:

- визначення стовпців;
- обмеження цілісності;
- табличний простір для таблиці;
- характеристики пам'яті;
- кластер (необов'язковий параметр);
- дані з довільного запиту.

Умови. Щоби створити таблицю у власній схемі, ви повинні мати системний привілей CREATE TABLE. Щоби створити таблицю в схемі іншого користувача, ви повинні мати системний привілей CREATE ANY TABLE. Окрім того, власник схеми, в якій створюється таблиця, повинен мати квоту пам'яті в табличному просторі, що він містить таблицю, або системний привілей UNLIMITED TABLESPACE.

Ключові слова й параметри

Схема. Схема, в якій створюється таблиця. Якщо схему опущено, то Oracle створює таблицю у вашій схемі.

Таблиця. Ім'я створюваної таблиці.

Стовпець. Визначає ім'я стовпця. Кількість стовпців у таблиці може варіюватись від 1 до 254.

Тип даних. Тип даних стовпця.

DEFAULT. Визначає значення, що замовчується, яке має призначатися стовпцеві, якщо в пропозиції INSERT, котра вставляє дані до таблиці, буде відсутнє значення для цього стовпця.

Обмеження стовпця. Визначає обмеження цілісності як частину визначення стовпця.

Обмеження таблиці. Визначає обмеження цілісності як частину визначення таблиці.

PCTFREE. Визначає, який простір треба залишити вільним у кожнім блоці Oracle для подальшого збільшування рядків при модифікуванні даних. Параметр відсоток пам'яті, який зарезервовано у кожнім блоці даних таблиці, для майбутніх відновлень рядків таблиці. Значення PCTFREE повинне бути позитивним цілим від 1 до 99. Значення 0 дозволяє цілковито заповнити блок

вставляннями нових рядків. Значення за замовчуванням дорівнює 10. Це значення резервує 10 % пам'яті в кожнім блоці для відновлень існуючих рядків і дозволяє вставлянням нових рядків заповнювати максимально 90 % місця в кожнім блоці. Комбінація PCTFREE та PCTUSED визначає, чи будуть рядки, що вставляються, потрапляти до існуючих блоків даних, чи то для них будуть потрібні нові блоки.

PCTUSED. Параметр PCTUSED задає для таблиці рівень, до якого повинен зменшуватися обсяг пам'яті, використовуваної в блоці (за рахунок рядків що вилучаються й модифікуються), перш ніж буде дозволено вставляти до цього блока нові рядки. Параметр PCTUSED визначає **мінімальний** відсоток використовуваної пам'яті, що його Oracle підтримує для кожного блока даних. Блок стає кандидатом на вставляння нових рядків, коли його використовувана пам'ять потрапляє нижче за PCTUSED. Параметр PCTUSED задається як позитивне ціле число від 1 до 99; значення за замовчуванням дорівнює 40 %.

Сума PCTFREE та PCTUSED повинна бути менше за 100. Ви можете задавати і змінювати значення PCTFREE та PCTUSED разом, щоби домогтися більш ефективного використання пам'яті усередині таблиці.

INITRANS. Визначає початкову кількість входів транзакцій, даних розподілюваних в кожнім блоці, розподілюваним в таблиці.

Це значення може варіюватися від 1 до 255; значення за замовчуванням дорівнює 1. Як правило, ви не повинні задавати значення INITRANS, відмінне від замовчуваного.

MAXTRANS. Визначає максимальну кількість одночасних транзакцій, які можуть обновлювати блок даних у таблиці. Цей ліміт не застосовується до запитів. Це значення може варіюватися від 1 до 255; значення за замовчуванням залежить від розміру блока даних. Ви не повинні задавати значення MAXTRANS, відмінне від замовчуваного.

Табличний простір (TABLESPECE). Визначає табличний простір, в якому Oracle створює таблицю. Якщо цю опція опущено, то Oracle створює таблицю в табличному просторі за замовчуванням власника схеми, в якій створюється таблиця.

STORAGE. Визначає характеристики пам'яті для таблиці. Ця фраза має певні варіанти продуктивності для великих таблиць. Слід розподіляти пам'ять у такий спосіб, щоб мінімізувати динамічний розподіл додаткової пам'яті.

CLUSTER. Зазначає, що таблиця має бути частиною кластера. Стовпці, перелічені в цій фразі, називають ті стовпці таблиці, котрі відповідають стовпцям ключа кластера. Зазвичай ці стовпці становлять первинний ключ таблиці або частину її первинного ключа. Оскільки кластеризована таблиця, використовує розподіл пам'яті кластера, не використовуйте з опцією CLUSTER параметри PCTFREE, PCTUSED, INITRANS та MAXTRANS, а також опцію TABLESPACE.

ENABLE. Вмикає обмеження цілісності.

DISABLE. Вимикає обмеження цілісності. Обмеження, обумовлені фразами ENABLE та DISABLE пропозиції CREATE TABLE, повинні бути

визначені в самій цій пропозиції. Ви можете також вмикати та вимикати обмеження за допомогою ключових слів ENABLE та DISABLE фрази CONSTRAINT. Якщо ви визначаєте обмеження, але явно не вмикаєте й не вимикаєте його, то Oracle за замовчуванням вмикає це обмеження. Не можна використовувати фрази ENABLE та DISABLE у пропозиції CREATE TABLE для вмикання або вимикання тригерів.

AS підзапит. Вставляє до таблиці при її створюванні рядки, що повертаються підзапитом. Кількість стовпців у таблиці має збігатися з кількістю виразів у підзапиті.

Використання оператора CREATE TABLE

Таблиці створюються порожніми (без даних), якщо не використовується підзапит. Ви можете додати рядки до таблиці за допомогою команди INSERT.

Іменування таблиць та стовпців

Перелічені нижче правила є обов'язкові для будь-якої таблиці чи стовпця.

- Максимальна довжина імені таблиці чи стовпця дорівнює 30 символам.
- Імена таблиць та стовпців можуть містити літери, цифри й символ підкреслювання (_)
- Імена таблиць та стовпців повинні розпочинатися з літерного символу. Ім'я може містити цифри чи символи підкреслювання, але в будь-якому разі має розпочинатися з літери.
- Символи верхнього й нижнього регістрів в іменах таблиць та стовпців вважаються за однакові.
- Ім'я таблиці чи стовпця не повинні містити прогалини.
- У СКРБД Oracle таблиці привласнюються користувачам; за замовчуванням вони привласнюються тому користувачеві, що він їх створив. Кожна з таблиць повинна мати ім'я, відмінне від імен інших таблиць цього користувача. Усі стовпці в межах таблиці повинні мати унікальні імена.
- Команди й зарезервовані слова Oracle не можуть бути іменами таблиць та стовпців.

Опис таблиць

У Oracle є команда, яка визначає структуру існуючої таблиці. Вона називається DESCRIBE (скорочено – DESC) і має такий синтаксис:

DESC ім'я_таблиці

Це одна з кількох команд, які не вимагають завершувати крапкою з комою.

Щоб побачити, як працює команда DESC, слід впровадити такий код:

```
DESC emp
```

Стовпці NULL та NOT NULL

При розроблянні таблиць для зберігання інформації певного призначення ви можете в доволі широких межах контролювати, що може, а що не може потрапляти до таблиць. Одним з рішень, яке слід ухвалити в першу чергу, є рішення про те, які стовпці запису мають містити дані, а які можуть залишатися порожніми.

За замовчуванням усі стовпці таблиці вважаються необов'язковими. Це означає, що ви можете вводити записи, які мають дані не у всіх стовпцях.

Стовпець, що він не містить даних, має непевне значення (NULL). Нуль та прогалину не можна розглядати як (null).

Нуль та прогалина є реальними значеннями, які залежно від контексту можуть враховуватися чи не враховуватися.

Непевний стовпець (NULL) не містить узагалі жодного значення.

Зазначити, які стовпці запису повинні містити дані, тобто бути обов'язковими, можна в команді CREATE TABLE. У цій команді стовпець з'являється шляхом додавання слів "NOT NULL" після його імені й довжини.

```
CREATE TABLE DEPT
 (DEPTNO NUMBER(2) NOT NULL,
DNAME VARCHAR2(14),
LOC VARCHAR2(13) );
```

У наведеному прикладі створення таблиці DEPT в операторі CREATE TABLE для стовпця DEPTNO NUMBER(2) з'являється NOT NULL, тобто він є обов'язковий і повинен містити дані.

Демонстраційні таблиці EMP й DEPT

Стандартні демонстраційні таблиці EMP та DEPT, які інсталюються разом із середовищем СКРБД Oracle, і є своєрідною вбудовуванню базою даних Oracle. Вони містять мінімальну кількість даних та стовпців тому чудово слугують за якості навчальний приклад.

За замовчуванням, ці демонстраційні таблиці належать користувачеві з обліковим записом **scott**.

Bruce Scott був одним з перших розроблювачів корпорації Oracle.

У додатку "А" на методичний посібник до лабораторних робіт №1-5 є файл з ім'ям DEMOBLD.SQL, який містить коди створювання і заповнювання даними демонстраційних таблиць. Кожен користувач, зареєстрований у СКРБД Oracle, може створювати, заповнювати даними, змінювати ці демонстраційні таблиці, використовуючи вміст файла DEMOBLD.SQL.

2.3.2 Типи даних і опис стовпців

При записуванні операторів SQL слід керуватися одним загальним правилом – не змішувати типи даних.

Опис типів даних:

Тип даних	**Опис**
VARCHAR2	Текстові рядки змінної довжини до 4000 байт (в ORACLE8i)
NVARCHAR2	Одно- й багатобайтові текстові рядки змінної довжини до 4000 байт (в Oracle8i)
CHAR	Фіксовані текстові рядки до 2000 байт (в Oracle8i)
NCHAR	Одно- й багатобайтові текстові рядки фіксованої довжини до 2000 байт (в Oracle8i)
NUMBER	Числові дані
DATE	Дати
RAW	Двійкові дані до 2000 байт (в Oracle8i)
LONG	Текстові дані до 2 Гбайт
LONG RAW	Двійкові дані до 2 Гбайт
ROWID	Адреси рядків таблиці
BLOB	Великі двійкові об'єкти (лише в Oracle8i)
CLOB	Великі символьні об'єкти (лише в Oracle8i)
NCLOB	Великі одно- й багатобайтові символьні об'єкти (лише в Oracle8i)
BFILE	Великі зовнішні файли (лише в Oracle8i)

Типи даних NUMBER, DATE, VARCHAR2, CHAR
Тип даних NUMBER

За допомогою типів цього сімейства зберігаються цілі й дійсні значення.

За допомогою даного типу можна зберігати числа, як цілі, так і з плаваючою крапкою. Синтаксис оголошення певного числа є такий:

NUMBER (P,S);

де P – точність (precision), а S – масштаб (scale). **Точність** – це кількість цифр у значенні; **масштаб** – кількість цифр праворуч від десяткової крапки. Використання як точності, так і масштабу не є обов'язкове, однак при

зазначенні масштабу слід зазначати й точність. Різні комбінації точності й масштабу продемонстровано в таблиці 2.6.

Таблиця 2.6 – Значення точності й масштабу

Оголошення	Привласнене значення	Збережене значення
NUMBER;	1234.5678	1234.5678
NUMBER(3);	123	123
NUMBER(3);	1234	Помилка – перевищення точності
NUMBER(4,3);	123.4567	Помилка – перевищення точності
NUMBER(4,3);	1.234567	1.235
NUMBER(7,2);	1234.67	12345.67
NUMBER(3,-3);	1234	1000
NUMBER(3,-1);	1234	1230

1 Якщо масштаб присвоюванного значення, перевищує зазначений, то збережене значення округлюється до числа цифр, зазначеного в масштабі.

2 Якщо масштаб є від'ємний, то збережене значення округлюється до числа цифр, зазначеного в масштабі.

Максимальною точністю є 38 цифр, а масштаб може перебувати в діапазоні від –84 до 127.

Тип даних DATE

Тип DATE використовується для зберігання інформації як про дати, так і про час, у тому числі про сторіччя, роки, місяці, дні, години, хвилини й секунди. Розмір змінної DATE становить 7 байт, по одному байтові на кожен компонент (від сторіччя до секунди).

Значення змінним DATE зазвичай привласнюються за допомогою вбудованої функції TO_DATE. Це дозволяє легко перетворювати символьні змінні на змінні DATE. Подібно до цього за допомогою функції TO_CHAR можна перетворювати змінні DATE на символьні змінні.

Тип даних VARCHAR2

За допомогою змінних типу VARCHAR2 можна зберігати рядки символів змінної довжини. Синтаксис оголошення змінної, яка має тип VARCHAR2, є такий:

VARCHAR2(L);

де L – максимальна довжина (length) змінної. Зазначення довжини є обов'язкове: – значення за замовчуванням не існує. Максимальна довжина змінної типу VARCHAR2 становить 32 767 байт. Зверніть увагу, що в полі стовпця бази даних, який має тип VARCHAR2, можна зберігати лише 2000 байт. Якщо довжина змінної типу VARCHAR2 перевищує 2000 байт, її можна увести лише до стовпця бази даних, який має тип LONG, максимальний розмір якого становить 2 Гбайт. Аналогічно, дані LONG не можна ввести до змінної VARCHAR2, якщо їхній розмір перевищує 32 767 байт.

У Oracle8 в полі стовпця бази даних, який має тип VARCHAR2, можна зберігати 4000 байт. Отже, змінна VARCHAR2 може бути введена до стовпця Oracle8 VARCHAR2, лише якщо її довжина не перевищує 4000 байт.

Підтип VARCHAR є еквівалентний до типу VARCHAR2.

Тип даних CHAR

Змінні цього типу являють собою рядки символів фіксованої довжини. Синтаксис оголошення змінної CHAR є такий:

CHAR(L);

де L – максимальна довжина в байтах. Однак, на відміну від типу VARCHAR2, у цьому разі зазначення довжини не є обов'язкове. Якщо довжину не зазначено, приймається значення 1 за замовчуванням, причому круглі дужки не потрібні. Змінні типу CHAR мають фіксовану довжину, тому за необхідності вони заповнюються до максимальної довжини прогалинами. Внаслідок цього змінні типу CHAR не завжди будуть збігатися при виконанні операцій порівнювання символів.

Максимальна довжина змінної типу CHAR дорівнює 32767 байт. Максимальна ж довжина поля стовпця бази даних, який має тип CHAR, становить 255 байтів. Отже, якщо у змінній CHAR міститься понад 255 байт, її можна увести лише до стовпця типу VARCHAR2 чи LONG.

Аналогічно, дані типу LONG можна внести до змінної типу CHAR лише в разі, якщо їхній розмір становить не більше за 32767 байт.

У Oracle8 в полі стовпця бази даних, який має тип CHAR, можна зберігати до 2000 байт.

Як і для VARCHAR2, довжина змінної типу CHAR, зазначається не в символах, а в байтах. Якщо в певному наборі символів бази даних містяться багатобайтові символи, то максимальна кількість символів, котрі можуть зберігатися у змінній типу VARCHAR2, можливо, буде менше за зазначену довжину.

Підтипом CHAR, який має такі самі обмеження, є CHARACTER. Семантика змінних VARCHAR2 та змінних CHAR істотно розрізнюється.

Типи даних CLOB, NCLOB, BLOB, BFILE

У версії Oracle 8.0 з'явилися типи даних, що вони дозволяють гнучко працювати з LOB-даними. Типи даних CLOB, NCLOB та BLOB називають внутрішніми LOB, оскільки така інформація зберігається безпосередньо в базі даних. Тип даних BFILE називається зовнішнім LOB, тому що цей тип зберігає дані в зовнішньому файлі.

Максимальний розмір усіх цих типів даних обмежено 4 Гбайт, але усі вони призначені для різних цілей.

CLOB – зберігає символьні дані в наборі символів бази даних. Інформація NCLOB відповідає національному наборові символів, визначеному для бази даних.

У Oracle8i й вище реальні дані й об'єктів CLOB, і об'єктів NCLOB зберігаються у двобайтових символах фіксованого розміру UNICODE, якщо набори символів бази даних або NCHAR мають змінний розмір. Згодом дані в міру потреби неявно перетворюються на набір символів, запитуваний клієнтом, навіть якщо цей набір є змінного розміру.

Тип даних BLOB використовується для зберігання неструктурованих двійкових даних. Тип даних BLOB є подібний до типу LONG RAW.

Тип даних BFILE дозволяє працювати в режимі «лише для читання» з великими двійковими файлами, розташованими поза базою даних. У полі типу BFILE зберігається покажчик на файл, який не обслуговується Oracle. Для змінювання цього файла використовуються зовнішні стосовно Oracle засоби.

2.3.3 Вставляння рядків до таблиці. Оператор INSERT

Оператор INSERT. Додає нові рядки до таблиці.

Умови. Для того, щоби ви могли вставляти рядки до таблиці, ця таблиця має існувати у вашій схемі або ви повинні мати привілей INSERT для цієї таблиці. Системний привілей INSERT ANY TABLE також дозволить вам уставляти рядки до будь-якої таблиці.

Синтаксис оператора INSERT має такий вигляд:

INSERT INTO [схема,] (таблиця | огляд) [@dblink]
[(стовпець [, стовпець] ...)]
(VFLUES (expr [, expr] …) | підзапит)

Ключові слова й параметри

Схема. Схема, що містить таблицю. Якщо схему опущено, то Oracle вважає, що таблиця перебуває в Вашій схемі.

Таблиця. Ім'я таблиці, до якої вставляються рядки.

Dblink. Повне чи часткове ім'я зв'язку до вилученої бази даних, в якій розташовано таблицю.

Стовпець. Стовпець таблиці. В рядку, що вставляється, кожен стовпець з цього списку набуває значення з фрази VALUES. Якщо ви опускаєте який-небудь стовпець таблиці з цього списку, то в рядку, що вставляється, цей стовпець набуде свого значення за замовчуванням, що було специфіковано при створюванні таблиці.

VALUES. Задає рядок значень, що його має бути вставлено до таблиці чи огляд.

Підзапит. У пропозиціях SELECT повертає рядки, що вони вставляються до таблиці. Список вибірки цього підзапиту повинен мати таку саму кількість стовпців, що й список стовпців пропозиції INSERT.

Використовування оператора INSERT

Пропозиція INSERT із фразою VALUES додає до таблиці один рядок. Цей рядок містить значення, визначуване фразою VALUES.

Пропозиція INSERT, що містить замість фрази VALUES підзапит, додає до таблиці всі рядки, які повертаються цим підзапитом. Oracle опрацьовує підзапит і вставляє до таблиці кожен рядок, що він повертає. Якщо підзапит не повертає жодного рядка, то Oracle не вставляє рядків до таблиці. Підзапит може звертатися до будь-якої таблиці, оглядової чи знімкової, у тому числі до цільової таблиці пропозиції INSERT.

2.3.4 Поняття обмеження. Обмеження NOT NULL, UNIQUE, CHECK

Обмеження (constraint) – це одна й більш умов, яким має задовольняти введений користувачем запис, щоби Oracle вставив її до таблиці. Обмеження зберігаються як частина визначення таблиці й після створювання застосовуються автоматично. Коли введена кимось команда INSERT чи UPDATE порушує обмеження, Oracle перериває її виконання, робить відкочування і видає повідомлення про помилку.

При створюванні обмеження цілісності йому привласнюється ім'я. Oracle зберігає це ім'я в словнику даних разом з визначенням обмеження.

Створіть демонстраційні таблиці EMP та DEMP, заповніть їх даними, використовуючи файл DEMOBLD.SQL. Виконайте команду визначальну структуру цих таблиць. Вона називається DESCRIBE

Виконайте команди: DESCRIBE emp та DESCRIBE dept.

Обмеження NOT NULL

Обмеження NOT NULL зазначає, що стовпець таблиці не може містити порожніх значень. Щоб задовольнити цьому обмеженню, кожен рядок у таблиці повинен містити значення для даного стовпця.

Ключове слово NULL свідчить, що стовпець таблиці може містити порожні значення. Воно саме по собі не визначає обмеження цілісності.

Команда зміни статусу існуючого стовпця на NOT NULL має такий синтаксис:

ALTER TABLE ім'я_таблиці MODIFY (ім'я_стовпця NOT NULL);
Застосуємо цю команду до таблиці DEPT.
SQL> ALTER TABLE dept MODIFY (deptno NOT NULL);
Table altered.

Виконайте команду: DESCRIBE dept.

Внаслідок виконання команди DESCRIBE dept ви побачите, що стовпцеві deptno таблиці DEPT привласнене обмеження NOT NULL.

Обмеження UNIQUE

Обмеження UNIQUE позначає стовпець чи комбінацію стовпців як унікальний ключ. Щоб задовольнити обмеженню UNIQUE, жодні два рядки в таблиці не можуть мати одне й те саме значення для унікального ключа. Однак унікальний ключ, який складається з єдиного стовпця, може мати порожні значення.

Синтаксис команди, використовуваної для створювання обмеження, яке гарантує унікальність значення, має такий вигляд:

ALTER TABLE ім'я_таблиці
ADD CONSTRAINT ім'я_обмеження UNIQUE
(імена_стовпців);

Подана нижче пропозиція, змінює таблицю DEPT, визначає й додає унікальний ключ за стовпцем DNAME:

SQL> ALTER TABLE dept
2 ADD CONSTRAINT unq_dname UNIQUE
3 (dname);

Table altered.

Обмеження unq_dname ідентифікує стовпець DNAME як унікальний ключ. Це обмеження гарантує, що жодні два відділи в таблиці не мають однакових імен.

Обмеження CHECK

Контрольне обмеження (check constraint) дозволяє визначити, яким умовам повинні задовольняти вхідні дані, щоб Oracle дозволив їхнє введення. Контрольне обмеження можна визначати для кожного стовпця таблиці.

Команда створювання контрольного обмеження на стовпець існуючої таблиці має такий синтаксис:

ALTER TABLE ім'я_таблиці ADD
CONSTRAINT ім'я_обмеження
CHECK (ім'я_стовпця умова);

Змінює таблицю DEPT та визначає контрольне обмеження CHECK за стовпцем DEPTNO пропозиція:

```
SQL> ALTER TABLE dept
  2  ADD CONSTRAINT check_deptno
  3  CHECK (deptno BETWEEN 10 AND 99);

Table altered.
```

Контрольне обмеження CHECK_DEPTNO гарантує, що жодний номер відділу не буде менш за 10 або більш за 99.

Створювання обмеження первинного ключа

Команда, котра дозволяє визначити, які стовпці існуючої таблиці будуть використовуватися як первинний ключ, має такий синтаксис:

```
ALTER TABLE ім'я_таблиці
ADD PRIMARY KEY (ім'я_стовпця1, ім'я_стовпця2, ...);
```

Створимо первинний ключ для стовпця DEPTNO головної таблиці DEPT, змінюючи її в такий спосіб:

```
SQL> ALTER TABLE dept
 2 ADD PRIMARY KEY (deptno);
Table altered.
```

Створимо первинний ключ для стовпця EMPNO підлеглої таблиці EMP, що, він також змінить її:

```
SQL> ALTER TABLE emp
  2  ADD PRIMARY KEY (empno);

Table altered.
```

Створювання обмеження зовнішнього ключа

Первинні й зовнішні ключі – це фізичні компоненти, без яких неможливо встановити зв'язок поміж таблицями. Однак самі по собі вони не забезпечують реляційної цілісності. Навіть якщо стовпці первинного і зовнішнього ключів мають однакові типи даних, Oracle не вважає їх пов'язаними, **допоки ви не оголосите про це.**

Останній крок має вирішальне значення: ви повинні визначити обмеження на підпорядковану таблицю, що її буде використовувано для звірювання значень, котрі вводяться до стовпця зовнішнього ключа, зі значенням первинного ключа головної таблиці. Без такого обмеження

користувач зможе ввести до зовнішнього ключа підпорядкованої таблиці значення, яких не існує у головній таблиці.

Команда створювання обмеження зовнішнього ключа має такий синтаксис:

```
ALTER TABLE ім'я_підпорядкованої_таблиці
ADD CONSTRAINT ім'я_обмеження
FOREIGN KEY (імена стовпців_підпорядкованої_таблиці)
REFERENCES ім'я_головної_таблиці;
```

Створимо обмеження зовнішнього ключа для підпорядкованої таблиці EMP, використовуване для встановлення зв'язку поміж таблицями DEPT та EMP.

```
SQL> ALTER TABLE emp
  2  ADD CONSTRAINT fk_deptno
  3  FOREIGN KEY (deptno)
  4  REFERENCES dept (deptno);

Table altered.
```

Приклади створювання обмежень первинного та зовнішнього ключів, при створюванні таблиць

Обмеження первинного та зовнішнього ключів можна створити, не змодифіковуючи таблицю, а при її створюванні.

Для цього вилучимо демонстраційні таблиці DEPT та EMP, використовуючи команди: DROP TABLE emp; та DROP TABLE dept;

Змодифікуємо коди створювання демонстраційних таблиць DEPT та EMP.

Змодифікуємо код головної таблиці DEPT при її створюванні включенням обмеження первинного ключа до стовпця DEPTNO:

```
SQL> CREATE TABLE DEPT
        2       (DEPTNO NUMBER(2) PRIMARY KEY,
        3       DNAME VARCHAR2(14),
        4       LOC VARCHAR2(13) );

        Table created.
```

Заповнимо таблицю DEPT даними з файла DEMOBLD.SQL. Отже, створено головну таблицю DEPT, і до неї при створюванні впроваджено обмеження первинного ключа PRIMARY KEY.

Змодифікуємо код підпорядкованої таблиці EMP при її створюванні впровадженням обмеження первинного ключа (PRIMARY KEY) до стовпця EMPNO і обмеження зовнішнього ключа – до стовпця DEPTNO виразом:

```
    CONSTRAINT fk_deptno
    FOREIGN KEY (deptno) REFERENCES dept (DEPTNO)

SQL> CREATE TABLE EMP
2       (EMPNO NUMBER(4) PRIMARY KEY,
3       ENAME VARCHAR2(10),
4       JOB VARCHAR2(9),
5       MGR NUMBER(4),
6       HIREDATE DATE,
7       SAL NUMBER(7, 2),
8       COMM NUMBER(7, 2),
9       DEPTNO NUMBER(2),
10      CONSTRAINT fk_deptno
11      FOREIGN KEY (deptno) REFERENCES dept (DEPTNO));

    Table created.
```

Заповнимо таблицю EMP даними з файла DEMOBLD.SQL. Отже, створено підпорядковану таблицю EMP, й до неї при створюванні впроваджено обмеження первинного ключа PRIMARY KEY за стовпцем EMPNO, а також обмеження зовнішнього ключа FOREIGN KEY за стовпцем DEPTNO.

2.3.6 Керування даними таблиці

Створювання таблиці на підставі існуючої

Для швидкого копіювання даних з однієї створеної і заповненими даними таблиці до іншої, котрої ще не існує, слід використовувати оператор CREATE TABLE, щоби створити таблицю а потім скопіювати рядки, отримані при виконанні оператора SELECT. Для цього слід використовувати різновид команди CREATE TABLE з таким синтаксисом:

```
CREATE TABLE ім'я_нової таблиці AS
оператор SELECT;
```

Для створення таблиці з ім'ям emp_test3 використовуємо такий код:

```
SQL> CREATE TABLE emp_test3 AS
  2  SELECT * FROM emp;

Table created.
```

Таблиця emp_test3 за вмістом цілковито відповідатиме таблиці EMP. Для перевірки цього слід виконати таку команду:

```
SET LINESIZE 135
SET PAGESIZE 35
SQL>SELECT * FROM emp_test3;
```

Якщо треба скопіювати лише певні рядки та стовпці вихідної таблиці, можна змодифікувати оператор CREATE TABLE в такий спосіб, щоби стовпці й рядки, що їх має бути перенесено до нової таблиці, визначив оператор SELECT.

Створимо таблицю з ім'ям emp_test4, до якої вставимо рядки з іменами службовців, наприклад, на літеру М з таблиці emp_test3.

```
SQL> CREATE TABLE emp_test4 AS
  2  SELECT empno,ename
  3  FROM emp_test3
  4  WHERE ename LIKE 'M%';

Table created.
```

Перевіримо, чи задовольняє вміст створеної таблиці emp_test4 нашій умові. Для цього виконаємо команду

```
SQL> SELECT * FROM emp_test4;
```

Умову, створювання таблиці emp_test4 виконано.

Перейменування таблиць

Синтаксис команди перейменування таблиць є такий:

```
RENAME старе_ім'я_таблиці TO нове_ім'я_таблиці;
```

Застосуємо цю команду для перейменування, наприклад таблиці emp_test4, й надамо їй нове ім'я test4_emp:

```
SQL> RENAME emp_test4 TO test4_emp;

Table renamed.
```

2.3.7 Змінювання структури таблиці

За час існування бази даних можуть змінитися бізнес-вимоги, яким вона повинна задовольняти. Найчастіше це слугує за причину для змінення структури таблиць, що вже містяться в базі даних. До таких змін структури таблиць належать: вилучання й додавання стовпців, змінювання типу даних існуючих стовпців, змінення NULL-опцій стовпців.

Вилучання стовпців

Синтаксис команди вилучання стовпця таблиці є такий:

```
ALTER TABLE ім'я_таблиці DROP COLUMN ім'я_стовпця;
```

Застосуємо цю команду для вилучання стовпця ENAME таблиці test4_emp:

```
SQL> ALTER TABLE test4_emp DROP COLUMN ENAME;

Table altered.
```

Визначимо структуру таблиці test4_emp для перевірки внесених до неї змін.

Виконаємо команду DESCRIBE test4_emp:

```
SQL> DESCRIBE test4_emp
Name          Null?    Type
 ----------------------------
EMPNO                  NUMBER(4)
```

Зміни потвердилися. У таблиці test4_emp залишився один стовпець з ім'ям EMPNO. Стовпець з ім'ям ENAME вилучено з таблиці test4_emp.

Додавання стовпців до таблиці

Нові стовпці розміщаються наприкінці табличної структури після всіх існуючих стовпців. Використовуваний для цього синтаксис виглядає як:

```
ALTER TABLE ім'я_таблиці
ADD ім'я_нового_стовпця тип даних;
```

Застосуємо цю команду, додамо стовпець з ім'ям ENAME і типом даних VARCHAR2(10) до таблиці test4_emp:

```
SQL> ALTER TABLE test4_emp
 2  ADD ename VARCHAR2(10);

Table altered.
```

Визначимо структуру таблиці test4_emp для перевірки внесених до неї змін, додавання стовпця з ім'ям ENAME і типом даних VARCHAR2(10). Виконаємо команду DESCRIBE test4_emp:

```
SQL> DESCRIBE test4_emp
Name          Null?    Type
 ----------------------- -------------
 EMPNO                  NUMBER(4)
 ENAME                  VARCHAR2(10)
```

Зміни потвердилися. До таблиці test4_emp додано один стовпець з ім'ям ENAME і типом даних VARCHAR2(10).

Змінювання типу даних стовпця

Команда, що змінює тип даних існуючого стовпця, має такий синтаксис:

```
ALTER TABLE ім'я_таблиці
MODIFY ім'я_стовпця новий_тип_даних;
```

Змінимо тип даних стовпця з ім'ям ENAME і типом даних VARCHAR2(10) на новий тип даних CHAR(10) для таблиці test4_emp:

```
SQL> ALTER TABLE test4_emp
2  MODIFY ENAME CHAR(10);

Table altered.
```

Визначимо структуру таблиці test4_emp для перевірки внесених до неї змін типу даних стовпця з ім'ям ENAME з типом даних VARCHAR2(10) на новий тип даних CHAR(10). Виконаємо команду DESCRIBE test4_emp:

```
SQL> DESCRIBE test4_emp
 Name        Null?    Type
 ----------- -------- ----------
 EMPNO                NUMBER(4)
 ENAME                VARCHAR2(10)
```

Зміни підтвердилися. У таблиці test4_emp стовпець з ім'ям ENAME має тип даних CHAR(10).

Змінювання NULL-опцій стовпців

Для змінювання NULL-опцій стовпця, тобто впровадження обмеження NOT NULL для стовпця таблиці, використовується такий синтаксис:

```
ALTER TABLE ім'я_стовпця
MODIFY ім'я_стовпця NOT NULL;
```

Виконаємо команду DESCRIBE emp_test3 для визначення структури таблиці emp_test3. Використовуючи наведений вище синтаксис команди, впровадимо обмеження NOT NULL для стовпця SAL таблиці emp_test3:

```
SQL> ALTER TABLE emp_test3
  2  MODIFY sal NOT NULL;

Table altered.
```

Визначимо структуру таблиці emp_test3 для перевірки впровадження обмеження NOT NULL для стовпця SAL Виконаємо команду DESCRIBE emp_test3:

SQL> DESCRIBE emp_test3

Структура таблиці emp_test3 змінилася. За стовпцем sal впроваджено обмеження NOT NULL. Це обмеження гарантує, що жодний службовець у таблиці emp_test3 не має порожнього значення платні.

2.4 Фільтрація записів у таблицях

2.4.1 Типи команд SQL

Oracle SQL відповідає таким промисловим стандартам, як ANSI SQL92. Команди мови SQL поділяються на такі категорії:

- DDL (Data Definition Language – мова визначання даних);
- DML (Data Manipulation Language – мова маніпулювання даними);
- DQL (Data Query Language – мова запитів до даних);
- DCL (Data Control Language – мова керування даними);
- Команди адміністрування даних;
- Команди керування транзакціями.

Визначання структур бази даних (Data Definition Language – DDL)

Мова визначання даних (DDL) є частиною SQL, що надає користувачеві можливість створювати різні об'єкти бази даних і перевизначати їхню структуру, наприклад створювати чи вилучати таблиці.

Серед основних команд DDL, є такі команди:

CREATE TABLE (створювання таблиці)
ALTER TABLE (змінювання таблиці)
DROP TABLE (вилучання таблиці)
CREATE INDEX (створювання індексу)
ALTER INDEX (змінювання індексу)
DROP INDEX (вилучання індексу)

Маніпулювання даними (Data Manipulation Language – DML)

Мова маніпуляцій даними (DML) є частиною SQL, що надає користувачеві можливість маніпулювати даними усередині об'єктів реляційної бази даних.

Три основні команди DML:
INSERT (вставляння)
UPDATE (змінювання)
DELETE (вилучання)

Добирання даних (Data Query Language – DQL)

Цей розділ мови SQL подано однією командою. Для користувача реляційної бази даних мова запитів до даних (DQL) є найголовнішою частиною SQL. Цією командою є команда SELECT.

Команда SELECT, що має безліч опцій та необов'язкових параметрів, використовується для побудови запитів до реляційних баз даних. За її допомогою можна конструювати запити будь-якої складності – від найзагальніших до вельми спеціальних і від найпростіших до неймовірно складних. Запит – це вимога на одержання інформації з бази даних.

Мова керування даними (Data Control Language – DCL)

Команди керування даними в SQL дозволяють здійснювати контроль над можливістю доступу до даних усередині бази даних. Команди DCL зазвичай використовуються для створювання об'єктів, стосовних керування доступом користувачів до бази даних, а також для призначання користувачам відповідних рівнів привілеїв доступу. До команд керування даними належать:

ALTER PASSWORD (змінити пароль)
GRANT (надати привілей)
REVOKE (скасувати привілей)
CREATE SYNONYM (створити синонім)

Команди адміністрування даними

Команди адміністрування даними надають користувачеві можливість виконувати аудит і аналіз усередині бази даних. Ці команди можуть також допомогти при аналізі продуктивності системи даних у цілому.

До загальних команд адміністрування даних належать:

START AUDIT (старт аудит)
STOP AUDIT (стоп аудит)

Команди керування транзакціями:

- Команда COMMIT використовується для того, щоби зберігати транзакції.
- Команда ROLLBACK використовується для того, щоби скасовувати транзакції.
- Команда SAVEPOINT створює крапки усередині груп транзакцій, до яких надсилається команда ROLLBACK.
- Команда SET TRANSACTION дозволяє призначати транзакції ім'я.

2.4.2 Оператор SELECT

Найчастіше з SQL-операторів у більшості середовищ баз даних виконуються запити. Запит (QUERY) чи оператор SELECT – це оператор вибору інформації в таблицях бази даних. Концептуально таблиця в Oracle

нагадує табличний список інформації, як, наприклад, у програмі електронних таблиць (spreadsheet). У поданому нижче програмному блоці демонструється оператор select, використовуваний для витягу інформації з таблиці з ім'ям EMP.

```
SELECT * FROM emp;
чи
SELECT empno, ename, job, mgr, hiredate, sal, comm., deptno FROM emp;
```

У запиті зазначається, щоби Oracle має повернути всі дані всіх стовпців таблиці EMP. У відповідь Oracle видає вміст таблиці EMP.

Головними компонентами оператора вибору є: конструкція **select**, чи стовпців, і конструкція **from**, чи таблична. У конструкції select розташовано список стовпців чи виразів, які містять потрібні дані. У першому операторі використовувався шаблоновий символ "зірочка" (*), що вказує Oracle на необхідність вибору інформації з кожного стовпця таблиці. Конструкція **from** повідомляє Oracle, з якої таблиці бази даних треба вилучити інформацію. Нерідко, окрім імені таблиці, інформацію яку потрібно зчитати, користувач бази даних повинен зазначати схему чи власника, якому належить таблиця.

Оператор SELECT не використовується сам собою, а вимагає зазначення певних параметрів за допомогою ключових слів. Окрім обов'язкових, у цього оператора є кілька необов'язкових ключових слів, які розширюють його можливості. З оператором SELECT має використовуватися ключове слово FROM, яке для цього оператора є обов'язковим.

Оператор SELSECT складається з виразів, котрі будуються на підставі чотирьох ключових слів:

- SELECT
- FROM
- WHERE
- ORDER BY

Ключове слово SELECT

В операторі SELECT ключове слово SELECT використовується разом з ключовим словом FROM, для того щоби організовувати вилучання даних з бази даних у зручному для читання форматі. Частина запиту, задана ключовим словом SELECT, визначає джерело добирання даних.

Синтаксис простого оператора SELECT є такий:

```
SELECT [ * ALL | DISTINCT СТОВПЕЦЬ1, СТОВПЕЦЬ2 ]
FROM ТАБЛИЦЯ1 [ , ТАБЛИЦЯ2 ];
```

За ключовим словом SELECT у запиті слідує список стовпців, значення яких ви бажали би бачити внаслідок запиту. За ключовим словом FROM слідує список таблиць, з яких мають вилучатися дані. Зірочка (*) використовується для вказівки на те, що внаслідок запиту має бути подано значення всіх стовпців таблиці. Опція ALL використовується тоді, коли потрібно показати всі значення стовпця, включаючи й повторювані. Опція DISTINCT використовується для того, щоби виключити повторення. З цих опцій використовуваною за замовчуванням опцією є ALL, яку через це зазначати необов'язково. Зверніть увагу на те, що імена стовпців у списку, що слідує за ключовим словом SELECT, розділяються комами, точно так само, як імена таблиць, які слідують за ключовим словом FROM.

В операторах SELECT коми використовуються для поділу аргументів у списках. Це можуть бути, наприклад, списки з іменами стовпців у запитах, списки з іменами таблиць, списки значень, розміщених в таблиці, чи списки значень, котрі задають умови з ключовим словом WHERE.

Аргументи – це значення, передбачені в синтаксисі оператора чи команди SQL. Аргументи можуть бути як обов'язковими, так і необов'язковими.

Ключове слово FROM

Ключове слово FROM завжди використовується з оператором SELECT. Це ключове слово є обов'язковим елементом запиту. Метою що задається за допомогою FROM виразу, є повідомлення базі даних про те, з якої таблиці чи таблиць мають вилучатися дані. Вираз FROM може включити як в одну, так і кілька таблиць.

Синтаксис, надаваний за допомогою FROM виразу є такий:

```
FROM ТАБЛИЦЯ1 [ , ТАБЛИЦЯ2 ]
```

Ключове слово WHER – використовування умов для добирання даних

Умова – це частина запиту, вона містить інформацію, на підставі якої добираються дані. Умова може набирати значень TRUE (істина) або FALSE (неправда), що й використовується для добирання. Вираз WHERE (де? куди? звідки?) використовується в запитах для виключення з розглядання певних рядків, з тих, що за відсутності умов їх було б долучено до результатів запиту.

У виразі WHERE може міститися кілька умов. Якщо умов є небагато, вони пов'язуються операціями AND та OR.

Операція – це символ чи ключове слово SQL, яке використовується для пов'язування елементів в операторі SQL.

Синтаксис оператора SELECT, котрий використовує WHERE, є такий:

```
SELECT [ * ALL | DISTINCT СТОВПЕЦЬ1, СТОВПЕЦЬ2 ]
FROM ТАБЛИЦЯ1 [ , ТАБЛИЦЯ2 ];
WHERE [ УМОВА | ВИРАЗ1 ]
[ AND УМОВА2 | ВИРАЗ2 ];
```

Підрахунок записів у таблиці

За допомогою простого запиту до таблиці можна дістати інформацію про кількість записів, що вони містяться в таблиці. Підрахунок здійснюється за допомогою функції COUNT.

Синтаксис використовування функції COUNT в запитах є такий:

```
SELECT COUNT (*)
FROM ім'я ТАБЛИЦІ
```

Функція COUNT використовується з дужками, в яких зазначається стовпець, за котрим слід провадити підрахунок, або зірочкою, якщо треба підрахувати всі рядки таблиці.

Підрахуємо всі записи в таблиці DEPT:

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

Поняття схеми

Іноді розроблювачі й адміністратори баз даних Oracle звертаються до її об'єктів як до елементів певної схеми. Схема – це логічна сукупність об'єктів бази даних, наприклад таблиць, які належать певному власникові. Коли користувачеві, що він реєструється в базі даних Oracle, надається можливість створювати об'єкти, подібні до таблиць, ці об'єкти належатимуть саме йому. Ідентифікатор, використовуваний при реєстрації в базі даних для виконання demobld.sql, визначає схему, якій належатимуть всі створювані таблиці.

У наведених вище прикладах жодне з посилань на таблиці не випереджалося ім'ям користувача-власника. У такій ситуації Oracle припускає, що потрібною є таблиця з вашої власної схеми. Якщо такої таблиці немає, ім'я таблиці слід випередити відомостями про власника схеми й відокремити власника від імені таблиці крапкою.

2.4.3 Опрацьовування NULL – значень

Опрацьовування NULL – значень

Іноді результатом запиту певної інформації є дещо непевне, в базах даних називане NULL-значенням. У теорії множин – математичній підставі реляційних баз даних – NULL являє собою значення порожньої множини, тобто множини даних, які не містять жодних значень. NULL – це відсутність інформації. Інакше кажучи, NULL – це не символ, одержуваний внаслідок натискання клавіші прогалини! Якщо не зазначено зворотне, то будь-який стовпець таблиці має припускати введення NULL-значень. Приклад зчитування NULL-значення наведено у стовпці MGR у такому програмному блоці (зверніть увагу на NULL-елемент у MGR для EMPNO 7839):

```
SELECT empno, ename, mgr
FROM emp;
```

Однак не завжди вас влаштовує наявність "порожнеч" у висновку, тоді можна задати певне значення на місці **порожньої множини**. Oracle забезпечує таку можливість спеціальною функцією, називаною **nvl ( ).** Припустімо, що ви не хочете бачити порожні простори в інформації для менеджера. Замість цього ви хочете, щоби висновок запиту містив нуль на місці, де має бути роздруковане NULL-значення. Запит, поданий у наведеному нижче блоці коду, ілюструє, як можна здобути бажаного результату:

```
SELECT empno, ename, nvl(mgr,0)
FROM emp;
```

Якщо значення стовпця, зазначеного в nvl ( ), – не NULL, повертається задане значення; коли значення стовпця – NULL, повертається спеціальний рядок символів. **Функцію nvl ( )** можна застосовувати щодо стовпців усіх типів даних, але слід пам'ятати про те, що значення стовпця, зазначене для видачі замість NULL-значення, має мати той самий тип даних, що й стовпець. Базовий синтаксис команди, що він запускає функцію nvl ( ), є такий:

```
NVL(coluwn_name, value_if_null)
```

2.4.4 Оператор SELECT в операціях зі стовпцями

Заміна заголовків стовпців псевдонімами

Повертаючи дані, Oracle створює для кожного стовпця заголовки, щоби користувач знав, що саме являють собою дані. За замовчуванням Oracle друкує для стовпця ім'я, що збігається з тим, яке було зазначено в операторі SELECT, з урахуванням функцій (за їхньої наявності). На жаль, при використанні такого способу дані стовпця схарактеризовуються недостатньо добре, й, окрім того, Oracle зрізає вираз відповідно до довжини стовпця й до його типу даних, що повертаються. На щастя, ці труднощі можна подолати завдяки тому, що для стовпців в операторі SELECT використовуються псевдоніми. Будь-якому стовпцеві при видаванні оператора SELECT можна привласнити інше, більш інформативне ім'я, не порушуючи правил щодо довжини, встановлених в описі типу даних стовпця:

```
SELECT empno, ename, nvl(mgr,0) as mgr
FROM emp;
```

Псевдоніми стовпців є корисні для додавання до вихідних даних SQL-запитів заголовків, що несуть певне значеннєве навантаження. Псевдоніми

зазначаються у два способи: після опису стовпця через прогалину або за допомогою ключового слова **as**, яке відокремлює псевдонім більш чітко.

Об'єднання стовпців за допомогою операції конкатенації

Вихідні дані запиту можна змінювати не лише шляхом змінювання заголовка стовпця в операторі SELECT і використовування операції nvl ( ). Можна об'єднати всі стовпці, здобувши більш цікавого чи зручного для читання результату. Метод, застосовуваний для злиття вихідних даних двох стовпців у єдине ціле, називається конкатенацією (concatenation) і позначається символами ||. Окрім того, існує операція concat ( ), в якій зазначаються імена двох стовпців. У нижчеподаному прикладі стовпець ENAME поєднується (конкатенується) зі стовпцем JOB, для чого використовуються обидва методи:

```
SELECT ename || ', who is the ' ||
        concat(job,' for the company')
       as "Name and Role"
       FROM emp;
```

При використанні операції конкатенації разом з текстовим рядком, укладеної в одиночні лапки, вихідні дані двох чи більш стовпців стають єдиним цілим, набуваючи нового змісту. Рекомендуємо для більш чіткого трактування імені поєднуваних стовпців користуватися псевдонімами.

2.4.5 Фільтрування записів у таблицях з використовуванням конструкції ORDER BY та ключового слова WHERE

Сортування даних, що повертаються, за допомогою конструкції ORDER BY

Oracle не завжди повертає запитані дані в певному порядку, за якимось стовпцем, в числовому чи літерному порядкові. Відповідно до теорії реляційних баз даних, таблиця за означенням – це неупорядкований набір даних. Це добре в теорії, але не завжди добре для ситуацій з реального життя. Oracle дозволяє упорядковувати вихідні дані операторів **select** за допомогою конструкції, котра вводиться до них, **order** by (упорядкувати за).

Упорядковування, що задається за допомогою ключового слова ORDER BY, за замовчуванням буде упорядковуванням за зростанням, позначається як A-Z (А-Я) у разі сортування імен. Абеткове упорядковування за спаданням відповідає порядкові Z-A (Я-А). Для числових значень поміж 1 та 9 упорядковування за зростанням позначається 1-9, а за спаданням – 9-1.

Синтаксис оператора SELECT, що використовує вираз ORDER BY, є таким:

```
SELECT [ * ALL | DISTINCT СТОВПЕЦЬ1, СТОВПЕЦЬ2 ]
FROM ТАБЛИЦЯ1 [ , ТАБЛИЦЯ2 ];
```

WHERE [УМОВА | ВИРАЗ1]
[AND УМОВА2 | ВИРАЗ2];
ORDER BY СТОВПЕЦЬ1 | ЦІЛЕ ЗНАЧЕННЯ [ASC | DESC];

Зверніть увагу на використовування опції ASC. У виразі, що задається ключовим словом ORDER BY, ця опція може зазначатися після імені кожного зі стовпців.

Загальний синтаксис конструкції order by припускає використовування безпосередньо ключових слів, а також стовпців чи псевдонімів стовпців, за якими Oracle буде впорядковувати результати. Окрім того, можна задати спеціальну команду, яка визначатиме напрямок упорядковування: **asc (ascending)** – у порядку зростання або **desc (descending**) – у порядку спадання. Значенням за замовчуванням є asc, а нижче наведено результати для desc:

SELECT * from emp
ORDER BY ename desc;

Order by може визначити порядок сортування (зростаючий чи спадний) для одного чи кількох стовпців.

2.5 Сортування записів у таблицях

2.5.1 Синтаксис оператора SELECT

Оператор SELECT виконує запит з однієї чи декількох таблиць, повертаючи рядки та стовпці даних.

Умови. Для того щоб ви могли обрати дані з таблиці, ця таблиця має міститися у вашій схемі або ви повинні мати привілей SELECT щодо цієї таблиці.

SELECT [ALL| DISTINCT]
[ім'я_стовпця | перелік_стовпців]
FROM [ім'я_таблиці | імені-подання]
[, (ім'я_таблиці | імені-поданння)]...
[WHERE умови]
[HAVING умови]
[ORDER BY (ім'я_стовпця | перелік_стовпців) [ASC | DESC]...]
[, (ім'я_таблиці | перелік_стовпців) [ASC | DESC]…]
[GROUPBYім'я_стовпця [, ім'я_стовпця]…]

Ключові слова й параметри

ALL повертає всі обрані рядки, включаючи всі повторювані.

DISTINCT повертає лише одну копію повторюваних рядків. Повторювані рядки – це рядки, в яких є значення, однакові для кожного виразу в select-списку. Опція DISTINCT слугує для придушення висновку повторюваних рядків.

FROM. FROM завжди використовується разом з оператором SELECT. Це ключове слово є елементом запиту. Метою задаваного за допомогою FROM виразу є повідомлення базі даних про те, з якої таблиці чи таблиць вилучаються дані. Вираження FROM може включати, як одну так і кілька таблиць.

WHERE – обмежує безліч обираних рядків умовою. Умова – це частина запиту, котра містить інформацію, на підставі якої добираються дані. Умова може набирати або значення TRUE – істина, або значення FALSE (неправда), що й використовується для добирання. У виразі WHERE може міститися кілька умов. Якщо умов небагато, вони пов'язуються операціями AND та OR.

GROUP BY. Ключове слово GROUP BY використовується в операторі SELECT для того, щоби об'єднати повторювані значення до груп. Ключове слово GROUP BY має випливати за виразом WHERE й **передувати** ключовому слову ORDER BY.

HAVING. Ключове слово HAVING використовується в операторі SELECT разом із ключовим словом GROUP BY**,** щоби зазначити, які з груп має бути подано у висновку. Умова WHERE задає умови для значень з обраних стовпців, а HAVING – задає умови для груп, створюваних за допомогою GROUP BY**.**

Ключове слово HAVING в операторі SELECT має слідувати за виразом ключового слова GROUP BY й так само передувати ключовому слову ORDER BY, якщо останнє використовується.

ORDER BY. Сортування (упорядковування), котре задається за допомогою ключового слова ORDER BY, за замовчуванням буде упорядкованим за зростанням Вираз ASC задає сортування за зростанням, а вираз DESC – за спаданням.

2.5.2 Використання виразів

При розроблянні прикладних програм часто доводиться обирати дані, здобуті на підставі обчислень чи інших операцій, а не конкретних значень стовпця. Наприклад, для обчислення щотижневої платні всі бухгалтерії, що служить, за умови його підвищення на 05 %, можна скористатися таким виразом:

```
SELECT ename, (sal*1.05)/52 res
FROM emp
WHERE deptno = 10;
```

У наведеному виразі використовуються числові дані та стандартні символи арифметичних та логічних операцій: + (додавання). – (віднімання), *

(множення) та / (ділення). У виразах для обчислювання дат використовуються оператори додавання й віднімання для визначання дат, які відстоять від початкової дати на певну кількість днів. Окрім того, можлива конкатенація (з'єднання, склеювання) символьних полів за допомогою оператора конкатенації (||).

2.5.3 Використання псевдонімів для імен стовпців та таблиць

Використання псевдонімів для імен стовпців та таблиць означає перейменування таблиці та стовпця в межах використовуваного оператора SQL. При цьому справжнє ім'я таблиці та стовпця в базі даних не змінюється. Існує два варіанти зазначення псевдоніма (aliases). Можна відразу після імені стовпця чи таблиці, через прогалину задати ім'я псевдоніма або скористатися для стовпців ключовим словом AS.

У першому випадку призначення псевдоніма задається через прогалину, після імені стовпця. Внаслідок виконання цього запиту виводиться інформація про всі прізвища службовців та номери відділів, що до них належать в таблиці EMP; при цьому дані виводяться в довільному порядку:

```
SELECT e.ename family, e.deptno department
FROM emp e;
```

У другому прикладі призначення псевдоніма для стовпця здійснюється з використанням ключового слова AS, при цьому його укладено у подвійні лапки:

```
SELECT e.ename AS "family",
e.deptno AS "department"
FROM emp e;
```

У наведених прикладах виведений оператором SELECT результат запиту цілковито збігається.

2.5.4 Вибір унікальних значень

При роботі з базою даних доволі часто виникає необхідність як наслідок вибирання даних одержати лише унікальні значення даних їхніх таблиць.

Вибір унікальних значень є схожий на вибір одиночного списку значень; для цього слід зазначити в команді SELECT модифікатор DISTINCT.

Наприклад, для одержання списку всіх займаних службовцями посад з таблиці EMP:

```
SELECT DISTINCT job
FROM emp;
```

До аналогічного результату призведе застосовування модифікатора UNIQUE замість DISTINCT:

```
SELECT UNIQUE job
FROM emp;
```

Модифікатори DISTINCT та UNIQUE надають однаковий ефект для вибірки унікальних значень даних з таблиць.

2.5.5 Використання оператора SELECT при з'єднанні в запиті кількох таблиць

З'єднання

З'єднання – це пропозиція SELECT, яка комбінує рядки з двох чи більш таблиць. Кожен рядок, який повертається, містить дані з понад однієї таблиці. Oracle виконує з'єднання щоразу, коли у фразі FROM пропозиції WHERE з'являється кілька таблиць. WHERE визначає, як Oracle комбінує рядки цих таблиць.

Синтаксис для оператора SELECT, який вилучає дані з двох таблиць, має вигляд:

```
SELECT ім'я_таблиці_1.ім'я_стовпця.
    ім'я_таблиці_2.ім'я_стовпця.
FROM ім'я_таблиці_1, ім'я_таблиці_2
WHERE ім'я_підпорядкованої _таблиці.зовнішній ключ =
ім'я_головної _таблиці.первинний ключ;
```

Просте з'єднання, яке повертає прізвища службовців, номер та назву відділу, в якому працює кожен службовець, має вигляд:

```
SELECT ename, emp.deptno, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

Ви маєте використовувати з'єднання, щоби повернути ці дані, тому що потрібна інформація зберігається у двох різних таблицях – DEPT та EMP. Oracle комбінує рядки обох таблиць згідно з умовою WHERE:

```
emp.deptno = dept.deptno;
```

Зв'язок поміж таблицями EMP та DEPT у цьому прикладі визначається з використанням номера відділу. Це поле (DEPTNO) оголошено як первинний ключ для таблиці DEPT, а в таблиці EMP є зовнішнім ключем, тобто надсилається на поле первинного ключа таблиці DEPT. У розглянутому

випадку маємо зв'язок «головний – підпорядкований» поміж таблицями відділів та службовців. Дані при цьому пов'язано відношенням «один до багатьох»: одне значення з таблиці DEPT може багаторазово використовуватися в таблиці EMP.

Написання команди, яка комбінує дані з понад однієї таблиці в єдиному списку, називається створюванням з'єднання (JOIN). Найважливіше, що треба пам'ятати про з'єднання, полягає в такому: слід включити до оператора SELECT конструкцію WHERE, котра описує, як пов'язані один з одним первинний ключ головної таблиці та зовнішній ключ підпорядкованої таблиці. Якщо цього не зробити, то Oracle з'єднає кожен запис головної таблиці з кожним записом підпорядкованої таблиці. наслідком може стати надто довгий список.

2.5.6 Підзапити

Підзапит – це запит, котрий міститься у виразі ключового слова WHERE іншого запиту з метою додаткових обмежень на виведені дані. Підзапити також називають вкладеними запитами. Підзапит в запиті, який його містить, використовують для накладання умов на виведені дані. Підзапити можуть використовуватися з операторами SELECT, INSERT, UPDATE або DELETE.

У деяких випадках підзапит можна використовувати замість зв'язування таблиць, тим самим пов'язуючи дані таблиць неявно.

При використовуванні в запиті підзапиту спочатку виконується підзапит, а лише потім – запит, який його містить, причому з урахуванням умов виконання підзапиту. Результати виконання підзапиту використовуються при опрацьовуванні умов у виразі ключового слова WHERE основного запиту.

Підзапит можна використовувати або у виразі ключового слова WHERE, або у виразі HAVING головного запиту.

Логічні операції й операції порівняння типу: =, >, <, < >, IN, NOT IN, AND, OR тощо можна використовувати як у підзапиту, так і для опрацьовування результатів підзапиту у виразах ключових слів WHERE та HAVING.

При складанні підзапитів слід дотримувати таких правил:

- підзапит слід укладати в круглі дужки;
- підзапит може посилатися лише на один стовпець у виразі власного ключового слова SELECT, за винятком випадків, коли в головному запиті використовується порівняння з кількома стовпцями з під запиту;
- ключове слово ORDER BY використовувати в підзапиті не можна, хоча в головному запиті ORDER BY – може. Замість ORDER BY у підзапиті можна використовувати GROUP BY;
- підзапит, який повертає кілька рядків даних, можна використовувати лише в операторах, які припускають безліч значень, наприклад у IN;

- у списках ключового слова SELECT не припускаються посилання на значення типів BLOB, ARRAY, CLOB або NCLOB;
- підзапит не можна безпосередньо використовувати як аргумент, який припускає безліч значень функцій;
- операцію BETWEEN стосовно підзапиту використовувати не можна, але її можна використовувати в самому підзапиті.

Підзапити в операторі SELECT

Базовий синтаксис оператора SELECT з підзапитом виглядає як:

```
SELECT ім'я_стовпця [, ім'я_стовпця ]
FROM таблиця1 [, таблиця2 ]
WHERE ім'я_стовпця ОПЕРАЦІЯ
    (SELECT ім'я_стовпця [, ім'я_стовпця ]
    FROM таблиця1 [, таблиця2 ]
    [WHERE...] [GROUP BY...];
```

Щоби з'ясувати, хто працює в одному відділі зі службовцем на прізвище JONES, видайте такий під запит:

```
SELECT ename, deptno
FROM emp
WHERE deptno =
    (SELECT deptno
    FROM emp
    WHERE ename = 'JONES');
```

Підзапити в операторі INSERT

Підзапити можуть використовуватися і з операторами мови маніпулювання даними (DML – Data Manipulation Language). До трьох основних команд DML належать оператори: INSERTE, UPDATE, DELETE.

Оператор INSERT використовує дані, які повертаються підзапитом для поміщення їх до іншої таблиці. Обрані в підзапиті дані можна змодифікувати за допомогою символьних або числових функцій, а також функцій дат та часу.

Базовий синтаксис підзапиту в операторі INSERT має вигляд:

```
INSERT INTO ім'я_таблиці [ (стовпець1 [, стовпець2 ] ) ]
SELECT [ * | стовпець1 [, стовпець2 ] ]
    FROM таблиця1 [, таблиця2 ]
        [ WHERE значення ОПЕРАЦІЯ значення ];
```

Підзапити в операторі UPDATE

Підзапити можна використовувати в операторі UPDATE. За допомогою оператора UPDATE з підзапитом можна оновлювати дані як одного, так і кількох стовпців водночас.

Базовий синтаксис підзапиту в оператора UPDATE є:

```
UPDATE таблиця
SET ім'я_стовпця [, ім'я_стовпця ]
        (SELECT ім'я_стовпця [, ім'я_стовпця ]
        FROM таблиця
        [ WHERE ] );
```

2.6 Керування транзакціями. Журнал транзакцій

2.6.1 Поняття транзакції

Забезпечення паралельного доступу і збереження цілісності – дві надто тісно пов'язані між собою функції СКРБД. В основі механізму реалізації цих функцій у СКРБД лежить поняття транзакції (TRANSACTION). Коли користувач намагається переглянути певні дані або змінити них, стосовно бази даних він виконує транзакцію.

Одним із ключових компонентів СКРБД є адміністратор транзакцій (TRANSACTION MANAGER), єдиною функцією якого є забезпечення цілісності даних за наявності паралельного доступу до бази даних. У нього доволі багато роботи, оскільки, як правило, до бази даних, водночас звертається велика кількість користувачів, причому надто часто – до тих самих даних, а адміністратор транзакцій має повертати дані до СКРБД, начебто операції виконувались кожного моменту часу лише одним користувачем, забезпечуючи тим самим цілісність та коректність даних.

От тут і є відповідь на запитання, як СКРБД має взаємо пов'язувати безліч копій вихідного фрагмента даних, що їх змінено різними користувачами.

При опрацьовуванні транзакцій, які виникають впродовж певного часу, наступність даних може бути гарантовано лише у разі строгого дотримування дисципліни редагування–зберігання–відновляння фрагмента даних, яке дістало назву серіализації (SERIALIZATION), простіше кажучи, так, начебто це послідовні операції змінювання даних файла.

Транзакція являє собою елементарну неперервну операцію в механізмі паралельного доступу й функціонування СКРБД у цілому. Не може бути нічого більш дрібного в плані виконуваних операцій, аніж транзакция.

Транзакція є найпростішою неперервною операцією в перебігу функціонування СКРБД. Транзакція, яка дала очікуваний результат, називається здійсненою, прийнятою (committed), у противному разі – відкинутою, анульованою (rolled back).

СКРБД організує відновляння даних, використовуючи транзакції як логічні одиниці роботи.

Нормальне завершення операції, явно перерване користувачем вже запущеної операції, чи її несподіване переривання внаслідок апаратного збою

чи ще з якоїсь іншої причини, яка виходить за межі нормального функціонування системи, – усі ці події спонукують СКРБД звернутися до безлічі сформованих заздалегідь копій фрагментів даних і або оновити їх, якщо транзакція вчинилася, або відновити дані, анулювавши всі незавершені зміни.

Для того, щоб таке анулювання було можливе, а також для повторення операції змінювання даних, перерваної на півдорозі через екстраординарні обставини, СКРБД веде журнал транзакцій (transaction log). Фактично операція анулювання (rolling back) – це відома багатьом операція відкочування (undo), а відновлена процедура після переривання (rolling forward) – операція повторення (redo). До останнього доводиться звертатися в тому разі, коли прийнята транзакція не завершилася перезаписом оновлених даних з пам'яті на диск внаслідок апаратного збою. За таких умов СКРБД повторює всю транзакцію.

З викладеного випливає, що саме атомарний, неподільний характер транзакції є ключовим моментом її можливого анулювання чи відновлення за потреби.

2.6.2 Опрацьовування транзакцій

Опрацьовування транзакцій є підгрунтям будь-якої СКРБД. Фактично без опрацьовування транзакцій не було б сенсу застосовувати СКРБД. Відповідно до стандарту ANSI/ISO мови SQL, транзакція – це один чи кілька операторів SQL, викликуваних на виконання одним користувачем й або виконуваних користувачем успішно, або цілковито видкидуваних. Транзакція розпочинається з виконання першого оператора SQL і завершується явною чи неявною фіксацією або відкочуванням. У багатьох випадках для фіксації чи відкочування безпосередньо застосовується оператор COMMIT або ROLBACK, а іноді додаток, приміром SQL*Plus виконує фіксацію від імені користувача автоматично.

Важливо пам'ятати, що транзакція – це одна логічна одиниця роботи. Транзакція повинна бути в цілому виконана успішно або цілковито завершитися невдачею. Приміром, банківська транзакція з переведення грошей з одного рахунку на іншій може складатися з операторів SQL, які знімають гроші з одного рахунку, а потім вносять гроші до іншого рахунку. Якщо не буде виконано лише першу частину цієї транзакції, гроші можуть бути цілковито загублені.

Типова транзакція є позв'язана з виконанням таких етапів:

А Долучання додатка (необов'язкове)

1 Додаток опрацьовує дані, впроваджені користувачем, і створює з'єднання із сервером через NET8.

2 Сервер приймає запит на з'єднання і створює серверний процес від імені користувача чи передає запит спільно використовуваному серверному процесові через планувальник.

В Опрацьовування транзакцій.

1 Користувач виконує оператор чи оператори SQL і фіксує транзакцію. Наприклад, користувач змінює значення рядка в таблиці.

2 Серверний процес приймає цей оператор SQL і перевіряє спільно використовуваний пул на наявність у спільно використовуваній області SQL оператора SQL, ідентичного до даного. Знайшовши ідентичний оператор SQL, серверний процес перевіряє наявність у користувача привілеїв доступу до даних і використовує оператор, який перебуває в спільно використовуваній області SQL, для опрацьовування запиту. За відсутності такого оператора серверний процес розподіляє в спільно використовуваній області SQL місце для нового оператора, інтерпретує оператор і в остаточному підсумку його виконує.

3 Серверний процес вибирає дані з SGA (якщо вони там є) або вибирає їх з файла даних у SGA.

4 Серверний процес змінює дані в SGA. Пам'ятайте, що серверні процеси можуть лише зчитувати з файлів даних.

5 Процес LGWR записує до файла інформацію відновлення. Оператор вважається за зафіксований лише після запису інформації відновлення до журналу. Через якийсь час процес DBWR записує змінені блоки до місця постійного зберігання.

6 Якщо транзакція пройшла успішно, – клієнтському процесові мережею повертається код завершення. Якщо відбувається відмовлення, – повертається повідомлення про помилку.

7 Користувач повертається до початку етапу В – «Опрацьовування транзакцій» і видає подальші транзакції, допоки не завершить роботу й не вийде з додатка.

Етап опрацьовування транзакцій у додатку повторюється доти, допоки користувач не завершить роботу з конкретним додатком і не вийде їхнього додатка.

С Завершення роботи додатка

1 Додаток виходить з СКРБД. Ця подія слугує для бази даних Oracle сигналом про те, що можна звільнити всі пов'язані з ним ресурси.

2 Процес PMON Oracle простежує завершення серверного процесу.

3 Звільняються всі ресурси, відведені додаткові, у тому числі й ресурси пам'яті.

Процес вилучання додатка відбувається лише у разі, якщо додаток чи користувач припиняє сеанс роботи із сервером. Додаток не повинен виконувати підмикання при опрацьовуванні кожного оператора.

В перебігу опрацьовування транзакцій фонові процеси Oracle забезпечують безперебійну роботу системи. Пам'ятайте, що при обслуговуванні вашого додатка можуть продовжувати роботу сотні інших користувачів.

Призначення СКРБД Oracle полягає в підтримуванні системи в працездатному стані, в керуванні конкуренцією та блокуваннями і забезпеченні

необхідної продуктивності. Навіть якщо додаток змінив певні дані в базі даних, ці дані можуть залишатися не записаними до файлів даних. Це може статися пізніше, коли процес DBWR запише ці зміни для постійного зберігання.

2.6.3 Журнал транзакцій

Дані про транзакції зберігаються в буфері доти, допоки їх не буде переписано до файла оперативного журналу транзакцій. Це типовий циклічний буфер – він заповнюється від початку до кінця, а потім нова інформація знову записується на початку буфера. Після заповнення вміст буфера переноситься до файла журналу транзакцій.

Розмір буфера журналу транзакцій задається розміром LOG_BUFFER. Значення параметра зазначає розмір буфера в байтах. Якщо це значення не є велике, то процес LGWR вельми часто переписуватиме інформацію з буфера до файла й тим самим сповільнюватиме роботу системи (однак це виявлятиметься лише в разі потужної інтенсивності звертання до бази даних). Сам процес запису до журналу можна оперативно контролювати за допомогою подання V$SYSSTAT. Запросіть його для поля value з полем name, дорівнюваним «redo log space request». Внаслідок цього ви довідаєтеся, як довго користувацькі процеси перебували в стані очікування при звертанні до буфера журналу транзакцій.

Для того щоби гарантувати послідовний характер запису до буфера журналу, сервер Oracle керує доступом до нього за допомогою засувок (latch). Така засувка являє собою не що інше, як блокування процесом Oracle певної структури пам'яті – аналогічно до блокування файла чи рядка таблиці. Процес блокує сторонні звертання до пам'яті, виділеної для буфера журналу транзакцій. Отже, якщо один процес наклав засувку на буфер, інші не можуть звертатися до нього доти, поки засувку не буде знято.

Сервер Oracle обмежує кількість транзакцій, дані про які заносяться до журналу водночас, параметром LOG_SMALL_ENTRY_MAX_SIZE. Значення цього параметра задається в байтах. Значення, прийняте системою за замовчуванням, може змінюватися залежно від використовуваної операційної системи й апаратних засобів.

Для багатопроцесорних серверів припускається виділення більшого простору, аніж задано параметром LOG_SMALL_ENTRY_MAX_SIZE. При цьому, замість засувки на виділення пам'яті, процеси можуть використовувати засувки копій журналу транзакцій (redo copy latch). Кількість таких засувок задається параметром LOG_SIMULTANEOUS_COPY, значення якого за замовчуванням дорівнює кількості процесів у системі. Використовуючи засувки копій журналу транзакцій, кілька процесів можуть водночас записувати інформацію до буфера журналу транзакцій.

2.6.4 Керування транзакціями

Користувач бази даних Oracle має одразу усвідомити, що змінювання даних у базі Oracle не стає фактом, що здійснився, негайно ж. У Oracle користувачеві дозволяється виконувати кілька операторів змінювання даних як одну логічну одиницю роботи, завершувану або зберіганням результатів у базі даних, або їхнім скасуванням. Така логічна одиниця роботи називається транзакцією (transaction**).** Вона розпочинається з першого SQL-оператора користувача, що виконується, і закінчується явною фіксацією (завершенням) або відкочуванням з боку користувача.

Транзакція (transaction) – логічне об'єднання операторів DML до однієї одиниці виконуваної роботи.

Транзакції опрацьовуються групою керувальних засобів, за допомогою яких користувач може видати оператор **INSERT, UPDATE** або **DELETE**, який зазначає початок серії операторів змінювання даних. Після внесення змін до бази даних користувач може зберігати дані в базі, завершивши транзакцію явно. Якщо ж у будь-якого моменту транзакції виникає помилка, користувач може скасувати зміни в базі даних, повернувши її до стану, у якому вона перебувала до початку транзакції.

Транзакції створюються за допомогою двох різних елементів бази даних Oracle. Перший – це набір команд, які визначають початок, контрольні точки й завершення транзакції, а другий – спеціальний механізм блокування, який запобігає змінюванню інформації рядків бази даних декількома користувачами одночасно.

Транзакції описуються такими командами:

- **SET TRANSACTION**. Розпочинає транзакцію і встановлює її базові характеристики. Ця команда не є обов'язкова. Транзакція розпочинається автоматично при запуску SQL*Plus, а також при завершенні чи відкочуванні попередньої транзакції.
- **COMMIT.** Завершує поточну транзакцію зберіганням змін у базі даних і розпочинає нову транзакцію.
- **ROLLBACK.** Завершує поточну транзакцію скасуванням змін у базі даних і розпочинає нову транзакцію.
- **SAVEPOINT.** Установлює контрольні точки (точки переривання) для транзакції, дозволяючи неповне відкочування.

Засоби опрацьовування транзакцій керують змінюванням інформації в СКРБД Oracle.

До засобів керування транзакціями належать команди початку й завершення транзакції, встановлення контрольних точок, а також механізми блокувань, які перешкоджають внесенню змін до бази даних кільком користувачам водночас.

2.6.5 Команди керування транзакціями

Команда SET TRANSACTION

Ця команда застосовується для початку транзакції. Якщо зміни в базу даних вносяться після видавання команди set transaction, але до завершення транзакції, усі ці зміни вважаються за частину транзакції. Оператор set transaction не є обов'язковий, тому що транзакція розпочинається:

- після реєстрування користувача в Oracle за допомогою SQL*Plus та виконання ним першої команди;
- одразу ж після видавання оператора rollback або commit, що завершує транзакцію;
- після виходу користувача з системи;
- внаслідок аварії системи;
- після видавання команди мови опису даними, наприклад alter database.

Команда SET TRANSACTION дозволяє змінювати властивості лише поточної транзакції. У ній можуть застосовуватися три опції, і вона повинна бути першою командою в транзакції. Якщо в поточній транзакции вже було видано команди, слід видати COMMIT або ROLLBACK, перш ніж буде прийнято команду SET TRANSACTION.

Команда SET TRANSACTION має такі параметри:

- READ ONLY. Установлює поточну транзакцію лише для читання.
- READ WRITE. Значення за замовчуванням.
- USE ROLLBACK SEGMENT (rollback_segment_name). Явно призначає сегмент відкочування.

Опція READ ONLY

Опція READ ONLY належить до процесу лише для читання, коли не можна виконувати відновлення певних таблиць у базі даних. Вона також встановлює рівень несуперечливості читання на рівень транзакції, що дозволяє бачити всі дані у вигляді знімка даних на момент запуску транзакції. Ця опція є зручна для сеансів, які лише запитують дані, оскільки витрати опрацьовування для транзакції цього типу є менше, аніж для нормальної транзакції. Вона є застосовна лише для таких транзакцій, як звіти, де потрібен знімок поточних даних.

У транзакцій лише для читання набір команд обмежений п'ятьма групами:

- ALTER SESSION
- ALTER SYSTEM
- LOCK TABLE
- SELECT (крім SELECT FOR UPDATE)
- SET ROLE

Опція READ WRITE

Транзакція цього типу застосовується за замовчуванням. Користувач має можливість оновлювати або вилучати дані, а також запитувати таблиці, якщо він має відповідні привілеї бази даних.

Опція USE ROLLBACK SEGMENT

Опція USE ROLLBACK SEGMENT використовується для встановляння сегмента відкочування для транзакцій, що оновлюють значні обсяги даних і тому створюють сегменти відкочування більші за звичайні. Такий сегмент відкочування може бути створено з більш потужними екстентами INITIAL та NEXT для запобігання досягненню максимальної кількості екстентів у сегменті відкочування.

Команда COMMIT (Завершити)

Оператор commit в опрацьовуванні транзакцій представляє той момент часу, коли користувач уже вніс необхідні зміни як до однієї логічної групи, а оскільки помилок не відбулося, він готовий до зберігання результатів своєї роботи. Ключове слово work (робота) в синтаксисі commit не є обов'язкове і просто робить текст програм зрозуміліше. Видаванням команди commit у базі даних ще й неявно закривається поточна транзакція й розпочинається нова.

Важливо розуміти також, що неявна операція commit виконується в базі даних тоді, коли користувач виходить з SQL*Plus або видає команду мови опису (означення) даних (DDL – data definition language), наприклад оператор create table, використовуваний для створювання таблиці бази, або alter table – для її змінювання.

```
COMMIT;
COMMIT WORK;
```

Команда ROLLBACK (Відкочування)

Видавши непотрібний оператор змінювання даних, завжди можна відмовитися від внесених до бази даних змін за допомогою оператора rollback. Після цього в сеансі зв'язку з базою даних неявно розпочинається нова транзакція. Відкочування виконується не лише внаслідок видачі оператора rollback, але й неявно, коли виконання оператора з тієї чи іншої причини завершується невдало або коли користувач скасовує оператор командою CTRL-C.

Можливості команди ROLLBACK не обмежуються лише одним рівнем скасовування. У сполученні з командою SAVEPOINT вона дозволяє повернутися до кожної з попередньо встановлених точок. Команда ROLLBACK у сполученні з командою SAVEPOINT має такий синтаксис:

```
ROLLBACK TO ім'я_точки_зберігання
```

Команда SAVEPOINT (точка зберігання)

Іноді при виконанні тривалих транзакцій або транзакцій, які змінюють великий обсяг даних, небажано відкидати всі зміни лише тому, що останній оператор змодифікував не те, що треба. Точки зберігання дозволяють розбити весь обсяг роботи, виконуваної транзакцією, на окремі сегменти. Відкочування можна робити лише до конкретної точки зберігання, залишаючи попередні зміни в силі. Точки зберігання є вельми корисні в ситуаціях, коли треба відновити частину незавершеної транзакції. Команда SAVEPOINT має такий синтаксис:

SAVEPOINT ім'я_точки_зберігання

За наявності оператора rollback to savepoint ім'я точки_зберігання в нижчеподаному програмному блоці зберігаються лише ті зміни, що було внесено до опису точки зберігання:

```
SQL> SAVEPOINT SP1_EMP_TEST1;

Savepoint created.

SQL> DELETE FROM emp_test1
  2    WHERE ename = 'JONES'
  3    AND  JOB = 'MANAGER'
  4    AND SAL = '3975';
1 row deleted.

SQL> SAVEPOINT SP2_EMP_TEST1;
Savepoint created.

SQL> ROLLBACK TO SP1_EMP_TEST1;
Rollback complete.
```

Що дає використання кількох точок повернення? Насамперед гнучкість. При виконанні великого пакета команд можна установити точки зберігання після кожної логічної групи команд і, якщо наступна група команд з якоїсь причини дає незадовільний результат, ви завжди зможете повернутися до останньої точки і застосувати лише ті зміни, що їх було виконано до неї. Видаючи команду SAVEPOINT, ви встановлюєте іменований маркер, до якого можна буде повернутися за командою ROLLBACK. Точки зберігання залишаються активними до завершення змінювання даних, після чого вони зникають.

Змінені дані стають очевидні іншим користувачам та програмам, які працюють з базою даних Oracle лише після завершення змін. (Нерозуміння цього факту може легко призвести до непорозуміння. Наприклад дехто вставляє,

оновлює або вилучає дані, а потім дивується, чому змін не видно на комп'ютерах його колег. Зазвичай це означає, що користувач забув завершити свої зміни.)

2.6.6 Команда ROLLBACK TO SAVEPOINT

Команда повернення до точи відкочування (зберігання) має такий синтаксис:

ROLLBACK TO ім'я_точки_відкочування;

Фіксування (команда COMMIT) й відкочування (команда ROLLBACK) завершують транзакцію. Внаслідок фіксування всі зміни, внесені до даних, стають незмінними. Відкочування повертає всі зміни, внесені впродовж транзакції, відновлюючи попередній стан усіх змінених даних. Точка зберігання дозволяє використовувати команду ROLLBACK для виконання відкочування лише частини транзакції.

Точки зберігання призначено для використання як логічні пункти зупину в окремій транзакції. Вони дозволяють розбивати занадто довгі транзакції на менші частини й надалі надають точку відновлення. Точки зберігання в транзакції надають можливість виконувати відкочування транзакції до будь-якої заданої точки зберігання, допоки не виконано команду фіксування (команда COMMIT).

Команда COMMIT (фіксування), негайно фіксує всі дані, стирає всі точки зберігання й завершує транзакцію.

Нижче наведено варіант вищеподаної послідовності дій із застосовуванням точок зберігання. Цей приклад показує, в який спосіб точка зберігання впливає на транзакцію:

1 Підключення до програми SQL*Plus (розпочинається сеанс 1).

2 Створювання тестової таблиці співробітників з ім'ям EMP_TEST1 на підставі існуючої таблиці EMP (розпочинається і завершується транзакція 1).

3 Змінювання даних в тестовій таблиці EMP_TEST1. Підвищення заробітної платні службовця з ім'ям JONES (розпочинається транзакція 2).

4 Переглядання записів у тестовій таблиці EMP_TEST1 для перевірки внесення до неї змін (продовження транзакції 2).

5 Створювання точки зберігання, наприклад з ім'ям SP1_EMP_TEST1 (SAVEPOINT SP1_EMP_TEST1;) (продовження трансакції 2).

6 Відновлення тестової таблиці службовців EMP_TEST1. Вилучення запису (рядка) з таблиці EMP_TEST1 з інформацією про службовця з ім'ям JONES (продовження транзакції 2).

7 Переглядання записів у тестовій таблиці EMP_TEST1 для перевірки внесення до неї змін, пов'язаних з вилучанням даних службовця з ім'ям JONES (продовження транзакції 2).

8 Створювання точки зберігання, наприклад, з ім'ям SP2_EMP_TEST1 (SAVEPOINT SP2_EMP_TEST1;) (продовження транзакції 2).

9 Створювання команди повертання до точки зберігання SP1_EMP_TEST1. Команда ROLLBACK TO SP2_EMP_TEST1;(продовження транзакції 2).

10 Переглядання записів у тестовій таблиці EMP_TEST1 для перевірки внесення до неї змін (продовження транзакції 2).

11 Видання оператора фіксації (COMMIT) (завершення транзакції 2).

12 Вилучання тестової таблиці EMP_TEST1 (розпочинається й завершується транзакція 3).

13 Вихід з програми SQL*Plus (завершення сеансу 1).

Зверніть увагу, що точка зберігання надає можливість виконати відкочування до певного місця в поточній транзакції, не впливаючи на попередні зміни, виконані перед точкою зберігання. Точки зберігання дозволяють створити набір точок відновляння, котрі перебуватимуть у вашому розпорядженні до завершення даної транзакції.

Застосовування команди COMMIT (фіксування), призводить до негайного фіксування всіх даних, стирання всіх точок зберігання і завершення транзакції.

2.7 Уявлення, послідовності, синоніми

2.7.1 Уявлення

Ідея **уявлень (view)** є проста: визначити запит, який передбачається часто використовувати, та зберегти його в базі даних Oracle і дозволити користувачам звертатися до нього по імені, як до звичайної таблиці. Коли користувач вибирає дані з уявлення, Oracle виконує відповідний запит, зорганізовує результати в такий спосіб, як зазначено в уявленні, і видає їх користувачеві. Для користувача уявлення виглядає як таблиця, з якої надходять дані. Проте насправді дані надходять через уявлення, з одного чи декількох інших джерел.

Навіщо потрібні уявлення? З цілої низки причин. Зокрема уявлення широко застосовуються для поєднання даних з двох і більш таблиць та видавання їх користувачам у вигляді одного списку, який легко читати. Це спрощує процес вибірки записів до такої міри, коли користувачам не треба знати, в який спосіб поєднуються таблиці, робить дані доступними для переважної кількості користувачів.

За допомогою уявлень зручно підтримувати безпеку, оскільки це дозволяє обмежувати діапазон рядків та стовпців, що повертаються користувачам. Якщо ви не бажаєте, щоби користувачі бачили стовпець із зарплатнею з таблиці особистих даних, просто не включайте його до визначеного уявлення. Для користувачів уявлення, цей стовпець не існуватиме. Це стосується й для рядків: включіть до уявлення конструкцію WHERE – і записи, що повертаються, будуть відфільтровані в будь-який потрібний вам спосіб.

Створення уявлення

Для створення уявлення слід зазначити лише ім'я уявлення й оператора SELECT, який виконуватиметься при зверненні до уявлення. Відповідний синтаксис має вигляд:

```
CREATR OR REPLACE VIEW ім'я уявлення AS
Оператор SELECT
;
```

Зверніть увагу, що тут присутній новий елемент: OR REPLACE. Він дозволяє створювати нове уявлення навіть тоді, коли уявлення із зазначеним ім'ям вже існує (зрозуміло, що існуюче уявлення при цьому перезаписується).

Щоб побачити, як працює уявлення, треба ввести такі команди.

```
SELECT * FROM purchase;

CREATE OR REPLACE VIEN sales_by_atlas_v AS
SELECT *
FROM purchase
WHERE salesperson = 'CA'
;

SELECT * FROM sales_by_atlas_v;
```

Наступні команди показують, як створити уявлення для переглядання даних із сполучених таблиць:

```
CREATE OR REPLACE VIEW sales_per_person_v AS
SELECT pers.first_name || ' ' || pers.last_name SALESPERSON,
     purc.product_name,
     purc.purchase_date,
     purc.quantity
FROM person   pers,
     purchase purc
WHERE  pers.person_code = purc.salesperson (+)
;

SELECT * FROM sales_per_person_v
ORDER BY salesperson, product_name, purchase_date;
```

Слід звернути увагу, що в останньому прикладі конструкцію ORDER BY включено до оператора SELECT, який витягує дані з уявлення, а не саме уявлення. До появи Oracle 8i уявлення не могли містити конструкцію ORDER BY. У Oracle 8i і в подальших версіях можна примусити уявлення сортувати

записи, що відображаються, зазначивши відразу після конструкції WHERE конструкцію ORDER BY, як в стандартному операторі SELECT.

Скасування уявлень

Скасовувати уявлення так само легко, як і таблицю, але ця дія є менш руйнівна, оскільки уявлення не містить жодних даних; найгірше, до чого може призвести випадкове скасування уявлення – це до необхідності створювати його знову. Команда, яка скасовує уявлення, має синтаксис:

DROP VIEW ім'я_уявлення;

Для скасовування уявлення sales_per_person_v ця команда застосовується у вигляді:

DROP VIEV sales_per_person_v;

Змінювання визначення уявлення

Oracle не дозволяє змінювати існуюче уявлення. Єдиний спосіб змінити його – це знищити його і створити знову. З цієї причини всі команди створення уявлень слід зберігати у файлах сценаріїв. Вирішивши змінити уявлення, вам достатньо буде відкрити файл сценарію, замінити команду CREATE VIEW, що міститься в ньому, й запустити сценарій знову ще раз.

2.7.2 Послідовності

Бази даних призначено для дотримання порядку в масивах інформації. Одним зі способів впорядковування записів є привласнення їм послідовних номерів. Oracle дозволяє створювати лічильники, які мають назву **послідовність (sequences),** вміст яких збільшується кожного разу, коли до них здійснюється звертання. Посилаючись на послідовність при вставлянні записів, можна гарантувати, що кожному записові буде привласнено новий, унікальний номер.

Створювання послідовності

Для створювання послідовності використовується синтаксис:

CREATE SEQUENCE ім'я послідовності;

Ця проста команда створює послідовність, яка розпочинається з 1 і збільшується на 1 при кожному звертанні. Нічого іншого від послідовностей часто й не вимагається. Проте при визначенні послідовності можна використовувати багато додаткових параметрів. Погляньте на наведений нижче синтаксис, де перелічено найкорисніші з них:

```
CREATE SEQUENCE ім'я послідовності
[INCREMENT BY значення інкремента]
[START WITH початкове значення]
[MAXVALUE найбільше значення]
[MINVALUE найменше значення]
[CYCLE]
;
```

Параметр INCREMENT BY дозволяє створювати послідовності з інкрементів, які відрізняються від 1. Значення цього параметра може вміщувати до 28 цифр (хоча важко передбачити ситуації, де може знайти застосування такий інкремент). Якщо зазначити тут від'ємне число, то значення послідовності зменшуватиметься при кожному звертанні.

Параметр START WITH дозволяє створювати послідовність, початкове значення якої відрізнюється від 1. Це може стати в пригоді при створюванні послідовності для таблиці, яка вже має записи, щоб розпочати послідовність з числа, яка слідує за найбільшим існуючим ідентифікатором запису.

Параметри MAXVALUE та MINVALUE дозволяють обмежувати інтервал чисел, котрі генеруються послідовністю. Якщо використовувати їх в поєднанні з параметром CYCLE, задана безліч значень циклічно повторюватиметься.

Найбільш поширені є послідовності з інкрементом 1 і без обмежень на значення, що генеруються. Створіть таку послідовність за допомогою команди:

```
CREATE SEQUENCE test_seq;
```

Використовування послідовності

Щоб дістати значення послідовності, на неї слід посилатися як на таблицю. Послідовності містять два „псевдо стовпці" з іменами CURRVAL та NEXTVAL, які повертають поточне й наступне значення послідовності відповідно. Вибірка із стовпця NEXTVAL викликає автоматичний інкремент послідовності. Щоб побачити, як це відбувається, введіть команди:

```
SELECT test_seq.nextval FROM DUAL;
SELECT test_seq.nextval FROM DUAL;
SELECT test_seq.nextval FROM DUAL;
```

Тепер вам треба навчитися заповнювати стовпці таблиці з послідовності. Це робиться шляхом включення посилання на послідовність в оператора INSERT, як в наведених нижче командах.

```
CREATE TABLE test (
    record_id   NUMBER(18,0),
    record_text VARCHAR2(10)
);
```

```
INSERT INTO test VALUES (
    test_seq.nextval,
    'Record A'
);

INSERT INTO test VALUES (
    test_seq.nextval,
    'Record B'
);

SELECT * FROM test;
```

Хоча послідовності зазвичай створюються для однієї таблиці, в Oracle не існує обмеження, яке б цього вимагало. Послідовність є незалежним об'єктом. Вона може використовуватися в одній таблиці, в багатьох таблицях або в жодній з таблиць.

У методі, який ви лише що використовували, демонструвалося звертання до послідовності за допомогою явного посилання в операторі INSERT. Можна також зорганізовувати автоматичне звертання до послідовності, аби не посилатися на неї в операторі INSERT.

2.7.3 Синоніми

Синонім (synonym**)** дозволяє посилатися на об'єкт Oracle за іменем, яке відрізняється від його справжнього імені. Синоніми можна визначати для таблиць, уявлень, послідовностей, а також функцій, процедур і пакетів. У цьому підрозділі йтиметься про синоніми таблиць, але все подане нижче можна застосовувати й до синонімів інших об'єктів.

Створювання синоніма потрібне для зручності за частого посилання на таблицю з довгим ім'ям, без перейменування таблиці й змінювання коду, який на неї посилається.

Зручність синонімів виявляється й у тому, що вони можуть полегшити доступ до ваших даних для інших користувачів. Таблиці зорганізовуються за ідентифікатором користувача Oracle, який їх створює і якщо інший користувач захоче звернутися до таблиці, створеної вами, то в загальному випадку йому доведеться розміщувати перед ім'ям таблиці ваше ім'я користувача, як показано нижче:

```
SELECT * FROM ваше ім'я користувача. ім'я вашої таблиці;
```

Синоніми дозволяють зробити таблицю "помітною" для всіх, навіть якщо не зазначено ім'я її власника. Завдяки цьому можна писати SQL-оператори, які продовжуватимуть працювати навіть в разі передання таблиці іншому користувачеві.

Створення синоніма
Команда створення синоніма має синтаксис:

CREATE [PUBLIC] SYNONYM ім'я синоніма
FOR ім'я об'єкту
;

У наведених нижче кодах зазначено, як створити синонім.

SELECT * FROM prod;

CREATE SYNONYM prod FOR product;

SELECT * FROM prod;

Якщо вам просто треба зробити таблицю приступною для інших користувачів, створіть синонім з тим самим ім'ям, що й у таблиці. Ось приклад команди такого типу:

CREATE PUBLIC SYNONYM product FOR product;

Змодифіковування існуючого синоніма
Команда вилучання синоніма має синтаксис:

DROP [PUBLIC] SYNONYM ім'я синоніма

3 ПРОГРАМУВАННЯ МОВОЮ PL/SQL В СУРБД ORACLE В ІНФОРМАЦІЙНИХ І СЕРВІСНИХ СЛУЖБАХ ЗВ'ЯЗКУ

3.1 Введення в PL/SQL

3.1.1 Призначення PL/SQL

PL/SQL – це потужна мова програмування, яка використовується для звертання до баз даних Oracle з різних середовищ. PL/SQL є інтегрований з сервером бази даних, тому його програми можуть виконуватися швидко й ефективно. Ця мова застосовується також в певних клієнтських інструментальних засобах Oracle. Розглянемо причини використання PL/SQL і його основні властивості, а також різні версії цієї мови і систем баз даних.

Oracle – це реляційна база даних. Мови програмування, які застосовуються для звертання до реляційних баз даних, називаються **мовами структурованих запитів (SQL - Structured Query Language;**). SQL – це гнучка й ефективна мова, всі засоби якої застосовуються для маніпулювання реляційними даними і для їхнього дослідження. Наприклад, нижче показано, як за допомогою SQL-оператора можна вилучити з бази даних відомості про всіх студентів (students), профілюючою дисципліною (major) яких є живлення (nutrition):

```
DELETE EROM students
WHERE major = 'Nutrition'
```

SQL належить до мов четвертого покоління **(4GL - fourth-genernlion language)**. Це означає, що дана мова описує те, що саме треба виконати, але не в який спосіб це має бути зроблено. Приміром, в наведеному вище операторі DELETE (вилучити) невідомо, як база даних насправді визначить студентів, котрі спеціалізуються із живлення. Щоби зорієнтуватися, які записи слід вилучити, сервер, швидше за все, перегляне відомості про всіх студентів, проте реальну картину цієї операції користувачі не побачать.

Мови третього покоління, наприклад C або COBOL, за природою своєю є більш процедурні. Програма, складена мовою третього покоління **(3GL – third-generation language)**, зреалізовується шляхом виконання покрокового алгоритму. Наприклад, можна виконати операцію DELETE в такий спосіб:

```
LOOP (циклічно переглянути) запис про кожного студента
IF (якщо) для цього запису major = 'Nutrition' THEN (тоді)
DELETE (вилучити) цей запис;
END IF (завершення якщо);
END LOOP (завершення циклу);
```

У кожної мови програмування є власні переваги й недоліки. Мови четвертого покоління, подібні до SQL, як правило, вельми прості (порівняно з мовами третього покоління) і містять меншу кількість команд. Окрім того, вони ізолюють користувача від базових структур даних та алгоритмів. Проте в певних випадках процедурні конструкції мов 3GL є корисні для точнішого опису програм. Саме для цього застосовується PL/SQL, який поєднує потужність та гнучкість SQL (мови 4GL) і процедурні конструкції мови 3GL.

PL/SQL означає Procedural Language/SQL (процедурна мова/SQL). Як видно з її назви, PL/SQL розширює можливості SQL, додаючи до неї конструкції процедурних мов, такі як:

- змінні й типи (як заздалегідь визначені, так і визначувані користувачами);
- керувальні структури, такі як оператори й цикли IF- THEN- ELSE;
- процедури й функції;
- об'єктні типи й методи (PL/SQL версії 8 і вище).

Процедурні конструкції повністю поєднуються з Oracle SQL, що дає в результаті структуровану й ефективну мову програмування.

Припустімо, що треба змінити профілюючу дисципліну якогось студента. Якщо цього студента не існує, слід створити новий запис. Це можна зробити за допомогою такої програми PL/SQL:

```
DECLARE
  /* Оголосимо змінні, які використовуватимуться в SQL- операторах. */
  v_NewMajor VARCHAR2(10):= 'History';
  v_FirstName VARCHAR2(10):= 'Scott';
  v_LastName VARCHAR2(10):= 'Urman';
BEGIN
  /* Відновимо таблицю students. */
  UPDATE students
    SET major = v_NewMajor
    WHERE first_name = v_FirstName
    AND last_name = v_LastName;
  /* Перевіримо, чи знайдено запис. Якщо запис є відсутній, треба його. */
  IF SQL%NOTFOUND THEN
    INSERT INTO students (ID, first_name, last_name, major)
      VALUES (student_sequence.NEXTVAL, v_FirstName, v_LastName,
v_NewMajor);
  END IF;
END;
```

В даному прикладі використовуються два різних SQL-оператори (UPDATE (відновити) та INSERT (впровадити)), а також низка операторів оголошення змінних та умовний оператор IF (якщо).

PL/SQL є унікальною мовою, оскільки поєднує гнучкість SQL з потужністю та здатністю до конфігурування мови 3GL. У ній є як необхідні

процедурні конструкції, так і можливість звертання до бази даних. Отже ця мова програмування є надійною, ефективною, добре налаштованою для розробляння складних додатків.

3.1.2 Модель "клієнт-сервер" та її використання в інформаційних та сервісних службах зв'язку

Багато додатків для роботи з базами даних створюються з використанням моделі клієнт-сервер. Сама програма розміщується на комп'ютері клієнта і надсилає запити на дістання інформації сервера бази даних. Запити ініціюються за допомогою SQL, що, як правило, призводить до наявності в мережі великої кількості надсилань – по одному на кожного SQL-оператора. Ця ситуація ілюструється в лівій половині рис. 3.1. Тепер звернімося до його правої половини. Кілька SQL-операторів можуть поєднуватися в єдиний блок PL/SQL і надсилатися серверові як єдине ціле. Внаслідок цього мережний трафік знижується, а додаток функціонує набагато швидше.

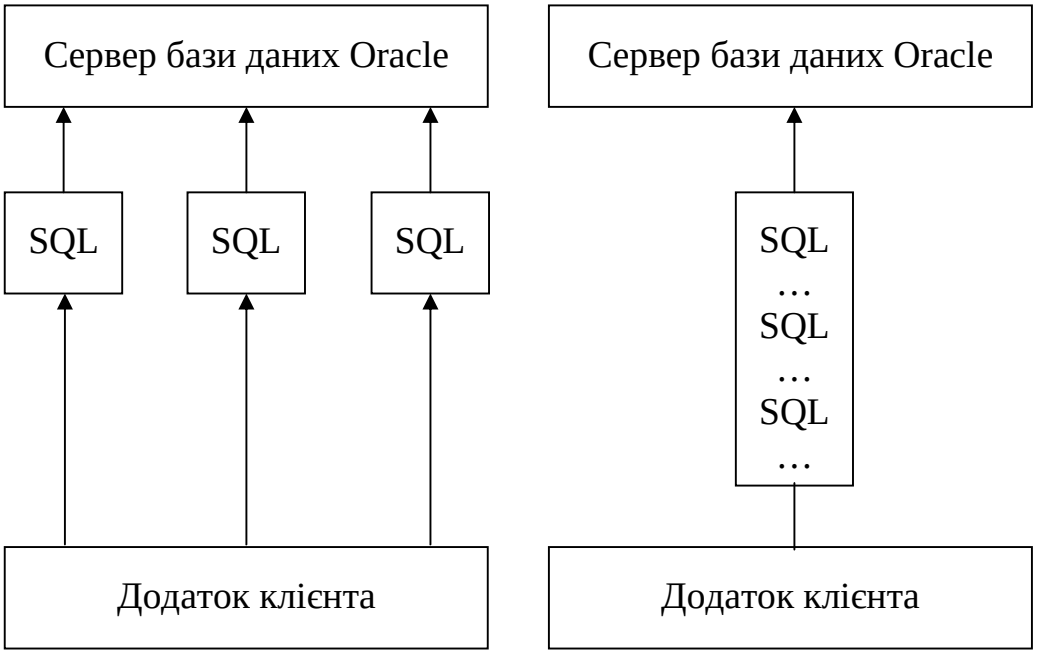

Рисунок 3.1 – PL/SQL в середовищі клієнт/сервер

Навіть якщо клієнт та сервер функціонують на одному й тому самому комп'ютері, продуктивність системи підвищується. При цьому мережа є відсутня, але пакетування SQL-операторів призводить до спрощення програми, яка тепер звертається до бази даних не так часто.

Стандарти

У СКБД Oracle мова SQL відповідає стандарту ANSI (American National Standards Institute – Американський національний інститут стандартів), окресленому документом ANSI X3. 135-1992 "Database Language SQL". Цей стандарт, відомий як SQL92 (або SQL2), визначає лише мову SQL і не описує його 3GL розширень, які пропонуються в PL/SQL. Стандарт SQL92 має три рівні узгодження: Entry (початковий), Intermediate (проміжний) та Full (повний). Oracle7 варіанта 7.2 (і решта версій вищого рівня, в тому числі й Orac1e8) відповідає стандартам Entry SQL92, що схвалено NIST (National Institute for Standards and Technology – Національним інститутом стандартів і технологій). Зараз Oracle працює з ANSI, щоб забезпечити відповідність майбутніх версій Oracle та PL/SQL повному стандарту.

3.2 Основи PL/SQL. Блок PL/SQL

3.2.1 Блок PL/SQL

Базовою одиницею будь-якої програми, написаної на PL/SQL, є блок.

З блоків складаються всі програми PL/SQL, причому блоки можуть іти один за іншим або бути вкладеними один в одного. Припустимі є такі види блоків:

- Анонімні блоки (anonymous blocks) створюються, як правило, динамічно й виконуються лише одноразово.
- Іменовані блоки (named blocks) – це анонімні блоки з позначками, які надають блокам імена. Вони також створюються, як правило, динамічно й виконуються лише одноразово.
- Підпрограми (subprograms) – це процедури, модулі й функції, зберіганні в базі даних. Ці блоки, як правило, не змінюються після свого створення й виконуються багаторазово явно за допомогою виклику процедури, модуля або функції.
- Тригери (triggers) – це іменовані блоки, які також зберігаються в базі даних. Вони теж, як правило, не змінюються після свого створення й виконуються багаторазово неявно за настання відповідних подій. Подією, яка спричинює активізування тригера, є оператор мови маніпулювання даними (DML – data manipulation language), виконуваний над певною таблицею бази даних. Оператори DML – це INSERT (впровадити), UPDATE (оновити) і DELETE (вилучити).

Наприклад, нижче наведений анонімний блок PL/SQL, за допомогою якого до таблиці **temp_table** два рядки впроваджуються, потім вибираються й відбиваються на екрані:

```
DECLARE
  /* Повідомляємо змінні, використовувані в цьому блоці. */
  v_Num1      NUMBER := 1;
  v_Num2      NUMBER := 2;
  v_String1   VARCHAR2(50) := 'Hello World!';
  v_String2   VARCHAR2(50) := '-- This message brought to you by PL/SQL!';
  v_OutputStr VARCHAR2(50);
BEGIN
  /* Спочатку за допомогою значень змінних впровадимо два рядки до
таблиці temp_table. */
  INSERT INTO temp_table (num_col, char_col)
    VALUES (v_Num1, v_String1);
  INSERT INTO temp_table (num_col, char_col)
    VALUES (v_Num2, v_String2);

  /* Тепер запросимо до temp_table два щойно впроваджених рядки й за
допомогою модуля DBMS_OUTPUT виведемо їх на екран. */
  SELECT char_col
    INTO v_OutputStr
        FROM temp_table
        WHERE num_col = v_Num1;
  DBMS_OUTPUT.PUT_LINE(v_OutputStr);

  SELECT char_col
    INTO v_OutputStr
        FROM temp_table
        WHERE num_col = v_Num2;
  DBMS_OUTPUT.PUT_LINE(v_OutputStr);
END;
/
```

Структура блока

Кожен блок складається з трьох різних розділів: розділу оголошень, виконуваного розділу й розділу виняткових ситуацій. Обов'язковий є лише виконуваний розділ; решта є необов'язкові. Наприклад, нижче наведено анонімний блок, який складається з трьох розділів:

```
DECLARE
  /* Початок розділу оголошень. */
  v_StudentID NUMBER(5) := 10000; --Числовій змінній
                                  --привласнюється значення 10000
  v_FirstName VARCHAR2(20); --Максимальна довжина
                            --послідовності символів
                             --для змінної дорівнює 20
BEGIN
```

```
 /* Початок виконуваного розділу. */
 -- Retrieve first name of student with ID 10,000
 SELECT first_name
  INTO v_FirstName
  FROM students
  WHERE id = v_StudentID;
EXCEPTION
 /* Початок розділу виняткових ситуацій. */
 WHEN NO_DATA_FOUND THEN
  -- Handle the error condition
  INSERT INTO log_table (info)
   VALUES ('Student 10,000 does not exist!');
END;
```

У цьому прикладі здійснюються посилання на дві додаткові таблиці – students і log_table.

У розділі **оголошень** розміщено всі змінні, курсори й типи, використовувані даним блоком. У цьому розділі може бути оголошено також локальні процедури й функції. Такі підпрограми будуть приступні лише в межах даного блока.

У **виконуваному** розділі зазначається, які операції виконуються в даному блоці. У цьому розділі можуть знаходитися як процедурні, так і SQL-оператори.

Помилки опрацьовуються в розділі **виняткових ситуацій**. Програмний код, який міститься в ньому, не буде виконано, якщо не станеться жодної помилки.

Розділи блока обмежені ключовими словами **DECLARE (оголосити), BEGIN (початок), EXCEPTION (виняткова ситуація) та END (кінець)**. Окрім того, наприкінці блока слід ставити крапку з комою – це синтаксичне правило є обов'язкове для блока. Отже структура анонімного блока виглядає як

```
DECLARE
/* Розділ оголошень */
BEGIN
/* Виконуваний розділ */
EXCEPTION
/* Розділ виняткових ситуацій */
END;
```

Якщо розділ оголошень є відсутній, виконання блока розпочинається з ключового слова BEGIN. Якщо є відсутній розділ виняткових ситуацій, ключове слово EXCEPTION пропускається і блок завершується ключовим словом END із крапкою з комою. Отже структура блока, що складається лише з виконуваного розділу, виглядатиме як

```
BEGIN
/* Виконуваний розділ */
END;
```

а блок з розділом оголошень та виконуваним розділом, але без розділу виняткових ситуацій, виглядатиме як

```
DECLARE
/* Розділ оголошень */
BEGIN
/* Виконуваний розділ */
END;
```

У наведених блоках конструкції, укладені в /* та */, є коментарями.

3.2.2 Оголошування змінних

У блоках PL/SQL взаємодія з базою даних здійснюється за допомогою змінних. **Змінні (variables) – це області пам'яті, в яких можуть зберігатися певні значення даних.** В міру виконання програми вміст змінних змодифіковується. Змінній може бути привласнено певну інформацію, котра зберігається в базі даних, або вміст змінної може бути внесено до бази даних. Змінні описуються в розділі оголошень блока. Кожна змінна має конкретний тип, який визначає тип інформації, котра зберігається в ній. Відомості про типи наведено в стислій формі.

Синтаксис оголошення змінних

Змінні описуються в розділі оголошень блока. Загальний синтаксис оголошення змінних такий:

```
ім'я_змінної тип [CONSTANT] [NOT NULL] [:= значення];
```

де **ім'я_змінної** – це ім'я змінної, **тип** – це тип, а **значення** – початкове значення змінної.

Нижче наведено приклади коректного оголошення змінних:

```
DECLARE
v_Description      VARCHAR2 (50);
v_NumberSeats      NUMBER := 45;
v_Counter          BINARY_INTEGER := 0;
```

Як ім'я змінної може бути використано будь-який дозволений ідентифікатор PL/SQL. Типи **VARCHAR2, NUMBER** та **BINARYINTEGER –** це коректні типи PL/SQL. У цьому прикладі для змінних **v_NumberSeats** і

v_Counter – задані початкові значення, відповідно 45 та 0. Якщо для змінної, наприклад для **v_Description,** не задано початкове значення (вона не ініціалізована), за замовчуванням їй привласнюється NULL-значення. **Якщо в оголошенні зазначено NOT NULL, – змінна повинна бути ініціалізована.** Більш того, забороняється привласнювати NULL-значення змінній, котра є обмежена як NOT NULL у виконуваному розділі або в розділі виняткових ситуацій блока. Нижченаведене оголошення є **помилкове,** тому що змінна **v_TempVar** обмежена як NOT NULL, але **не ініціалізована**:

```
DECLARE
V_TempVar NUMBER NOT NULL;
```

Це можна виправити, якщо присвоїти v_TempVar значення за замовчуванням, наприклад:

```
DECLARE
V_TempVar NUMBER NOT NULL := 0;
```

Якщо в оголошенні змінної зазначено CONSTANT, то вона повинна бути ініціалізована і її початкове значення не може бути змінено.

Змінна-константа в частині блока, що залишилася, розглядається в якості змінної "лише для читання". Константи часто використовуються для зберігання тих значень, які є відомі до моменту створення блока, наприклад:

```
DECLARE
c_MinimumStudentID CONSTANT NUMBER (5) := 10000;
```

За бажання замість := можна скористатися ключовим словом DEFAULT (за замовчуванням).

Наприклад:

```
DECLARE
V_NumberSeats Number DEFAULT 45;
V_Counter BINARY_INTEGER DEFAULT 0;
V_FirstName VERCHAR2 (20) DEFAULT 'Scott';
```

У розділі оголошень в одному рядку може бути описано лише одну змінну. Нижченаведений розділ є помилковий, тому що в однім і тім же рядку оголошуються дві змінні:

```
DECLARE
v_FirstName, v_LastName VARCHAR2 (20);
```

Конкретний варіант цього блока повинен виглядати так:

```
DECLARE
v_FirstName VARCHAR2 (20);
v_LastName VARCHAR2 (20);
```

3.3 Основи PL/SQL. Типи PL/SQL

3.3.1 Скалярні типи

У PL/SQL версій 1 та 2 було три категорії типів: скалярні, складені й посилальні. У PL/SQL 8.0 уведено додаткова категорія – типи LOB. Усередині скалярних типів не міститься жодних компонентів, тоді як у складених типах вони є присутні. Посилальний тип є покажчиком на інший тип. У таблицях 3.1, 3.2 наведено список усіх типів PL/SQL, описаних в наступних розділах.

Таблиця 3.1 – Скалярні типи PL/SQL

Скалярні типи				
Сімейство числових типів	Сімейство символьних типів	Сімейство логічних типів	Сімейство часових типів	Сімейство типів Trusted
BINARY_INTE-GER	CHAR	BOOLEAN	DATE	MLSLA-BEL
DEC	CHARACTER			
DECIMAL	LONG			
DOUBLE PRECISION	NCHAR			
FLOAT	NVARCHAR2			
INT	STRING			
INTEGER	VARCHAR			
NATURAL	VARCHAR2			
NATURALN1				
NUMBER				
NUMERIC				
PLS_INTEGER				
POSITIVE				
POSITIVEN				
REAL				
SIGNTYPE				
SMALLINT				

Таблиця 3.2 – Складені, посилальні типи, типи LOB PL/SQL

Складені типи	Посилальні типи	Типи LOB
RECORD	REF CURSOR	BFLE
TABLE	REF об'єктний тип	LOB
VARRAY		CLOB
		NLOB

Типи PL/SQL подано в модулі за назвою STANDARD. Звернутися до вмісту цього модуля можна з будь-якого блока PL/SQL. Окрім типів, у модулі STANDARD подано убудовані SQL-функції та функції перетворювання.

Скалярні типи

Припустимі скалярні типи включають типи, аналогічні до тих, що застосовуються для визначення стовпців таблиць бази даних, і низка додаткових типів. Скалярні типи можна поділити на сім сімейств: числові типи, символьні типи, типи без опрацьовування, типи дати, типи ROWID, логічні типи й типи Trusted. Усі вони описані нижче.

3.3.2 Числові типи

Сімейство числових типів

За допомогою типів цього сімейства зберігаються цілі й дійсні значення. Існує три базових типи: NUMBER, PLS_INTEGER та BINARY_INTEGER. У змінних, які мають тип NUMBER, можна зберігати як цілі, так і дійсні величини, а у змінних типу BINARY_INTEGER чи PLS_INTEGER – лише цілі числа.

Тип даних NUMBER

За допомогою даного типу можна зберігати числа, як цілі, так і з плаваючою крапкою. Синтаксис оголошення певного числа такий:

NUMBER (P,S);

де P – точність (precision), а S – масштаб (scale). **Точність** – це кількість цифр у значенні, **масштаб** – кількість цифр праворуч від десяткової крапки. Використання як точності, так і масштабу є необов'язкове, однак при зазначенні масштабу слід зазначати і точність. Різні комбінації точності і масштабу продемонстровано в таблиці 3.3.

Таблиця 3.3 – Значення точності й масштабу типу даних NUMBER

Оголошення	Привласнене значення	Збережене значення
NUMBER;	1234.5678	1234.5678
NUMBER(3);	123	123
NUMBER(3);	1234	Помилка – перевищення точності
NUMBER(4,3);	123.4567	Помилка – перевищення точності
NUMBER(4,3);	1.234567	1.235
NUMBER(7,2);	1234.67	12345.67
NUMBER(3,-3);	1234	1000
NUMBER(3,-1);	1234	1230

Зауваги:

1 Якщо масштаб значення, що присвоюється, перевищує зазначений, то зберігане значення округлюється до числа цифр, зазначеного в масштабі.

2 Якщо масштаб є від'ємний, то те значення, що зберігається, округлюється до числа цифр, зазначеного в масштабі.

Максимальною точністю є 38 цифр, а масштаб може перебувати в діапазоні від –84 до 127.

Підтип (subtype) – це альтернативне ім'я для певного типу, за допомогою якого можна обмежити множину значень, припустимих для змінних, які мають даний підтип. До типу NUMBER є еквівалентна ціла низка підтипів, що, по суті, є тим же самим, що й NUMBER, тому що жоден з них не є обмежений. Альтернативні імена можна використовувати для підвищення зручності читання програм або для забезпечення сумісності з типами даних, застосовуваними в інших базах даних. **До типу NUMBER є еквівалентні:**

DEC
DECIMAL
DOUBLE PRECISION
INTEGER
INT
NUMERIC
REAL
SMALLINT

Тип даних BINARY_INTEGER

Дані, які мають тип NUMBER, зберігаються в десятковому форматі, що його спеціально створено для точного й ефективного зберігання інформації. Унаслідок цього арифметичні операції не можна виконувати безпосередньо над значеннями типу NUMBER. Щоб числові величини могли бути опрацьовані,

значення типу NUMBER повинні бути перетворені до двійкового типу. Якщо є арифметичний вираз, в якому використовуються значення типу NUMBER, то в PL/SQL це перетворювання, а також (за необхідності) перетворювання результату назад до типу NUMBER виконуються автоматично.

Однак, якщо певне значення не зберігатиметься в базі даних, а застосовуватиметься лише в обчисленнях, можна скористатися типом даних BINARY_INTEGER. Цей тип застосовується для зберігання знакових цілих значень, які можуть перебувати в діапазоні від -2147483647 до +2147483647. Дані зберігаються у форматі точного двійкового доповнення, а це означає, що для їхнього використання в різних обчисленнях не треба виконувати їхнє перетворювання. **Тип BINARY_INTEGER** часто мають **лічильники циклів.**

Як і для NUMBER, для типу BINARY_INTEGER визначено ряд підтипів. Однак, на відміну від NUMBER, підтипи BINARY_INTEGER є **обмежені**, тобто за їхньою допомогою можна зберігати значення, на які накладено певні обмеження. Підтипи BINARY_INTEGER наведено в табл. 3.4.

Таблиця 3.4 – Підтипи BINARY_INTEGER

Підтип	Обмеження
NATURAL	0..2147483647
NATURALN	0..2147483647 NOT NULL
POSITIVE	1..2147483647
POSITIVEN	1..2147473647 NOT NULL
SIGNTYPE	-1, 0, 1

Тип даних PLS_INTEGER

Тип PLS_INTEGER має той самий діапазон значень, що й BINARY_INTEGER - від -2147483647 до +2147483647 - і зреалізований також за допомогою формату точного двійкового доповнення. Однак, якщо при виконанні певної операції переповнюється значення типу PLS_INTEGER, виникає помилка. У разі ж переповнення значення, яке має тип BINARY_INTEGER, результат може бути привласнено змінній типу NUMBER (який має більший діапазон) без жодної помилки.

3.3.3 Символьні типи

Сімейство символьних типів

Змінні цих типів використовуються для зберігання даних рядка, чи символьних даних. До цього сімейства входять типи VARCHAR2, CHAR та LONG, а також NCHAR та NVARCHAR2 (два останніх типи є приступні лише в PL/SQL версії 8.0 й вище).

Тип даних VARCHAR2

Цей тип є аналогічний до типу VARCHAR2, застосовуваному в базах даних. За допомогою змінних типу VARCHAR2 можна зберігати рядки символів змінної довжини. Синтаксис оголошення змінної, яка має тип VARCHAR2, є

VARCHAR2(L);

де **L** по – **максимальна довжина (length) змінної.** Зазначання довжини обов'язкове: **значення за замовчуванням не існує.** Максимальна довжина змінної типу VARCHAR2 становить 32 767 байт.

У Oracle8 у полі стовпця бази даних, який має тип VARCHAR2, можна зберігати 4000 байт. Отже, змінна PL/SQL VARCHAR2 може бути введена в стовпець Oracle8 VARCHAR2, лише якщо її довжина не перевищує 4000 байт.

Для значень VARCHAR2 довжина зазначається не в символах, а в байтах. Реальна інформація зберігається в базі даних за допомогою прийнятого набору символів, наприклад ASCII чи EBCDIC Code Page 500. Якщо в певному наборі символів бази даних містяться багатобайтові символи, максимальне число символів, що може зберігатися у змінній типу VARCHAR2, можливо, буде менше за зазначену довжину. Річ у тім, що для подавання одного символу може використовуватися понад один байт.

Підтип VARCHAR є еквівалентний до типу VARCHAR2.

Тип даних CHAR

Змінні цього типу являють собою рядки символів фіксованої довжини. Синтаксис оголошення змінної CHAR є

CHAR(L);

де L – максимальна довжина в байтах. Однак, на відміну від типу VARCHAR2, у цьому разі зазначання довжини є необов'язкове. Якщо її не зазначено, приймається значення за замовчуванням 1, причому круглі дужки не потрібні. Змінні типу CHAR мають фіксовану довжину, тому за необхідності вони заповнюються до максимальної довжини прогалинами. Внаслідок цього змінні типу CHAR не завжди збігатимуться при виконанні операцій порівнювання символів.

Максимальна довжина змінної типу CHAR дорівнює 32 767 байт. Максимальна ж довжина поля стовпця бази даних, який має тип CHAR, становить 255 байт. Отже, якщо у змінній CHAR міститься понад 255 байт, її можна увести лише до стовпця типу VARCHAR2 чи LONG.

Аналогічно, дані типу LONG можна вибрати у змінну типу CHAR лише в разі, якщо їхній розмір не становить понад 32 767 байт.

У Oracle8 у полі стовпця бази даних, який має тип CHAR, можна зберігати до 2000 байт.

Як і для VARCHAR2, довжина змінної типу CHAR зазначається не в символах, а в байтах. Якщо в певному наборі символів бази даних містяться багатобайтові символи, то максимальне число символів, що може зберігатися у змінній типу VARCHAR2, можливо, буде менше за зазначену довжину.

Підтипом CHAR, який має ті саме обмеження, є CHARACTER. Семантика змінних VARCHAR2 та змінних CHAR істотно розрізнюється.

Тип даних LONG

На відміну від типу LONG, застосовуваного в базах даних, що дозволяє зберігати до 2 Гбайт інформації, за допомогою типу PL/SQL LONG можна зберігати послідовності символів змінної довжини, максимальний розмір яких дорівнює 32760 байт. Змінні LONG є вельми схожі на змінні VARCHAR2. Як і у випадку з змінними VARCHAR2, якщо в полі стовпця LONG бази даних міститься понад 32760 байт інформації, вибрати цю інформацію в змінну PL/SQL LONG не можна. Однак максимальна довжина змінної PL/SQL LONG є менше, аніж поле LONG бази даних, тому змінна PL/SQL LONG може бути введена до стовпця LONG бази даних без жодних обмежень.

Типи даних NCHAR та NVARCHAR2

У Oracle8 передбачено два додаткових типи, які використовуються й у PL/SQL 8.0. Це символьні типи NLS **(NLS – National Language Support – підтримка національних мов:** NCHAR та NVARCHAR2. Вони застосовуються для зберігання рядків символів за допомогою набору символів, відмінного від того, котрий використовується в мові програмування PL/SQL. Такий набір називається **національним набором символів (national character set).**

Змінні типів NCHAR та NVARCHAR2 описуються точно так само, як і змінні типів CHAR та VARCHAR2. Однак довжина може змінюватися залежно від застосовуваного національного набору символів. Якщо в такому наборі розмір символів є фіксований, довжина зазначається в символах. Якщо ж їхній розмір є змінюваний, довжина зазначається в байтах.

3.4 Основи PL/SQL. Керувальні структури PL/SQL

3.4.1 Вирази та операції

За допомогою виразів та операцій здійснюється скріплення змінних PL/SQL. За допомогою операцій визначаються способи привласнення змінним конкретних значень і способи роботи з цими значеннями. **Вираз (expression)** – це певна послідовність змінних та літералів, розділених знаками операцій, або **операторами (operators)**. Значення виразу визначається значеннями змінних та літералів, які складають цей вираз, а також описом використовуваних операцій.

Привласнення

Основною операцією є операція привласнення (assignment). Її синтаксис є такий:

змінна := вираз;

де **змінна** – це змінна PL/SQL, а **вираз** – це вираз PL/SQL. Дана операція дозволена у виконуваному розділі і в розділі виняткових ситуацій блока. Нижче наведено низку прикладів використання операції привласнення.

```
DECLARE
v_String1 VARCHAR2 (10)
v_String2 VARCHAR2 (15)
v_Numeric NUMBER;
BEGIN
v_String1 := 'Hello';
v_String2 := 'v_String1';
v_Numeric := -12.4;
END;
```

Величина, яка знаходиться в лівій частині операції привласнення, називається значенням, яке іменує (rvalue), а величина, розташована в правій частині, – значенням виразу (rvalue). Значення, яке іменує, має посилатися на певну реальну область пам'яті, оскільки в неї записуватиметься значення виразу. У прикладі, наведеному вище, всі значення, що іменують, є змінними. PL/SQL виділяє для зберігання змінних певну область пам'яті, і значення 'Hello' та -12.4 може бути розміщено в цій області. Значення виразу може бути вмістом області пам'яті, на яку посилається змінна або літерал. У нашому прикладі проілюстровано обидва випадки: 'Hello' – це літерал, а v_Stringl – змінна. Значення виразу прочитуватиметься, а значення, що іменує, – записуватиметься. Окрім того, всі значення, що іменують, є і значеннями виразів.

На відміну від інших мов програмування, наприклад С, в будь-якому операторі (statement) PL/SQL може бути лише одна операція привласнення. Отже, подана нижче операція привласнення є помилкова:

```
DECLARE
v_Va11 Number;
v_Va12 Number;
v_Va13 Number;
BEGIN
v_Va11 := v_Va12 := v_Va13 :=0;
END;
```

Вирази

Вирази PL/SQL використовуються як значення виразів (rvalues), тому вони обов'язково повинні бути елементом іншого оператора. Наприклад, вираз може знаходитися в правій частині операції привласнення або бути елементом SQL-оператора. Тип виразу визначають операції, які складають цей вираз, а також тип їхніх операндів.

Операнд – це аргумент операції. У операціях PL/SQL використовується або один аргумент (унарна, або одномісна, операція), або два аргументи (бінарна, або двомісна, операція). Наприклад, операція заперечення (-) є унарною, а операція множення (*) – бінарною.

Пріоритет операцій виразу визначає порядок їхнього виконання. Розглянемо такий кількісний вираз:

```
3 + 5 * 7
```

Пріоритет операції множення є вищий за пріоритет операції складання, тому результатом цього виразу буде 38 (3+35), а не 56 (8*7). Щоби змінити у виразі порядок пріоритетів, заданий за умовчанням, слід використовувати круглі дужки. Наприклад, результатом наведеного нижче виразу буде 56:

```
(3 + 5) * 7
```

Символьні вирази

Існує лише одна символьна операція – це операція конкатенації, або зчеплення (| |). За допомогою даної операції з'єднуються два рядки символів (або аргументи, які може бути неявно перетворено на рядки символів). Наприклад, результатом виразу

```
'Hello' || 'World' || '!'
```

буде

```
'Hello World !
```

Якщо всі операнди у виразі конкатенації мають тип CHAR, то й воно саме має тип CHAR. Якщо ж хоча б один операнд має тип VARCHAR2, вираз має тип VARCHAR2. Вважається, що рядкові літерали мають тип CHAR, тому і результатом попереднього прикладу є значення типа CHAR. Проте в блоці, розглянутому нижче, змінній **v_Result** привласнюється вираз, результат якого – значення типу VARCHAR2:

```
DECLARE
v_TempVar VARCHAR2 (10) := 'PL'
v_Result VARCHAR2 (20);
BEGIN
v_Result := v_TempVar || '/SQL';
END;
```

Логічні вирази

У всіх керувальних структурах PL/SQL (за винятком GOTO) використовуються логічні вирази, які також називаються умовами. Логічний, або булевий вираз – це будь-який вираз, який дає в результаті логічне значення (TRUE (істина), FALSE (неправда) або NULL). Нижче наведено низку логічних виразів:

```
X > Y
NULL
(4 > 5) OR (-1 != Z)
```

У трьох операціях – AND (і), OR (чи) та NOT (ні) – логічні значення використовуються як аргументи і повертаються як результати. Можливі є ситуації, які описано в таблицях істинності, зображених на табл. 3.5. За допомогою цих операцій зреалізовується стандартна тризначна логіка. AND повертає TRUE лише в тому разі, якщо істинні є обидва операнди, OR повертає FALSE лише тоді, коли обидва операнди є помилкові.

Таблиця 3.5 – Таблиці істинності

NOT	**TRUE**	**FALSE**	**NULL**
	FALSE	TRUE	NULL
AND	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL
OR	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NULL-значення ускладнюють логічні вирази (нагадаємо, що NULL – це пропущене або невідоме значення). Результатом виразу

```
TRUE AND NULL
```

є NULL, оскільки невідомо, чи істинний другий операнд чи ні.

У операціях порівняння, або співвідношення, як операнди використовуються числа, символи або дані, а повертаються логічні значення. Нижче наведено таблицю 3.6, де описані ці операції.

Таблиця 3.6 – Головні операції співвідношення

Операція	Опис
=	Рівно (рівність)
!=	Не рівно (нерівність)
<	Менше
>	Більше
<=	Менше або рівно
>=	Більше або рівно

У операції IS NULL значення TRUE повертається лише тоді, коли операндом є NULL. NULL-значення не можуть бути перевірені на істинність за допомогою операцій співвідношення, тому що будь-який вираз співвідношення з NULL як операндом повертає NULL.

Операція LIKE (подібність) застосовується для порівняння з певним зразком рядків символів, подібно до того, як це робиться в регулярних виразах системи Unix. Знак підкреслювання (_) відповідає лише одному символу, а знак відсотка (%) – нулю й більш символів. Наведені нижче вирази повертають істинні значення (TRUE):

'Scott' LIKE 'Sc%t'
'Scott' LIKE 'Sc_tt'
'Scott' LIKE '%'

У операції BETWEEN (поміж) операції <= і >= поєднуються в одному виразі. Наприклад, наведений нижче вираз повертає помилкове значення (FALSE):

100 BETWEEN 110 AND 120

а цей вираз повертає TRUE:

100 BETWEEN 90 AND 110

Результатом операції IN (в) буде істинне значення, коли перший операнд міститься в наборі, визначуваному другим операндом. Наприклад, результат цього виразу – FALSE:

'Scott' IN ('Mike', 'Pamela', 'Fred')

Якщо в наборі містяться NULL-значення, вони ігноруються, оскільки при порівнянні певного значення з NULL-значенням завжди повертатиметься NULL.

3.4.2 Структура IF – THEN – ELSE

У PL/SQL, як і в інших мовах програмування третього покоління, є різноманітні структури, які дозволяють керувати роботою блока. Цими структурами є умовні оператори й цикли. Саме ці структури разом зі змінними забезпечують потужність та гнучкість PL/SQL.

Оператор IF – THEN – ELSE

Синтаксис оператора IF – THEN – ELSE (якщо, – то інакше):

```
IF логічний_вираз1 THEN
Послідовність_операторів1;
[ELSIF логічний_вираз2 THEN
послідовність_операторів2;]
…
[ELSE
послідовність_операторів3;
END IF;
```

де **логічний вираз –** будь-який вираз, результатом якого є логічне значення. Умови ELSIF та ELSE не є обов'язкові, причому умов ELSIF може бути скільки завгодно.

Слід звернути увагу на правопис ELSIF – в цьому слові відсутня літера E і немає пропуску. Такий синтаксис запозичено з мови програмування Ada.

NULL умови

Послідовність операторів в операторі IF – THEN – ELSE виконується лише в разі істинності відповідної умови. Якщо результатом умови є FALSE або NULL, послідовність операторів не виконується. Як приклад розглянемо два блоки:

```
/* Блок 1 */
DECLARE
v_Number1 Number;
v_Number2 Number;
v_Result VARCHAR2 (7);
BEGIN
…
IF v_Number1 < v_Number2 THEN
v_Result := 'Yes';
```

```
ELSE
v_Result := 'No';
END IF;
END;

/* Блок 2 */
DECLARE
v_Number1 Number;
v_Number2 Number;
v_Result VARCHAR2 (7);
BEGIN
…
IF v_Number1 >= v_Number2 THEN
v_Result := 'No';
ELSE
v_Result := 'Yes';
END IF;
END;
```

Розглянемо випадок, коли **v_Number1 = 3,** а **v_Number2 =** 7. В результаті умова блока 1 (3 < 7) буде істинна і змінна **v_Result** матиме значення „Yes". Умова блока 2 буде помилковою, а змінна **v_Result** також матиме значення „Yes". Для будь-яких значень **v_Number1** та **v_Number2,** що **не** є NULL-значеннями, розглянуті блоки функціонують однаково.

Тепер припустімо, що **v_Number1 = 3,** але **v_Number2** є NULL-значенням. Що відбудеться в цьому разі? Умова блока 1 (3 < NULL) дає в результаті NULL, тому буде виконана інша умова – ELSE – і змінній **v_Result** буде привласнено значення „No". Умова блока 2 також дає в результаті NULL, тому буде виконано умову ELSE і змінній **v_Result** буде привласнено значення „Yes". Якщо одна зі змінних **v_Number1** та **v_Number2** містить NULL-значення, то блоки функціонують по-різному.

3.4.3 Цикли

У PL/SQL є можливість повторювання операторів за допомогою **циклів (loops).** Цикли поділяються на чотири категорії. Прості цикли, цикли WHILE та цикли FOR, курсорні цикли FOR.

Прості цикли

Синтаксис простих циклів (основних циклів мови) є такий:

```
LOOP
послідовність_операторів;
END LOOP;
```

Послідовність операторів виконуватиметься нескінченно довго, оскільки в цьому циклі відсутня умова його завершення. Таку умову можна передбачити, якщо додати оператора EXIT (вихід), який має такий синтаксис (WHEN – коли):

EXIT [WHEN умова]

Нижче розглянуто блок, за допомогою якого до таблиці **temp_table** впроваджено 50 рядків.

```
DECLARE
v_Counter BINARY_INTEGER := 1;
BEGIN
LOOP
--Впровадимо до таблиці temp_table рядок з поточним значенням лічильника циклів.
    INSERT INTO temp_table
    VALUES (v_Counter, 'Loop index');
    v_Counter := v_Counter + 1;
--Умова виходу: коли вміст лічильника циклів перевищує 50, цикл буде завершено.
    F v_Counter > 50 THEN
    EXIT;
    END IF;
END LOOP;
END;
```

Оператор
EXIT WHEN умова
еквівалентний до оператора:

```
IF умова THEN
      EXIT;
END IF;
```

Цикли WHILE

Синтаксис циклу WHILE (циклу з умовою продовження) є:

```
WHILE умова LOOP
Послідовність_операторів;
END LOOP;
```

Перевірка умови виконується перед кожною ітерацією (кроком) циклу. Якщо умова є істинна, виконується послідовність_операторів. Якщо ж перевірка умови дає помилкове або NULL-значення, – цикл завершується і

керування програмою передається оператору, який слідує за оператором END LOOP. Тепер перепишемо даний блок за допомогою циклу WHILE:

```
DECLARE
v_Counter BINARY_INTEGER := 1;
BEGIN
--Перевіряємо вміст лічильника циклу перед кожною ітерацією циклу, щоб гарантувати, що значення лічильника є менше за 50.
  WHILE v_Counter <= 50 LOOP
    INSERT INTO temp_table
      VALUES (v_Counter, 'Loop index');
    v_Counter := v_Counter + 1;
  END LOOP;
END;
```

Щоб перервати цикл і вийти з нього, можна всередині циклу WHILE використовувати оператори EXIT та EXIT WHEN.

Якщо при першій перевірці умова циклу не є істинна, – цикл не виконується взагалі.

Числові цикли FOR

Кількість ітерацій в простих циклах та циклах WHILE є наперед невідоме: воно залежить від умов, заданих в циклах. У числових же циклах FOR кількість ітерацій є наперед визначено. Синтаксис циклу FOR є

```
FOR лічильник_циклу IN [REVERSE] нижня_межа ... верхня_межа LOOP
послідовність_операторів
END LOOP;
```

де лічильник_циклу – неявно створювана індексна змінна, нижня_межа і верхня_межа зазначають число ітерацій, а послідовність_операторів є вмістом циклу.

Межі циклу зазначаються одноразово і визначають загальну кількість ітерацій, які проходить лічильник_циклу від нижньої_межі до верхньої_межі. При цьому лічильник кожного разу збільшується на одиницю доти, поки цикл не завершиться. Подамо наш приклад за допомогою циклу FOR:

```
BEGIN
  FOR v_Counter IN 1..50 LOOP
    INSERT INTO temp_table
      VALUES (v_Counter, 'Loop Index');
  END LOOP;
END;
```

Правила, які визначають область дії лічильника

Лічильник (індекс) циклу FOR неявно оголошується з типом BINARY_INTEGER. Оголошувати його перед циклом є необов'язково. Якщо його все ж таки оголошено, цикл приховує це зовнішнє оголошення так само, як оголошення змінної у внутрішньому блоці приховує її оголошення в зовнішньому блоці:

```
DECLARE
  v_Counter  NUMBER := 7;
BEGIN
--Введемо значення 7 до таблиці temp_table.
     INSERT INTO temp_table (num_col)
     VALUES (v_Counter);
--Цей цикл зазначає v_Counter як BINARY_INTEGER, що приховує
--оголошення v_Counter як NUMBER.
      FOR v_Counter IN 20..30 LOOP
--Усередині циклу v_Counter змінюється від 20 до 30.
     INSERT INTO temp_table (num_col)
     VALUES (v_Counter);
     END LOOP;
--Ще раз введемо значення 7 до таблиці temp_table.
     INSERT INTO temp_table (num_col)
     VALUES (v_Counter);
END;
```

3.5 Записи та таблиці

3.5.1 Записи PL/SQL

Скалярні типи (NUMBER, VARCHAR2, DATE і тощо) заздалегідь окреслено в модулі STANDARD. Тому для використовування одного з таких типів в програмі слід лише оголосити змінну цього типу. Складені ж типи визначаються користувачем. Щоб застосувати складений тип, треба спочатку його описати й лише потім оголосити змінну цього типу.

Записи (records) PL/SQL є аналогічні до структур мови С. За допомогою запису можна працювати з кількома окремими, але пов'язаними змінними як з однією програмною одиницею.

Загальний синтаксис опису типу запису є

```
TYPE тип_запису IS RECORD (
поле1 тип1 [NOT NULL] [:= вираз1],
поле2 тип2 [NOT NULL] [:= вираз2],
…
полеn типn [NOT NULL] [:= виразn];
```

Тут **тип_запису** – це ім'я нового типу, **поле1 ... полеn** – імена полів запису, а **тип1 ... типn** – типи відповідних полів. У запису може бути скільки завгодно полів. Опис кожного поля провадиться аналогічно до опису змінних поза записом. Схожість цих описів полягає ще й у тому, що для полів теж можна зазначати обмеження NOT NULL та початкові значення. **Вираз1… виразn –** початкові значення для кожного поля. Подібно до опису змінної поза записом, початкове значення й обмеження NOT NULL для поля не є обов'язкові. Нижче наведено розділ оголошень, в якому спочатку визначається тип запису **t_SampleRecord**, а потім оголошуються дві змінні з цим типом.

```
DECLARE
TYPE t_SampleRecord IS RECORD (
Count          NUMBER (4),
Name           VARCHAR2 (10) := 'Scott',
EffectiveDate DATE,
Description VARCHAR2 (45) NOT NULL := 'Unkown');
v_Sample1 t_SampleRecord;
v_Sample2 t_SampleRecord;
```

Якщо поле обмежено як NOT NULL, то для нього слід зазначити початкове значення. Будь-яке поле без початкового значення ініціалізується як NULL. Для вказівки значення за умовчанням можна використовувати як ключове слово DEFAULT, так і знак :=.

Для посилання на нулі певного запису використовується уточнюючий запис через крапку. Її синтаксис є

Ім'я_запису. Ім'я_поля

Привласнення записів

Щоби привласнити одному записові значення іншого, вони мають бути одного типу. Далі наведено приклад правильної операції привласнення, в якій використовуються записи v_Sample1 та v_Sample2, що їх розглянуто вище.

```
v_Sample1 := v_Sample2
```

При виконанні подібних операцій привласнення використовується так звана **семантика копіювання:** значенням полів **v_Sample1** привласнюються значення відповідних полів **v_Sample2.** Не можна привласнювати значення записів, котрі мають **різні типи**, навіть коли описи їхніх полів збігаються. Нижче наведено приклад, який є помилковим, і внаслідок його виконання повертається повідомлення про помилку "PLS-382: expression is of wrong type" ("тип виразу невірний").

```
DECLARE
  TYPE t_Rec1Type IS RECORD (
    Field1 NUMBER,
    Field2 VARCHAR2(5));
  TYPE t_Rec2Type IS RECORD (
    Field1 NUMBER,
    Field2 VARCHAR2(5));
  v_Rec1 t_Rec1Type;
  v_Rec2 t_Rec2Type;
BEGIN
  /* Хоча записи v_Rec1 and v_Rec2 мають однакові імена й типи полів, типи їхніх записів є різні. Така операція привласнення є помилкова, і її виконання призводить до встановлення прапорця помилки PLS-382. */
  v_Rec1 := v_Rec2;

  /* Проте типи полів збігаються, тому наступні операції привласнення є правильні. */
  v_Rec1.Field1 := v_Rec2.Field1;
  v_Rec2.Field2 := v_Rec2.Field2;
END;
```

Використання %ROWTYPE

У PL/SQL широко використовується оголошення записів з тими самими типами, які мають рядки бази даних. Для полегшення цього процесу в PL/SQL пропонується операція %ROWTYPE. Як і %TYPE, %ROWTYPE повертатиме тип на підставі опису таблиці. Наприклад, опис

```
DECLARE
v_RoomRecord rooms%ROWTYPE
```

визначає запис, поля якого відповідають полям стовпців таблиці rooms. Конкретно запис **v_RoomRecord** виглядатиме як

```
(Room_id NUMBER (5),
Building VARCHAR2 (15),
Room_number NUMBER (4),
Number_seats NUMBER (4),
Description VARCHAR2 (50))
```

Як і у разі з %TYPE, обмеження NOT NULL для стовпців є відсутнє. Проте зазначено довжину полів стовпців VARCHAR2 і CHAR, а також точність та масштаб полів стовпців NUMBER.

Якщо опис таблиці змінюється, – %ROWTYPE також змінюється. Як і %TYPE, %ROWTYPE виконується кожного разу, коли анонімний блок передається PL/SQL і коли компілюється об'єкт, який зберігається.

3.5.2 Таблиці

Таблиці PL/SQL є аналогічні до масивів мови С. Синтаксично вони і є масивами, проте зреалізовані в інший спосіб. Щоб оголосити таблицю PL/SQL, спочатку треба визначити її тип, а потім оголосити змінну даного типу. Це ілюструє розділ оголошень

```
DECLARE
/* Визначимо тип таблиці. Змінні цього типу можуть містити послідовності символів, причому максимальна довжина кожної послідовності дорівнює 10 символам. */
TYPE t_CharacterTable IS TABLE OF VARCHAR2 (10)
INDEX BY BINARY_INTEGER;
/* Оголосимо змінну цього типу і тим самим виокремимо фактичне місце зберігання інформації. */
v_Characters t_CharacterTable
```

Загальний синтаксис опису таблиці є

```
TYPE тип_таблиці IS TABLE OF тип INDEX BY BINARY_INTEGER;
```

де **тип_таблиці** – ім'я нового типу, а **тип** – раніше певний скалярний тип або посилання на скалярний тип за допомогою %TYPE. У наведеному прикладі тип_таблиці – це **t_CharacterTable,** а тип VARCHAR2 (10). Розділ оголошень, наведений нижче, ілюструє опис низки різних типів таблиць та змінних PL/SQL.

```
DECLARE
TYPE t_NameTable IS TABLE OF students.first_name%TYPE
INDEX BY BINARY_INTEGER;
TYPE t_DateTable IS TABLE OF DATE
INDEX BY BINARY_INTEGER;
v_Names t_NameTable;
v_Dates t_DateTable;
```

Після того як оголошено тип та змінну, можна посилатися на окремі елементи таблиці PL/SQL в такий спосіб:

```
Ім'я_таблиці (індекс)
```

де ім'я_таблиці – це ім'я таблиці, а індекс – або змінна, що має тип BINARY_INTEGER, або змінна чи вираз, які можуть бути перетворені до типу BINARY_INTEGER. Приклад, приведений нижче, ілюструватиме це:

```
BEGIN
v_Names (1) := 'Scott';
v_Dates (-4) := SYSDATE – 1;          /* SYSDATE – 1 дає в результаті дату і час, які були 24 години тому. */
END;
```

Зверніть увагу, що посилання на таблицю, як і посилання на запис чи змінну, є іменованим виразом (lvalue), оскільки воно указує на область пам'яті, наданої PL/SQL.

Таблиці і масиви

Синтаксично таблиці PL/SQL є масивами. Проте фактична реалізація таблиці відрізняється від реалізації масиву. Таблиця PL/SQL є схожа з таблицею бази даних і містить два стовпці: KEY (ключ) та VALUE (значення). Тип ключа – BINARY_INTEGER, а тип значення – це тип, зазначений в описі.

З урахуванням опису **t_NameTable** та **v_Names** в розділі оголошень, наведеному вище, припустімо, що виконується послідовність операторів

```
v_Names (0) := 'Harold';
v_Names (-7) := 'Susan';
v_Names (3) := 'Steve';
```

Структура даних виглядатиме як

Ключ	**Значення**
0	Harold
-7	Susan
3	Steve

При роботі з розглянутими таблицями PL/SQL слід звернути увагу на наступне:

- Кількість рядків таблиці нічим не обмежене. Єдине обмеження – це значення, які можливо представити типом BINARY_INTEGER.
- Порядок елементів таблиці PL/SQL необов'язково повинен бути суворо визначений. Ці елементи зберігаються в пам'яті не підряд, як масиви, і тому можуть вводитися з довільними ключами.
- Ключі, що використовуються в таблиці PL/SQL необов'язково робити послідовними. Як індекс таблиці може бути використане будь-яке значення або вираз, що має тип BINARY_INTEGER.

Привласнення значення і-тому елементові таблиці PL/SQL створює цей елемент. Це є вельми схоже на операцію впровадження INSERT, яка виконується над таблицею бази даних. Якщо посилання на і-й елемент здійснюється до його створення, PL/SQL повертає повідомлення про помилку:

ORA- 1403: no date found
(дані не знайдено.)

Пам'ять таблиці виділяється по введенні даних і зростає зі збільшенням кількості рядків таблиці. Пам'ять не залежить від того, які значення використовуються для ключа.

Атрибути таблиць

Таблиця 3.7 – Атрибути таблиць PL/SQL

Атрибут	**Тип, що повертається**	**Опис**
COUNT	NUMBER	Повертає кількість рядків таблиці
DELETE	–	Вилучає рядки таблиці
EXISTS	BOOLEAN	Повертає TRUE, якщо зазначений елемент розташовано в таблиці
FIRST	BINARY_INTEGER	Повертає індекс першого рядка таблиці
LAST	BINARY_INTEGER	Повертає індекс останнього рядка таблиці
NEXT	BINARY_INTEGER	Повертає індекс рядка таблиці, яку розміщено за зазначеним рядком
PRIOR	BINARY_INTEGER	Повертає індекс рядка таблиці, яка передує зазначеному рядкові

Синтаксис використання атрибутів є:

таблиця.атрибут

де **таблиця** – це посилання на таблицю PL/SQL, а **атрибут** – необхідний атрибут. Атрибути таблиць PL/SQL описані в таблиці 3.7.

Атрибут COUNT

Атрибут **COUNT (число)** повертає поточне число рядків таблиці PL/SQL.

```
DECLARE
  TYPE t_NumberTable IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
```

```
  v_Numbers t_NumberTable;
  v_Total NUMBER;
BEGIN
-- Введемо 50 рядків до таблиці.
  FOR v_Counter IN 1..50 LOOP
    v_Numbers(v_Counter):= v_Counter;
  END LOOP;
  v_Total := v_Numbers.COUNT;
END;
```

v_Numbers.COUNT повертає 50, – і це значення привласнюється змінній **v_Total.**

Атрибут DELETE

Атрибут DELETE (видалити) вилучає рядки таблиці PL/SQL. Він використовується в такий спосіб:

- таблиця.DELETE – вилучити всі рядки таблиці
- таблиця.DELETE(i) – вилучується рядок таблиці з індексом і
- таблиця.DELETE (I,j) – видаляються всі рядки таблиці з індексом від і до j

Атрибути FIRST та LAST

FIRST та LAST повертають індекси відповідно **першого (first)** й **останнього (last)** рядків таблиці PL/SQL. Зверніть увагу: вони повертають не значення, котре міститься в цих рядках, а лише індекси. Перший рядок – це рядок з мінімальним індексом, а останній – з максимальним.

Атрибути NEXT та PRIOR

У атрибутах **NEXT (наступний)** і **PRIOR (попередній)** використовується один аргумент, як і в атрибуті DELETE. Вони повертають відповідно індекс **наступного** або **попереднього** елемента таблиці. Ці атрибути можна застосовувати в циклі, який перебирає всю таблицю незалежно від того, які значення використовуються в індексах.

Рекомендації щодо використовування таблиць PL/SQL

Нижче наведені рекомендації щодо використовування таблиць PL/SQL.

1 Використовуйте окрему змінну для лічильника рядків таблиці. У PL/SQL це не є обов'язково, оскільки завжди можна скористатися атрибутом COUNT, проте рекомендується застосовувати саме цей метод. Розмір таблиці не є обмежений, тому програма повинна контролювати кількість рядків, що вводяться в таблицю.

2 Розпочинайте із значення індексу, дорівнюваного 1, і збільшуйте його на 1 з кожним новим елементом. Наступний елемент повинен мати

індекс 2, далі – 3 тощо. За допомогою цього способу досить просто проходити у циклі елементи таблиці, керуючі при цьому самим процесом. Окрім того, такий спосіб індексування таблиці робить можливим скріплення її з масивом С, коли блок PL/SQL викликається з програми PRO*C або OCI чи вбудовується в таку програму.

3 Не забувайте, що елемент таблиці є не визначено до тих пір, допоки йому не привласнено певне значення. Якщо звернутися до елемента таблиці до привласнення йому значення, повертається повідомлення про помилку ORA-1403.

4 За винятком атрибуту DELETE, в PL/SQL 2.3 не існує способу вилучити всі рядки таблиці. Проте при першому створенні таблиці рядків в ній немає. Тому, якщо треба вилучити всю таблицю PL/SQL, можна привласнити їй значення порожньої таблиці:

3.6 SQL в PL/SQL

Мова структурованих запитів (SQL – Structured Query Language) визначає способи роботи з даними в Oracle. Процедурні конструкції стають набагато корисніше, коли вони поєднуються з потужними засобами опрацьовування інформації, пропонованими SQL, оскільки це дозволяє програмам PL/SQL маніпулювати даними в Oracle.

3.6.1 Використовування SQL в PL/SQL

Зі всіх SQL-операторів в програмах PL/SQL можна використовувати лише оператори DML та оператори керування транзакціями. Інакше кажучи, оператори DDL використовувати не можна. Оператор EXPLAIN PLAN (з'ясувати план), хоча він і належить до категорії DML, застосовувати також не дозволяється. Щоби пояснити значення цих обмежень, розглянемо засади створювання програм PL/SQL.

Власне кажучи, в будь-якій мові програмування прив'язка змінних може бути або ранньою, або пізньою. **Прив'язка (binding)** змінної – це процес зазначення області пам'яті, відповідної ідентифікаторові програми. У PL/SQL до процесу прив'язки входить також перевірка бази даних на наявність повноважень, які дозволяють звертатися до об'єктів схем. У тій мові, де використовується **рання прив'язка (early binding),** цей процес здійснюється на етапі компілювання програми, а в мові, де застосовується **пізня прив'язка (late binding)**, вона відкладається до етапу виконання програми. Рання прив'язка означає, що компілювання програми займатиме більший час (тому що при цьому треба прив'язувати змінні), проте виконуватися така програма буде швидше, оскільки прив'язку буде вже завершено. Пізня прив'язка скорочує час компілювання, але збільшує час виконання програми.

При розроблянні PL/SQL було ухвалено рішення про використовування ранньої прив'язки, щоби до моменту виконання блока об'єкти бази даних було вже перевірено і щоб блок міг бути виконаний максимально швидко. **Це цілком виправдано, оскільки блоки PL/SQL можна зберігати в базі даних як процедури, функції, модулі й тригери.** Такі об'єкти зберігаються в скомпільованому вигляді, тобто за необхідності їх можна завантажити з бази даних і виконати. Саме тому оператори DDL використовувати не можна. Оператор DDL змодифіковує об'єкт бази даних, отже, повноваження на об'єкт слід знову потверджувати. Процес потвердження повноважень вимагає прив'язки ідентифікаторів, а це вже було здійснено під час компілювання.

Для ілюстрації викладеного розглянемо блок PL/SQL:

```
BEGIN
CREATE TABLE temp_table (
num_value NUMBER,
char_value CHAR (10);
INSERT INTO temp_table (num_value, char_value)
VALUES (10, 'Hello');
END;
```

Щоби скомпілювати цей блок, слід прив'язати ідентифікатор **temp_table.** Процес прив'язування перевірятиме, чи існує зазначена таблиця. Допоки блок не виконано, таблиці не існує. Через це даний блок не може бути скомпільовано, а тим більше виконано.

Оператори DML і оператори керування транзакціями – це єдині SQL-оператори, за допомогою яких не можна змодифіковувати об'єкти схем або повноваження на об'єкти схем, тому в PL/SQL можна використовувати лише ці SQL-оператори.

Використання DDL

Використання мови DDL дозволяє обійти це обмеження, про яке говорилося вище.

У PL/SQL 2.1 і вище можна скористатися модулем DBMS_SQL. Він дозволяє створювати SQL-оператор динамічно, під час виконання програми, а потім провести його синтаксичний аналіз і опрацьовування.

3.6.2 DML в PL/SQL

Оператори DML – це SELECT (вибрати), INSERT (вставити), UPDATE (відновити) і DELETE (скасувати). Кожен оператор діє відповідно до власної назви: за допомогою SELECT в таблиці вибираються рядки, зазначені в умові WHERE; за допомогою INSERT до таблиці бази даних додаються рядки; за допомогою UPDATE рядки, зазначені в умові WHERE, змодифіковуються; за допомогою DELETE ці рядки скасовуються.

Оператор SELECT

За допомогою оператора SELECT дані обираються в базі даних і записуються до змінних PL/SQL. Загальний вигляд оператора SELECT наведено на рис. 3.2.

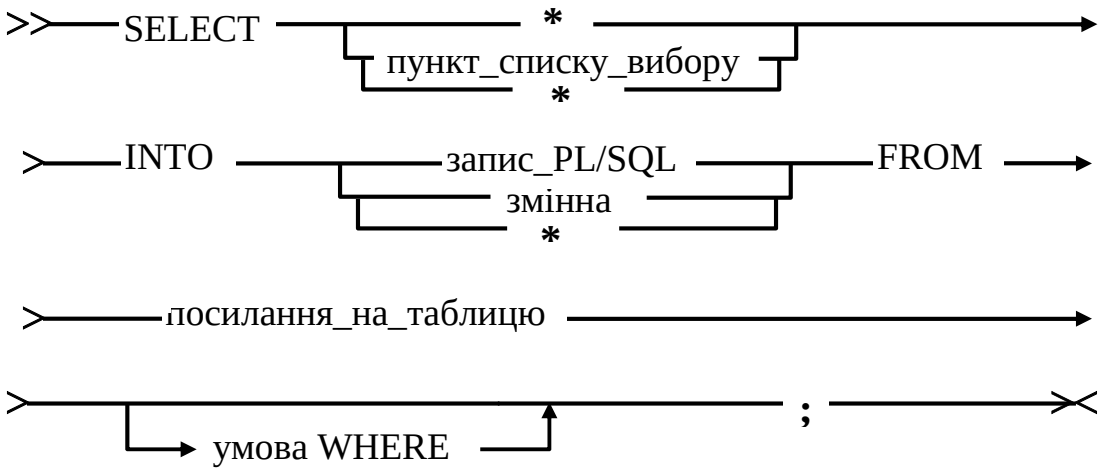

Рисунок 3.2 – Синтаксис оператора SELECT

У операторі SELECT можна використовувати велику кількість різних конструкцій, зокрема ORDER BY (упорядкувати за) і GROUP BY (згрупувати за).

За допомогою оператора SELECT, вигляд якого описано вище, можна обрати не більш одного рядка. Заданий критерій вибору співвідноситься з кожним рядком таблиці, і, якщо умові задовольнятиме кілька рядків, PL/SQL поверне повідомлення про помилку.

ORA-1427: Single-row query returns more than one row

(Однорядковий запит повертає більше одного рядка)

В цьому разі слід скористатися курсором і обрати кожен рядок окремо.

Оператор INSERT

Синтаксис оператора INSERT наведено на рис. 3.3. Зверніть увагу, що в операторі є відсутня умова WHERE (хоча її можна задати в підзапиті).

Тут **посилання_на_таблицю** указує на таблицю Oracle, **ім'я_стовпця –** на стовпець цієї таблиці, а вираз є виразом мови SQL або PL/SQL. (Посилання на таблиці детальніше обговорюються нижче.) Якщо в операторі INSERT є **оператор_вибору,** то список вибору повинен відповідати стовпцям, до яких вноситься інформація.

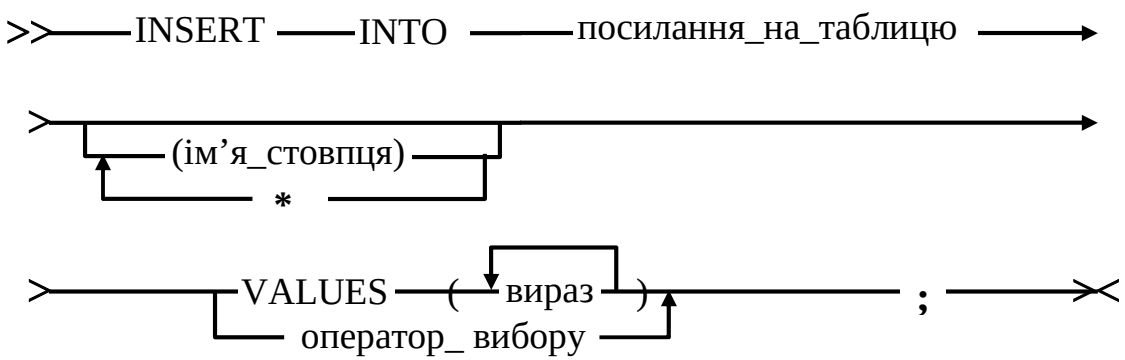

Рисунок 3.3 – Синтаксис оператора INSERT

Оператор UPDATE

Синтаксис оператора UPDATE приведено на рис. 3.4.

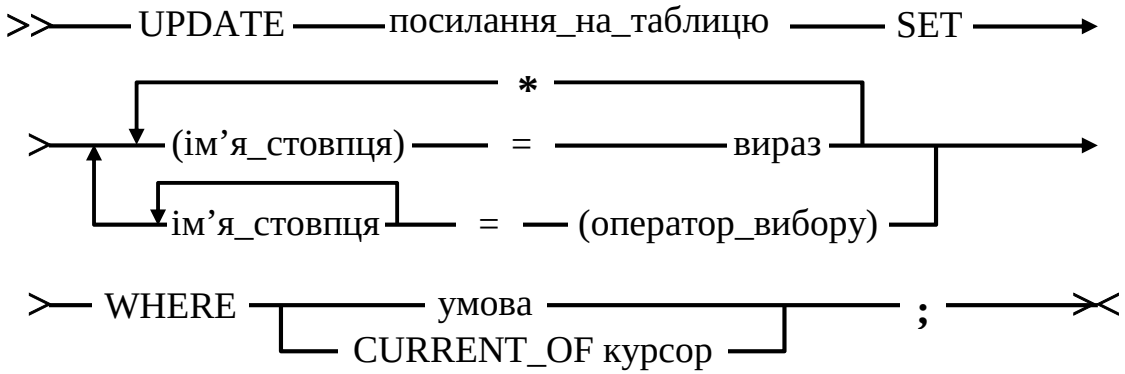

Рисунок 3.4 – Синтаксис оператора UPDATE

Тут **посилання_на_таблицю** указує на таблицю, котра поновлюється, **ім'я_стовпця –** на стовпець, значення якого буде змінено, а вираз – це вираз SQL. Якщо в операторі міститься **оператор_вибору,** то список вибору має відповідати стовпцям команди SET (встановити).

Оператор DELETE

За допомогою оператора DELETE з таблиці бази даних вилучаються рядки.

Умова WHERE оператора указує на ті рядки, які має бути вилучено. Синтаксис оператора DELETE приведений на рис. 3.5.

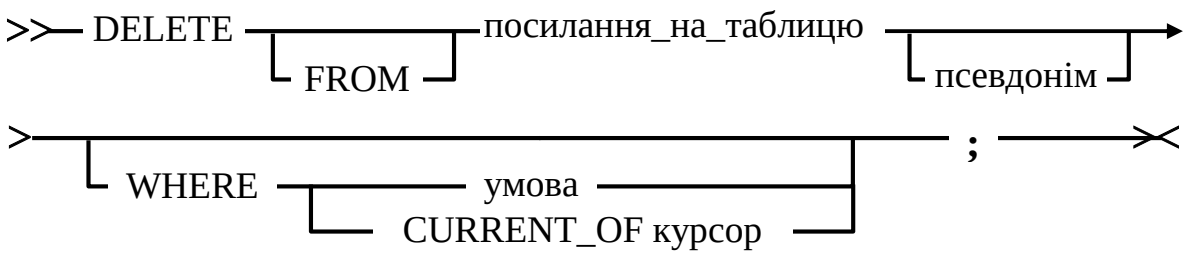

Рисунок 3.5 – Синтаксис оператора DELETE

Тут **посилання_на_таблицю** указує на таблицю Oracle, а **умова** визначає групу рядків, які буде знищено. Спеціальний синтаксис **CURRENT OF курсор** використовується разом з курсором.

Умова WHERE

До складу операторів SELECT, UPDATE та DELETE як найважливіша частина входить умова WHERE. Воно визначає **активний набір (active set)** – безлічі рядків, які повертається запитом SELECT або на які впливають оператори UPDATE чи DELETE.

Умова WHERE складається з конструкцій, сполучених логічними операціями AND (та), OR (чи) і NOT (ні). Ці конструкції зазвичай є відношеннями, як, наприклад, в такому операторі DELETE:

```
DECLARE
v_Department CHAR(3);
BEGIN
v_Department := 'CS'
-- Вилучимо всі комп'ютерні групи.
DELETE FROM classes
WHERE department = v_Department;
END;
```

За допомогою наведеного блока буде вилучено всі рядки таблиці **classes,** для яких задана умова є істинна (тобто значення нуля **department** дорівнює 'CS'). При обчисленні відносин слід звертати увагу на кілька моментів, зокрема на важливість імен змінних та на порівняння символів.

Імена змінних

Уявімо собі, що ім'я змінної, яку було розглянуто в попередньому блоці, є змінено з **v_Department** на **department:**

```
DECLARE
Department CHAR (3);
BEGIN
```

```
Department := 'CS';
-- Вилучимо всі комп'ютерні групи
DECLARE FROM classes
WHERE department = Department;
END;
```

Ця проста зміна справляє значний вплив на результати виконання оператора: за допомогою змодифікованого блока з таблиці **classes** буде вилучено всі рядки, а не лише ті, для яких значення факультету є 'CS'! Вся справа полягає в способі здійснення синтаксичного аналізу ідентифікаторів SQL-оператора. Коли PL/SQL зустрічає умову вигляду

```
вираз1 = вираз2
```

то ці вирази спочатку перевіряються на відповідність стовпцям таблиці, над якою виконується операція. Потім провадиться перевірка наявності в блоці PL/SQL змінних. PL/SQL є нечутливий до регістру символів, тому в нашому блоці як **department,** так і **Department** асоціюються зі стовпцем таблиці classes**, а не** зі змінною. Зазначена умова є істинна для кожного рядка таблиці, тому всі рядки буде вилучено.

Якщо для блока створена позначка, то для змінної та стовпця можна використовувати однакові імена. Для цього треба вказати позначку при посиланні на змінну. Нижче приведено блок, виконання якого дає потрібний ефект, тобто вилучаються лише ті рядки, в яких **department = 'CS'.**

```
DECLARE
Department CHAR (3);
BEGIN
Department := 'CS'
– Вилучимо всі комп'ютерні групи.
DELETE FROM classes
WHERE department = 1_ DeleteBlock. Department;
END;
```

Хоча застосовувати такий спосіб і дозволено, використання одного й того самого імені для змінної PL/SQL і для стовпця таблиці вважається за поганий стиль програмування.

Посилання на таблиці

У всіх операціях DML є посилання на таблиці. Загальний вигляд посилання на таблицю можна подати в такий спосіб:

```
[схема.]таблиця[@зв'язок_баз_даних]
```

де **схема** ідентифікує власника таблиці, а **зв'язок_баз_даних –** таблицю видаленої бази даних.

Щоби встановити з'єднання з базою даних, слід вказати ім'я користувача і пароль для конкретної схеми. SQL-оператори, які будуть виконуватимуться під час сеансу роботи, звертатимуться до цієї схеми за умовчанням. Якщо посилання на таблицю не є уточнене, наприклад:

```
UPDATE students
SET major = 'Music'
WHERE id = 10005;
```

то ім'я таблиці (в даному прикладі **students)** має вказувати на таблицю, яка належить до схеми, заданій за умовчанням. Якщо це не так, повертається повідомлення про помилку:

```
ORA-942:    table or view does not exist
(таблиці або уявлення не існує).
або
PLS-201:    identifier must be declared
(треба оголосити ідентифікатор).
```

Схема за умовчанням – це схема, з якою користувач з'єднується до початку виконання будь-яких команд SQL або PL/SQL. Якщо таблиця міститься в іншій схемі, на неї можна послатися за допомогою імені схеми, наприклад:

```
UPDATE example.students
SET major = 'Music'
WHERE id = 10005;
```

Даний оператор UPDATE буде виконано в тому разі, якщо з'єднання встановлене з схемою example або з іншою схемою, якій надано привілей UPDATE на таблицю students.

3.7 Вбудовані SQL-функції

У SQL є низка вбудованих функцій, які може бути викликано з SQL-оператора. Наприклад, нижче наведено оператор SELECT, в якому функція UPPER (верхній регістр) використовується для виведення імен студентів, які написано великими літерами, а не в тому вигляді, в якому вони зберігаються:

```
SELECT UPPER (first_name)
FROM students;
```

Багато SQL-функцій можуть викликатися і з процедурних операторів PL/SQL. Наприклад, в такому блоці також використовується функція UPPER в операторі привласнення:

```
DECLARE
v_FirstName students.first_name%TYPE;
BEGIN
v_FirstName := UPPER ('Chalie');
END;
```

SQL-функції можна поділити на три категорії відповідно до типу використовуваних аргументів. Наприклад, для функції UPPER передбачено використання символьного аргументу. Якщо вказати некоректний аргумент, то до виклику функції він буде автоматично перетворений PL/SQL відповідно до правил перетворювання типів даних. Окрім того, по кількості опрацьовуваних рядків SQL-функції можна поділити на групові й однорядкові. **Групові функції** опрацьовують кілька рядків даних і повертають один результат. Їх дозволено застосовувати лише в списках вибору або в конструкціях HAVING (маючи) запитів. У процедурних функціях PL/SQL їх використовування заборонене. Прикладом групової функції є функція COUNT (кількість рядків). **Однорядкові функції**, наприклад UPPER, опрацьовують одне значення, а повертають інше. Їх можна використовувати в будь-яких SQL-операторах, де дозволено застосовувати вирази, а також в процедурних операторах PL/SQL.

3.7.1 Символьні функції, що повертають символьні значення

Ці функції за аргументи використовують символи (окрім функції CHR) і повертають символьні значення. Більшість з них повертає значення, котрі мають тип VARCHAR2. Тип даних, котрий повертаються символьними функціями, має ті самі обмеження, що й основний тип бази даних, тобто значення типу VARCHAR2 обмежено 2000 символами (4000 в Oracle8), а значення типу CHAR – 255 символами (2000 в Oracle8). При використанні в процедурних операторах ці значення можна привласнювати змінним PL/SQL, котрі мають тип або VARCHAR2 або CHAR.

Символьна функція CHR
Синтаксис

CHR (x)

Призначення. Повертає символ, який має код, дорівнюваний x, в наборі символів бази даних. Функції CHR та ASCII є зворотними по відношенню одна до одної. CHR повертає символ для коду символу, а ASCII – код символу для символу.

Область застосування. Процедурні та SQL–оператори.

Символьна функція CONCAT
Синтаксис

CONCAT (рядок_символів1, рядок_символів_2)

Призначення. Повертає рядок символів 1, конкатенований (зчеплений) з рядком символів 2. Ця функція є ідентична до логічної операції ||.
Область застосування. Процедурні та SQL–оператори.

Символьна функція INITCAP
Синтаксис

INITCAP (рядок_символів)

Призначення. Повертає рядок символів, в якому кожне слово розпочинається з великої літери і продовжується малими. Слова розділяються прогалинами або не літеро-цифровими символами. Символи, котрі не є літерами, функцією не змінюються.
Область застосування. Процедурні та SQL–оператори.

Символьна функція LOVER
Синтаксис

LOVER (рядок_символів)

Призначення. Повертає рядок з рядковими символами. Символи, що не є літерами, не змінюються. Якщо тип рядка символів – CHAR**,** результат також має тип CHAR. Якщо тип рядка символів – VARCHAR2, результат також має тип VARCHAR2.
Область застосування. Процедурні та SQL–оператори.

3.7.2 Символьні функції, котрі повертають числові значення

Ці функції як аргументи використовують символи, а повертають числові значення. Аргументи можуть мати тип CHAR або VARCHAR2. Хоча фактично багато результуючих значень є цілими числами, вони мають тип NUMBER без заданих точності й масштабу.

Символьна функція ASCII
Синтаксис

ASCII (рядок_символів)

Призначення. Повертає десяткове представлення першого байта рядка символів відповідно до застосовуваного набору символів. Зверніть увагу, що ця функція називається ASCII, хоча набір символів може не бути 7-бітовим ASCII. Функції CHR і ASCII є оберненими відносно одна до одної. CHR повертає символ для коду символу, а ASCII – код символу для символу.
Область застосування. Процедурні та SQL–оператори.

Символьна функція INSTR
Синтаксис

INSTR (рядок_символів1, рядок_символів2 [,a [b]])

Призначення. Повертає місцеположення рядка символів2 в рядку символів1. Рядок символів1 є видимим ліворуч, починаючи з позиції **a.** Якщо **a** є, від'ємне, то рядок символів1 є видимим праворуч. Повертається позиція, яка вказує місцеположення **b-го** входження. Значенням за умовчаням як для **a**, так і для **b** є 1, що дає в результаті позицію першого входження рядка символів2 до рядка символів. Якщо за заданих **a** та **b** рядок символів 2 не віднайдено, повертається 0. Позиція визначається щодо початку рядка символів1 незалежно від значень a та b.
Область застосування. Процедурні та SQL–оператори.

3.7.3 Часові функції

Аргументи часових функцій мають тип DATE. За винятком функції MONTHS_BETWEEN, яка повертає значення типу NUMBER, всі функції повертають значення типу DATE.

Часова функція ADD_MONTHS
Синтаксис

ADD_MONTHS (d,x)

Призначення. Повертає дату **d** плюс **x** місяців; **x** може бути будь-яким цілим числом. Якщо в місяці, дістаному в результаті, число днів є менше, аніж в місяці **d**, то повертається останній день місяця-результату. Якщо кількість днів не є менше, то день місяця-результату й день місяця **d** збігаються. Часові компоненти дати d й результату є однакові.

Область застосування. Процедурні та SQL–оператори.

Часова функція LAST_DAY
Синтаксис

LAST_DAY (d)

Призначення. Повертає дату останнього дня того місяця, до якого входить **d**. Цю функцію можна застосовувати для визначення кількості днів, які залишилися в поточному місяці.

Область застосування. Процедурні та SQL–оператори.

Приклад:

Часова функція MONTHS_BETWEEN
Синтаксис

MONTHS_ BETWEEN (дата1, дата2)

Призначення: Повертає число місяців поміж датою1 й датою2. Якщо дні в даті1 й даті2 збігаються або якщо обидві дати є останніми днями своїх місяців, то результат є цілим числом. Інакше результат міститиме дробову частину (по відношенню до 31-денного місяця).

Область застосування. Процедурні та SQL–оператори.

Приклад:

Часова функція NEXT_DAY
Синтаксис

NEXT_DAY (d, рядок_символів)

Призначення. Повертає дату першого дня, котрий настає після дати d та позначеного рядком символів**.** Рядок символів указує день тижня мовою поточного сеансу. Часовий компонент значення, що повертається, є той самий, що й часовий компонент d. Регістр символів рядка значення не має.

Область застосування. Процедурні та SQL–оператори.

3.8 Курсори

Для опрацьовування SQL-оператора Oracle виділяє область пам'яті, яка зветься контекстною областю (context area). Вона містить інформацію, необхідну для завершення опрацьовування, зокрема: кількість рядків, опрацьовуваних оператором, покажчик на представлення цього оператора після

синтаксичного аналізу та активний набір (active set), тобто набір рядків, котрі повертаються запитом.

3.8.1 Визначення курсору

Курсор (cursor) – це покажчик на контекстну область, за допомогою якої програма PL/SQL може керувати контекстною областю та її станом під час опрацьовування оператора. Використовуючи курсор, можна окремо опрацьовувати кожен рядок пов'язаного з ним SQL-оператора. Курсор оголошується в секції оголошень базового блока. Він відкривається командою OPEN, а вибирання рядків здійснюється за допомогою команди FETCH.

Після завершення всього опрацьовування курсор закривається командою CLOSE. Закриття курсору звільнює ті системні ресурси, які використовувалися, допоки він був відкритий. Рядки, обрані курсором, можна заблокувати, щоби запобігти їхньому змодифіковуванню іншими користувачами. Закриття курсору або виконання явної операції COMMIT або ROLLBACK призведе до розблокування рядків.

Для SQL-операторів, використовуваних в коді PL/SQL, застосовуються приховані або **неявні (implicit),** курсори. **Явні (explicit)** курсори, це ті, – яким привласнено ім'я користувача.

Нижче наведено блок PI./SQL, якій ілюструє використання циклу вибирання курсора. Тут запит повертає кілька рядків даних.

```
DECLARE
 /* Вихідні змінні для зберігання результатів запиту.*/
 v_StudentID    students.id%TYPE;
 v_FirstName    students.first_name%TYPE;
 v_LastName     students.last_name%TYPE;

 /* Змінні прив'язки, використовувані в запиті.*/
 v_Major        students.major%TYPE := 'Computer Science';

 /*Створення курсора*/
 CURSOR c_Students IS
  SELECT id, first_name, last_name
    FROM students
    WHERE major = v_Major;
BEGIN
 /* Позначимо рядки активного набору і підготуємося до подальшого
опрацьовування даних. */
 OPEN c_Students;
 LOOP
  /* Обираємо кожен рядок активного набору до змінних PL/SQL.*/
  FETCH c_Students INTO v_StudentID, v_FirstName, v_LastName;
```

```
  /* Якщо рядки, які треба вибрати, закінчилися, вийдемо з циклу.*/
    EXIT WHEN c_Students%NOTFOUND;
  END LOOP;

  /* Звільнимо ресурси, використовувані у запиті.*/
  CLOSE c_Students;
END;
```

В даному прикладі ілюструється використовування **явного (explicit)** курсора. У ньому ім'я курсора явно привласнене операторові SELECT за допомогою оператора CURSOR IS. Для всіх решти SQL-операторів застосовуються **неявні (implicit)** курсори. Опрацьовування явного курсора виконується в чотири етапи. Опрацьовування неявного курсора здійснюється в PL/SQL автоматично.

3.8.2 Опрацьовування явних курсорів

Для опрацьовування явного курсора в PL/SQL слід виконати чотири кроки:

1. оголосити курсор;
2. відкрити курсор для запиту;
3. обрати результати до змінних PL/SQL;
4. закрити курсор.

Оголошення курсору є єдиним кроком, який виконується в розділі оголошень блока; інші три кроки є частиною виконуваного розділу або розділу виняткових ситуацій.

Оголошення курсора

При оголошенні курсора йому призначається ім'я і ставиться у відповідність деякий оператор SELECT. Синтаксис оголошення курсора такий:

```
CURSOR ім'я_курсора [([параметр_1 [, параметр_2…])]
[RETURN специфікація_повертання]
IS
        Оператор_SELECT
              [FOR UPDATE
                    [OF таблиця_або_стовпець_1
                          [, таблиця_або_стовпець_2…]
                    ]
              ]
;
```

де і**м'я_курсора –** це ім'я курсора, а **оператор_SELECT –** запит, який опрацьовуватиметься. Що стосується області дії й області видимості, курсори відповідають стандартним правилам, які встановлено для ідентифікаторів

PL/SQL Оскільки ім'я курсора є ідентифікатором PL/SQL, воно повинне бути оголошене до того, як на нього буде зроблено посилання. Можна використовувати будь-які оператори SELECT, зокрема з'єднання (joins), та оператори, до складу яких входять конструкції UNION (об'єднання) та MINUS (мінус).

Конструкція INTO (в) не входить до складу оператора SELECT, а є частиною оператора FETCH (обрати, вважати).

Параметри курсору схожі на параметри процедури, за тим винятком, що вони завжди є вхідними (IN). Використання параметрів OUT або IN OUT є неможливе, оскільки курсор не може їх модифікувати. Параметри використовуються в конструкції WHERE курсорного оператора SELECT.

Специфікація повертання зазначає, записи якого типу обиратимуться оператором SELECT. **Таблиця_або_стовпець –** це ім'я стовпця, який належить обновлювати, або ім'я таблиці, в якій належить вилучати або обновлювати рядки. Воно має входити до числа імен таблиць та стовпців, зазначених в операторі SELECT курсора, і призначено для документування, зазначаючи, які елементи може бути потенційно змодифіковано кодом, за використовування даного курсора. Команда FOR UPDATE блокує рядки, обрані оператором SELECT при відкриванні курсора. Рядки залишаються заблокованими доти, допоки курсор не буде закрито розглянутими вище способами.

При оголошенні курсора можна посилатися на змінні PL/SQL в умові WHERE, які розглядаються як змінні привласнення, або прив'язки. Для курсорів є чиними загальні правила визначення області дії, тому ці змінні мають бути видимі в точці оголошення курсора

Відкриття курсора

Синтаксис відкриття курсора виглядає як:

```
OPEN ім'я_курсора;
```

де **ім'я_курсора** позначає заздалегідь оголошений курсор. Коли курсор відкривається, відбуваються такі дії:

- аналізуються значення змінних прив'язки;
- на підставі значень змінних прив'язки визначається активний набір;
- покажчик активного набору встановлюється на перший рядок.

Змінні прив'язки аналізуються лише під час відкривання курсора. Розглянемо блок мови PL/SQL:

```
DECLARE
  v_RoomID      classes.room_id%TYPE;
  v_Building    rooms.building%TYPE;
  v_Department  classes.department%TYPE;
  v_Course      classes.course%TYPE;
  CURSOR c_Buildings IS
```

```
   SELECT building
     from rooms, classes
     where rooms.room_id = classes.room_id
     and department = v_Department
     and course = v_Course;
BEGIN
-- Перед відкриттям курсора привласнимо змінним прив'язки певні значення.
v_Department := 'HIS';
  v_Course := 101;

-- Відкриємо курсор.
  OPEN c_Buildings;

-- Привласнимо змінним прив'язки нові значення – це в жодний спосіб не
-- впливає на програму, оскільки курсор вже відкрито.
  v_Department := 'XXX';
  v_Course := -1;
END;
```

Коли відкрито курсор **c_Building,** у змінних **v_Department** та **v_Course** міститься відповідно 'HIS' та 101. Ці значення використовуються в запиті. Хоча після відкривання курсору значення змінних **v_Department** та **v_Course** змінилися, це в жодний спосіб не впливає на активний набір запиту. Такий алгоритм називається узгодженістю читання (read-consistency). Його розроблено для забезпечення цілісності інформації бази даних. Щоби переглянути нові значення, треба закрити і повторно відкрити курсор. Запит побачить зміни, які внесено до бази даних до виконання оператора OPEN. Якщо інший користувач змінив дані, але ще не зафіксував ці зміни, їх видно не буде.

Активний набір, або набір рядків, які задовольняють умові запиту, визначається під час відкривання курсора. Наприклад, в попередньому запиті повертається один рядок ('Building Seven'). Для таблиці (таблиць), на яку робиться посилання в конструкції FROM запиту, перевіряється умова WHERE, і всі рядки, для яких умова істинна, додаються до активного набору. Покажчик на цей набір також створюється під час відкриття курсора і повідомляє, який рядок треба зчитати курсором наступного разу.

Можна відкрити вже відкритий курсор. Перед другим відкриттям PL/SQL неявно виконує оператора CLOSE. Окрім того, будь-якого моменту може бути відкрито кілька курсорів.

Зчитування рядків з курсора

Частиною оператора FETCH є список INTO. Оператор FETCH має дві форми:

FETCH ім'я_курсора INTO список _змінних;

та

FETCH ім'я_курсора INTO запис PL/SQL;

де і**м'я_курсора** позначає заздалегідь оголошений і відкритий курсор: **список_змінних** є списком заздалегідь оголошених змінних PL/SQL, розділених комами, а **запис_PL/SQL –** заздалегідь оголошений запис PL/SQL. У будь-якому разі змінна (змінні) в конструкції INTO має мати тип, сумісний зі списком вибору запиту. З урахуванням створеного курсора **c_Buildings** виконаємо коректний оператор

FETCH c_Buildings INTO v_Building;

У прикладі, приведеному нижче, ілюструються правильні й помилкові оператори FETCH.

```
DECLARE
  v_Department  classes.department%TYPE;
  v_Course      classes.course%TYPE;
  CURSOR c_AllClasses IS
   SELECT *
     FROM classes;
  v_ClassesRecord  c_AllClasses%ROWTYPE;
BEGIN
  OPEN c_AllClasses;

-- Це правильний оператор FETCH, який повертає перший рядок до
--запису PL/SQL, який відповідає спискові вибору запиту.
FETCH c_AllClasses INTO v_ClassesRecord;

-- Цей оператор FETCH помилковий, тому що список вибору запиту
-- повертає всі сім стовпців таблиці classes, але зчитування відбувається
-- лише в дві змінні.
-- Це призведе до встановлення прапорця помилки "PLS-394: wrong
-- number of values in the INTO list of a FETCH statement".
("помилкове число значень у списку INTO оператора FETCH.")
 FETCH c_AllClasses INTO v_Department, v_Course;
END;
```

Після кожного зчитування покажчик активного набору збільшується і переходить до наступного рядка. Отже, кожен оператор FETCH послідовно повертатиме рядки активного набору, поки не буде повернено весь набір.

Для визначення моменту зчитування всього набору використовується атрибут %NOTFOUND. При виконанні останнього зчитування вихідним

змінним не привласнюватимуться нові значення, вони збережуть старі значення.

Закриття курсора

Коли вибрано весь активний набір, курсор слід закрити. Це означає, що програма завершила роботу з курсором і надані їй ресурси може бути звільнено. До складу цих ресурсів входить простір для зберігання активного набору, а також тимчасовий простір, використовуваний для визначення такого набору. Синтаксис закриття курсора є:

CURSOR ім'я_курсора;

де і**м'я_курсора** позначає раніше відкритий курсор. Після закриття курсора зчитуватати з нього рядки неможливо. Якщо спробувати це зробити, то Oracle повідомить про помилку:

ORA-1001: Invalid Cursor (помилковий курсор)

або

ORA-1002: Fetch out of Sequence (непослідовне зчитування).

Спроба закрити раніше закритий курсор також призведе до виникнення помилки ORA-1001.

3.9 Підпрограми: процедури

3.9.1 Створювання процедур і функцій

Процедури та функції PL/SQL є надто схожі на процедури та функції інших мов третього покоління і мають багато їхніх властивостей. У сукупності процедури та функції називаються **підпрограмами (subprograms).** Розглянемо фрагмент програми, яка створює процедуру в базі даних:

```
CREATE OR REPLACE PROCEDURE AddNewStudent (
  p_FirstName  students.first_name%TYPE,
  p_LastName   students.last_name%TYPE,
  p_Major      students.major%TYPE) AS
BEGIN
-- Внесемо новий рядок в таблицю students. Скористаємося послідовністю
-- student_sequence, щоби зробити ідентифікатор для нового студента і 0
--для current_credits.
INSERT INTO students (ID, first_name, last_name,
                      major, current_credits)
    VALUES (student_sequence.nextval, p_FirstName, p_LastName,
          p_Major, 0);
```

```
  COMMIT;
END AddNewStudent;
```

Після того як процедуру створено, її можна викликати з іншого блока PL/SQL в такий спосіб:

```
BEGIN
AddNewStudent ('David', 'Dinsmore', 'Music');
END;
```

Цим прикладом ілюструються важливі аспекти:

- Процедура **AddNewStudent** створюється за допомогою оператора CREATE OR REPLACE PROCEDURE. Вона спочатку компілюється, а потім зберігається в базі даних в скомпільованому вигляді. Скомпільований код можна згодом виконати з іншого блока PL/SQL.
- При викликанні процедури їй можна передавати параметри. У нашому прикладі під час виконання процедури передаються ім'я, прізвище і профілюючий предмет студента. Усередині процедури параметр p_FirstName матиме значення 'David', p_LastName – 'Dinsmore', а pMajor – 'Music', оскільки саме ці літерали було зазначено при викликанні процедури.
- Викликом процедури є оператор PL/SQL; виклик процедури не може бути частиною виразу. При викликанні процедури керування програмою передається першому виконуваному операторові цієї процедури. Коли процедура завершується, керування повертається до оператору, наступного за викликом цієї процедури. У цьому значенні процедури PL/SQL функціонують так само, як і процедури інших мов третього покоління.
- Процедура – це блок PL/SQL, до складу якого входять розділ оголошень, виконуваний розділ та розділ виняткових ситуацій. Як і в анонімних блоках, необхідним тут є лише виконуваний розділ. Наприклад, в процедурі **AddNewStudent** міститься лише виконуваний розділ.

3.9.2 Створювання процедури

Синтаксис оператора CREATE OR REPLACE PROCEDURE є:

```
CREATE [OR REPLACE] PROCEDURE ім'я_процедури
[(аргумент [{ I OUT I IN OUT}] тип,
…
аргумент [{ I OUT I IN OUT}] {IS I AS}
тіло_процедури
```

де **ім'я_процедури –** це ім'я створюваної процедури, **аргумент –** ім'я параметра процедури, **тип** – це тип відповідного параметра, а **тіло_процедури –** блок PL/SQL, в якому міститься текст процедури.

Для змінення тексту процедури слід вилучити й повторно створити її. Під час розробляння процедур ця операція виконується доволі часто, тому ключові слова OR REPLACE (або замінити) дозволяють виконати таку операцію за один раз. Якщо процедура існує, вона спочатку вилучається без жодного попередження, для чого використовується команда DROP PROCEDURE. Якщо процедура до цього не існувала, її треба створити. Якщо ж процедура існує, а ключові слова OR REPLACE не вказані, оператор CREATE повертає помилку Oracle:

ORA – 00955: name is already used by an existing object (ім'я вже використовується іншим об'єктом.)

Як і інші оператори CREATE (створювання процедури) є операцією DDL, тому до і після створення процедури неявно виконуються оператори COMMIT. При цьому можна використовувати еквівалентні один одному ключові слова IS і AS.

Параметри та їхні види

Викличемо процедуру AddNewStudent з такого анонімного блока PL/SQL:

```
DECLARE
-- Змінні, що описують нового студента.
  v_NewFirstName  students.first_name%TYPE := 'Margaret';
  v_NewLastName   students.last_name%TYPE := 'Mason';
  v_NewMajor      students.major%TYPE := 'History';
BEGIN
-- Введемо відомості про Margaret Mason до бази даних.
  AddNewStudent(v_NewFirstName, v_NewLastName, v_NewMajor);
END;
```

Змінні, оголошені в попередньому блоці **(v_NewFirstName, v_NewLastName, v_NewMajor),** передаються процедурі **AddNewStudent** як аргументи. У цьому контексті вони є **фактичними параметрами (actual parameters)**, а параметри, зазначені при оголошенні процедури **(p_FirstName, p_LastName, p_Major), – формальними параметрами (formal parameters).** Фактичні параметри містять значення, які передаються процедурі при її викликанні, і результати, котрі повертаються процедурою. Саме значення фактичних параметрів використовуються в процедурі. Формальні параметри виступають в ролі вмістища фактичних параметрів. При викликанні процедури формальним параметрам привласнюються значення фактичних параметрів. Усередині процедури всі дії виконуються над формальними параметрами. Після

завершення процедури фактичним параметрам привласнюються значення формальних параметрів. Всі операції привласнення відповідають стандартним правилам, встановленим в PL/SQL для таких операцій, зокрема правилам перетворювання типів.

Формальні параметри бувають трьох видів **(modes): IN** (в), **OUT** (з) і **IN OUT** (комбінація In та OUT) Якщо ж вигляд не зазначено, то за умовчанням встановлюється **IN**. Відмінності поміж цими видами описано в таблиці 3.12, а також таким прикладом:

```
CREATE OR REPLACE PROCEDURE ModeTest (
  p_InParameter    IN NUMBER,
  p_OutParameter   OUT NUMBER,
  p_InOutParameter IN OUT NUMBER) IS

  v_LocalVariable  NUMBER;
BEGIN
  /* Привласнимо p_InParameter змінній v_LocalVariable. Це дозволено зробити, оскільки здійснюється читання параметра IN і інформація до нього не записується. */
  v_LocalVariable := p_InParameter;  -- Правильно.
```

Таблиця 3.12 – Види параметрів

Вираз	Опис
IN	Значення фактичного параметра передається до процедури при її викликанні. Усередині процедури формальний параметр розглядається як параметр лише для читання – він не може бути змінений. Коли процедура завершується й керування програмою повертається до середовища виклику, фактичний параметр не змінюється
OUT	Будь-яке значення, котре має фактичний параметр при викликанні процедури, ігнорується. Усередині процедури формальний параметр розглядається як параметр лише для запису – йому можна лише привласнити значення, але читати з нього значення не можна. Коли процедура завершується і керування програмою повертається до середовища виклику, вміст формального параметра привласнюється фактичному параметру
IN OUT	Цей вираз є комбінацією видів IN та OUT. Значення фактичного параметра передається до процедури при її викликанні. Усередині процедури формальний параметр можна прочитати і до нього може бути записано значення. Коли процедура завершується і керування програмою повертається в середовище викликання, вміст формального параметра привласнюється фактичному параметру

/* Привласнимо 7 параметру p_InParameter. Цього зробити не можна, оскільки здійснюється запис до параметра IN. */
p_InParameter := 7; – **Неправильно.**

/* Привласнимо 7 параметру p_OutParameter. Це дозволено, оскільки здійснюється запис до параметра OUT, а не читання з нього. */
p_OutParameter := 7; – **Правильно.**

/* Привласнимо p_OutParameter змінній v_LocalVariable. Цього виконати неможливо, оскільки здійснюється читання параметра OUT. */
v_LocalVariable := p_outParameter; – **Неправильно.**

/* Привласнимо p_InOutParameter змінній v_LocalVariable. Це дозволено, оскільки здійснюється читання параметра IN OUT. */
v_LocalVariable := p_InOutParameter; – **Правильно.**

/* Привласнимо 7 параметру p_InOutParameter. Це можна зробити, так як здійснюється запис до параметра IN OUT. */
p_InOutParameter := 7; – **Правильно.**
END ModeTest;

На прикладі процедури ModeTest показано правильні і неправильні операції привласнення PL/SQL. Значення змінних можна привласнити і в конструкції INTO операторів SELECT...INTO та FETCH...INTO, на які накладаються ті ж самі обмеження.

При створюванні процедури PL/SQL перевіряє правильність операцій привласнення. Наприклад, при спробі скомпілювати ModeTest буде видано повідомлення про помилку, оскільки в цій процедурі містяться помилкові операції привласнення:

PLS – 363: expression 'p_INPARAMETER' cannon be used as an assignment target (вираз 'p_INPARAMETER' не можна використовувати як об'єкт призначення операції привласнення.)

PLS – 365: 'P_OUTPARAMETER' is an OUT parameter and cannot be read
('P_OUTPARAMETER' є параметром OUT і його не можна зчитати)

Параметр IN є значенням виразу (rvalue) усередині процедури; його можна зазначити лише в правій частині операції привласнення. Параметр OUT – це іменуюче значення (lvalue); його можна зазначати лише в лівій частині операції привласнення і ніколи в правій, навіть коли він привласнюється об'єктові в процедурі. Параметр IN OUT – це і значення виразу, й значення, що іменує, тому він може бути зазначений в будь-якій частині операції привласнення.

Якщо параметр має вигляд OUT або IN OUT, то процедура записуватиме значення до фактичного параметра. При завершенні процедури вміст формальних параметрів записується до відповідних фактичних параметрів, які повинні бути значеннями, які іменують. Літерали є значеннями виразів, і пам'ять для їхнього постійного зберігання не виділяється; тому літерали не можуть бути використані для параметрів OUT або IN OUT. Наприклад, нижче наведено правильний виклик процедури **ModeTest,** оскільки фактичними параметрами для **p_OutParameter** та **p_InOutParameter** є змінні:

```
DECLARE
v_Variable1 NUMBER;
v_Variable2 NUMBER;
BEGIN
ModeTest (12, v_Variable1, v_Variable2);
END;
```

3.9.3 Тіло процедури

Тіло (body) процедури – це блок PL/SQL, який містить розділ оголошень, виконуваний розділ та розділ виняткових ситуацій. Розділ оголошень розташовується поміж ключовими словами IS або AS та ключовим словом BEGIN, виконуваний розділ (єдиний, який є обов'язковим) – поміж ключовими словами BEGIN та EXCEPTION, розділ виняткових ситуацій – поміж ключовими словами EXCEPTION та END.

У описі процедури або функції ключове слово DECLARE є відсутнє. Замість нього використовується ключове слово IS або AS. У цьому знов виявляється аналогія з синтаксисом мови програмування Ada.

Отже, структура процедури є:

```
CREATE OR REPLACE PROCEDURE ім'я_процедури AS
/* Розділ оголошень. */
BEGIN
/* Виконуваний розділ. */
EXCEPTION
/* Розділ виняткових ситуацій. */
END [ім'я_процедури];
```

При описуванні процедури **її ім'я** можна за бажанням зазначати наприкінці останнього оператора END. Якщо після END немає жодного ідентифікатора, цей оператор повинен відповідати потрібному імені процедури. Зазначення імені процедури наприкінці є добрим стилем, оскільки виокремлює оператор END, відповідний до оператора CREATE.

3.10 Підпрограми: функції

3.10.1 Створення функцій

Функції дуже схожі на процедури. Вони також приймають аргументи, які можуть мати будь-який вид. Функції й процедури – це різні форми блоків PL/SQL, до складу кожного з них можуть входити розділ оголошень, виконуваний розділ та розділ виняткових ситуацій. Як функції, так і процедури можна зберігати в базі даних або описувати в блоці. Проте виклик процедури сам собою є оператором PL/SQL, тоді як виклик функції – це частина певного виразу.

Виклик функції є значенням виразу (rvalue). Нижче наведено приклад виклику функції, яка повертає TRUE, якщо зазначену учбову група заповнено понад на 90 відсотків, і FALSE – в іншому разі.

```
CREATE OR REPLACE FUNCTION AlmostFull (
  p_Department classes.department%TYPE,
  p_Course     classes.course%TYPE)
  RETURN BOOLEAN IS

  v_CurrentStudents NUMBER;
  v_MaxStudents     NUMBER;
  v_ReturnValue     BOOLEAN;
  v_FullPercent     CONSTANT NUMBER := 90;
BEGIN
-- Визначимо поточну й максимальну кількість студентів у зазначений групі.
  SELECT current_students, max_students
    INTO v_CurrentStudents, v_MaxStudents
    FROM classes
    WHERE department = p_Department
    AND course = p_Course;

-- Якщо відсоток заповнення групи перевищує заданий в v_FullPercent,
-- повертається TRUE. Інакше – повертається FALSE.
  IF (v_CurrentStudents / v_MaxStudents * 100) > v_FullPercent THEN
    v_ReturnValue := TRUE;
  ELSE
    v_ReturnValue := FALSE;
  END IF;

  RETURN v_ReturnValue;
END AlmostFull;
```

Функція **AlmostFull** повертає логічне значення. Нижче подано блок PL/SQL, в якому викликається ця функція. Зверніть увагу: виклик функції не є оператором, він є фрагментом умовного оператора **IF**, розташованого всередині циклу.

```
DECLARE
  CURSOR c_Classes IS
    SELECT department, course
      FROM classes;
BEGIN
  FOR v_ClassRecord IN c_Classes LOOP
-- Запишемо інформацію про всі групи, в яких залишилося
-- надто мало місця, в таблицю temp_table.
    IF AlmostFull(v_ClassRecord.department, v_ClassRecord.course)
THEN
      INSERT INTO temp_table (char_col) VALUES
       (v_ClassRecord.department || ' ' || v_ClassRecord.course ||
        'is almost full!');
    END IF;
  END LOOP;
END;
```

3.10.2 Описування функцій

Синтаксис для створення функції, котра зберігається, є вельми схожий на синтаксис для створення процедури:

```
CREATE [OR REPLACE] FUNCTION ім'я_функції
[(аргумент [{IN | OUT | IN OUT}] тип,
…
аргумент [{IN | OUT | IN OUT }] тип)]
RETURN тип_що повертається {IS | AS}
тіло_функції
```

де **ім'я_функції** – ім'я функції, в якій аргумент та тип є аналогічні до аргумента й типу, зазначених при створюванні процедури; **тип_що повертається** – це тип значення, котре повертається функцією; **тіло_функції** – блок PL/SQL, котрий містить програмний текст даної функції.

Як і для процедур, список аргументів не є обов'язковий. В цьому разі ані при описування функції, ані при її викликанні круглі дужки писати не потрібно. Проте тип, який повертається функцією, є необхідний, оскільки виклик функції є частиною певного виразу. Тип функції використовується для визначення типу виразу, який містить виклик цієї функції.

Оператор RETURN

Усередині тіла функції оператор RETURN застосовується для повертання керування програмою і результату виконання функції до середовища, яке робило її викликання. Загальний синтаксис оператора RETURN виглядає таким чином:

```
RETURN вираз;
```

де **вираз** – це значення, яке повертається. Значення виразу перетворюється до типу, що його зазначено в команді RETURN при описуванні функції, якщо це значення вже не належить до даного типу. При виконанні оператора RETURN керування програмою одразу ж повертається до середовища, котре робило її викликання.

У функції може бути кілька операторів RETURN, хоча виконуватися буде лише один з них. Завершення функції без оператора RETURN є помилкою. Нижче наведено приклад, котрий ілюструє використання в одній функції кількох операторів RETURN. За наявності п'яти різних операторів RETURN виконується лише один з них. Який саме оператор виконується, залежить від кількості студентів в навчальній групі, що зазначено за допомогою **p_Department** і **p_Course**.

```
CREATE OR REPLACE FUNCTION ClassInfo (
  /* Повертає 'Full', якщо групу заповнено на 100 %,
     'Some Room' якщо групу заповнено на 80 %,
     'More Room' якщо групу заповнено на 60 %,
     'Lots of Room' якщо групу заповнено менш ніж на 60 %, і
     'Empty' якщо жодного студента не зареєстровано. */
  p_Department classes.department%TYPE,
  p_Course     classes.course%TYPE)
  RETURN VARCHAR2 IS

  v_CurrentStudents NUMBER;
  v_MaxStudents     NUMBER;
  v_PercentFull     NUMBER;
BEGIN
-- Визначимо поточну й максимальну кількість студентів у в зазначеній
-- групі.
  SELECT current_students, max_students
    INTO v_CurrentStudents, v_MaxStudents
    FROM classes
    WHERE department = p_Department
    AND course = p_Course;
```

```
-- Обчислимо відсоток заповнення групи.
  v_PercentFull := v_CurrentStudents / v_MaxStudents * 100;
  IF v_PercentFull = 100 THEN
    RETURN 'Full';
  ELSIF v_PercentFull > 80 THEN
    RETURN 'Some Room';
  ELSIF v_PercentFull > 60 THEN
    RETURN 'More Room';
  ELSIF v_PercentFull > 0 THEN
    RETURN 'Lots of Room';
  ELSE
    RETURN 'Empty';
  END IF;
END ClassInfo;
```

При використанні оператора RETURN у функції з ним слід зв'язати вираз. Проте RETURN можна використовувати і в процедурі. В цьому разі аргументи, що приводять до негайного передання керування програмою до середовища, яке зробило виклик програми, не зазначаються. Поточні значення формальних параметрів, описаних як OUT або IN OUT, привласнюються фактичним параметрам, і виконання програми продовжується з оператора, який слідує за викликом процедури.

Властивості функцій

Багато властивостей функцій є аналогічні до властивостей процедур:

- функції можуть повертати понад одне значення за допомогою параметра виду OUT;
- програмний код функції складається з розділу оголошень, виконуваного розділу й розділу виняткових ситуацій;
- функції можуть використовувати значення за умовчанням;
- функції можна викликати, використовуючи позиційне або іменне уявлення.

Коли застосовувати функцію, а коли процедуру, залежить від того, скільки значень повинна повертати дана підпрограма і як будуть використовуватися ці значення. Зазвичай є чинне таке правило: якщо повертається понад одне значення, треба використовувати процедуру, а якщо одне, – то функцію. Хоча функції теж не заборонено мати параметри OUT (тобто повертати кілька значень), проте це не вважається за добрий стиль програмування.

3.11 Модулі

Модулі, разом з процедурами й функціями, є третім видом іменованих блоків PL/SQL. Це є надто корисний засіб, який розширює можливості мови програмування PL/SQL.

Модуль (package) – ще один засіб, який прийшов до PL/SQL з мови програмування Ada. Модуль – це конструкція PL/SQL, котра дозволяє зберігати пов'язані об'єкт в одному місці. Модуль складається з двох різних частин: опису й тіла, кожна з яких зберігається окремо в словнику даних. На відміну від процедур та функцій, які містяться локально в блоці або зберігаються в базі даних, модулі можуть бути такими, що лише зберігаються і ніколи локальними. Модулі дозволяють поєднувати пов'язані об'єкти, а також використовувати менше обмежень, визначуваних залежностями. Окрім того, вони мають низку властивостей, які сприяють підвищенню продуктивності системи.

По суті, модуль є іменованим розділом оголошень. Все, що входить до складу розділу оголошень блока, може входити й до модуля: процедури, функції, курсори, типи й змінні. Розміщення їх в модулі дозволяє посилатися на них з інших блоків PL/SQL, тому в модулях можна описувати глобальні змінні для PL/SQL.

3.11.1 Опис модуля

У опису модуля (package specification), який зветься також **заголовком модуля (package header),** міститься інформація про склад модуля, але до опису не входить текст процедур. Розглянемо приклад:

```
CREATE OR REPLACE PACKAGE ClassPackage AS
-- Додаємо нового студента до зазначеної групи
  PROCEDURE AddStudent(p_StudentID  IN students.id%TYPE,
                       p_Department IN classes.department%TYPE,
                       p_Course     IN classes.course%TYPE);
-- Вилучаємо цього студента із зазначеної групи
  PROCEDURE RemoveStudent(p_StudentID  IN students.id%TYPE,
                          p_Department IN classes.department%TYPE,
                          p_Course     IN classes.course%TYPE);
-- Виняткова ситуація, встановлювана процедурою RemoveStudent
  e_StudentNotRegistered EXCEPTION;
-- Табличний тип, використовуваний для зберігання інформації про
-- студентів
  TYPE t_StudentIDTable IS TABLE OF students.id%TYPE
    INDEX BY BINARY_INTEGER;
-- Повертає таблицю PL/SQL з відомостями про студентів, включених до
-- зазначеної групу тепер
```

```
    PROCEDURE ClassList(p_Department  IN  classes.department%TYPE,
                        p_Course      IN  classes.course%TYPE,
                        p_IDs         OUT t_StudentIDTable,
                        p_NumStudents IN OUT BINARY_INTEGER);
END ClassPackage;
```

У модулі **ClassPackage** містяться описи трьох процедур, типу і виняткової ситуації. Загальний синтаксис створювання заголовка модуля виглядає як:

```
CREATE [OR REPLASE] PACKAGE і'мя_модуля {IS | AS}
опис_процедури |
опис_функції |
оголошення_змінної |
визначення_типу |
оголошення_виняткової ситуації |
оголошення_курсора
END [ім'я_модуля];
```

де **ім'я_модуля** – це ім'я модуля. Елементи модуля (описи процедур та функцій, змінні й т. і.) є аналогічні до зазначених в розділі оголошень анонімного блока. Для заголовка модуля є чинні ті ж самі синтаксичні правила, які встановлено для розділу оголошень, за винятком оголошень процедури й функції. Перелічімо ці правила:

- Елементи модуля можуть зазначатися у будь-якому порядку. Проте, як і в розділі оголошень, об'єкт має бути оголошено до того, як на нього буде зроблено посилання. Наприклад, якщо частиною умови WHERE курсора є певна змінна, то її може бути оголошено до оголошення курсора.
- Присутність елементів всіх видів зовсім не є обов'язкова. Наприклад, модуль може складатися лише з описів процедур та функцій, без оголошення виняткових ситуацій чи типів.
- Оголошення всіх процедур та функцій повинні бути попередніми. Цим модуль відрізняється від розділу оголошень блока, де можуть міститися як попередні оголошення, так і реальний текст процедур та функцій. Програмний текст процедур та функцій модуля міститься в тілі цього модуля.

3.11.2 Тіло модуля

Тіло модуля (package body) – це об'єкт словника даних, який зберігається окремо від заголовка модуля. Тіло модуля не можна успішно скомпілювати без успішного компілювання заголовка. У тілі міститься текст

підпрограм, заздалегідь оголошених в заголовку модуля. Тіло модуля **ClassPackage** подано так:

```
CREATE OR REPLACE PACKAGE BODY ClassPackage AS
-- Додаємо нового студента до зазначеної групу.
 PROCEDURE AddStudent(p_StudentID  IN students.id%TYPE,
                      p_Department IN classes.department%TYPE,
                      p_Course     IN classes.course%TYPE) IS
 BEGIN
  INSERT INTO registered_students (student_id, department, course)
   VALUES (p_StudentID, p_Department, p_Course);
  COMMIT;
 END AddStudent;

 -- Вилучаємо цього студента із зазначеної групи.
 PROCEDURE RemoveStudent(p_StudentID  IN students.id%TYPE,
                         p_Department IN classes.department%TYPE,
                         p_Course     IN classes.course%TYPE) IS
 BEGIN
  DELETE FROM registered_students
   WHERE student_id = p_StudentID
   AND department = p_Department
   AND course = p_Course;

-- Перевіримо, чи успішно було виконано операцію DELETE. Якщо
-- зазначені рядки не знайдено, – встановлюється прапорець помилки.
  IF SQL%NOTFOUND THEN
   RAISE e_StudentNotRegistered;
  END IF;

  COMMIT;
 END RemoveStudent;

-- Повертаємо таблицю PL/SQL з відомостями про студентів, що включено до зазначеної групи щойно
 PROCEDURE ClassList(p_Department  IN  classes.department%TYPE,
                     p_Course      IN  classes.course%TYPE,
                     p_IDs         OUT t_StudentIDTable,
                     p_NumStudents IN OUT BINARY_INTEGER) IS

  v_StudentID  registered_students.student_id%TYPE;

-- Локальний курсор для вибору зареєстрованих студентів
  CURSOR c_RegisteredStudents IS
```

```
    SELECT student_id
      FROM registered_students
      WHERE department = p_Department
      AND course = p_Course;
  BEGIN
   /* p_NumStudents буде індексом таблиці. Цей індекс розпочинатиметься з нуля і збільшуватиметься з кожним повторенням циклу вибірки. Наприкінці циклу в індексі міститиметься число лічених рядків, тобто число рядків, котрі повертаються до p_IDs. */

    p_NumStudents := 0;

    OPEN c_RegisteredStudents;
    LOOP
     FETCH c_RegisteredStudents INTO v_StudentID;
     EXIT WHEN c_RegisteredStudents%NOTFOUND;

     p_NumStudents := p_NumStudents + 1;
     p_IDs(p_NumStudents):= v_StudentID;
    END LOOP;
   END ClassList;
END ClassPackage;
```

Текст підпрограм, заздалегідь оголошених в заголовку модуля, міститься в тілі цього модуля. На об'єкти в заголовку, що їх не оголошено попередньо (наприклад e_StudentNotRegistered), можна посилатися в тілі модуля без повторного оголошення.

Тіло модуля не є обов'язковою його частиною. Якщо в заголовку не зазначено певні процедури або функції (а лише змінні, курсори, типи тощо), тіло можна не створювати. Такий спосіб є доцільний при оголошенні глобальних змінних, оскільки всі об'єкти модуля є видимі поза його межами.

Будь-яке попереднє оголошення в заголовку модуля має бути розкрито в його тілі. Опис процедури чи функції має бути таким самим і включати до свого складу ім'я підпрограми, імена її параметрів та вид кожного параметра. Наприклад, наведений нижче заголовок не відповідає тілу модуля, тому що в тілі використовується список параметрів функції **FunctionA**, відмінний від списку параметрів, зазначеного в заголовку.

```
CREATE OR REPLACE PackageA AS
FUNCTION FunctionA(p_Paramater1 IN NUMBER,
                   p_Paramater2 IN DATE)
RETURN VARCHAR2;
END PackageA;
CREATE OR REPLACE PACKAGE BODY PackageA AS
```

FUNCTION FunctionA(p_Paramater1 IN CHAR)
RETURN VARCHAR2;
END PackageA;

Якщо спробувати створити PackageA в такий, як показано вище, то дістанемо повідомлення про помилки:

PLS-00328: A subprogram body must be defined for the forward declaration of FUNCTIONA.

(Тіло підпрограми має бути описано в такий спосіб, як це зроблено в попередньому оголошенні FUNCTIONA.)

PLS-00323: subprogram or cursor 'FUNCTIONA'is declared in a package specification and must be defined in the package body.

(підпрограму (курсор) оголошено в описі модуля і має бути описано в тілі модуля.)

3.11.3 Модулі і області дії

Будь-який об'єкт, оголошений в заголовку модуля, перебуває у області дії і його видно поза межами цього модуля. Для звертання до об'єкта треба зазначити ім'я модуля при посиланні на цей об'єкт. Наприклад, можна викликати процедуру **ClassPackage.RemqveStudent** з блока PL/SQL:

BEGIN
ClassPackage. RemoveStudent (10006, 'HIS', 101);
END;

При цьому викликання процедури є аналогічне до виклику процедури, не включеної до модуля. Єдина відмінність такого виклику – присутність перед ім'ям процедури імені модуля. Для модульних процедур можуть задаватися параметри за умовчанням, і викликати такі процедури можна за допомогою як позиційного, так і іменного уявлення, тобто так само, як і звичайні процедури, які зберігаються.

Усередині тіла модуля можна посилатися на об'єкти, наявні представлені в його заголовку без зазначення імені модуля. Наприклад, процедура **RemoveStudent** може посилатися на виняткову ситуацію просто як на **e_StudentNotRegistered,** а не як на **ClassPackage.e_StudentNotRegistered.** Проте можна скористатися як повним, так і уточненим ім'ям.

3.12 Тригери

3.12.1 Створювання тригера

Четвертий тип іменованих блоків PL/SQL – тригери. Тригери багато в чому є схожі на підпрограми, але певні їхні характеристики істотно розрізнюються.

Тригери так само, як процедури й функції, є іменованими блоками PL/SQL з розділом оголошень, виконуваним розділом та розділом виняткових ситуацій. Подібно до модулів, тригери треба зберігати в базі даних, а не локально в блоці. Процедура викликається в явний спосіб з іншого блока, причому при викликанні процедури їй можуть передаватися різні аргументи, а тригер виконується неявно, кожного разу, коли відбувається подія, котра запускає цей тригер, причому використання аргументів не припускається. Акт виконання тригера називається його **активізацією (firing).** Запускається тригер операцією DML (INSERT, UPDATE або DELETE), виконуваною над базою даних. Тригери можна використовувати для:

- реалізації складних обмежень цілісності даних, які неможливо здійснити через описові обмеження, встановлювані при створюванні таблиці;
- стеження за інформацією, яка зберігається в таблиці, шляхом записування змін, котрі вносяться, й користувачів, які вносять ці зміни;
- автоматичного повідомлення інших програм про те, що слід чинити в разі змінювання інформації, котра міститься в таблиці.

Припустімо, що треба відстежувати статистичні показники, стосовні різних профілюючих дисциплін студентів, зокрема кількість зареєстрованих студентів і загальна кількість одержаних заліків. Результати зберігатимуться в таблиці **major_stats:**

```
CREATE TABLE major_stats (
Major            VARCHAR2 (30)
Total_credits    NUMBER,
Total_Students   NUMBER);
```

Для стеження за актуальністю інформації в таблиці **major_stats** створимо тригер для таблиці **students**, який оновлюватиме **major_stats** кожного разу за змінювання students. Назвемо цей тригер UpdateMajorStats. Він вмикатиметься при виконанні будь-якої операції DML над **students.** Тіло тригера звертається до таблиці **students** із запитом і оновлює статистичні показники таблиці **major_stats** свіжою інформацією**:**

```
CREATE OR REPLACE TRIGGER UpdateMajorStats
/* Підтримує актуальність таблиці major_stats, відстежуючи всі зміни, які вносяться до таблиці students. */
  AFTER INSERT OR DELETE OR UPDATE ON students
DECLARE
  CURSOR c_Statistics IS
    SELECT major, COUNT(*) total_students,
          SUM(current_credits) total_credits
      FROM students
      GROUP BY major;
BEGIN
  /* Послідовно проглянемо інформацію про кожну дисципліну і спробуємо відновити статистичні показники major_stats, відповідні дисципліні. Якщо рядка не існує, створимо його. */
  FOR v_StatsRecord in c_Statistics LOOP
    UPDATE major_stats
      SET total_credits = v_StatsRecord.total_credits,
        total_students = v_StatsRecord.total_students
      WHERE major = v_StatsRecord.major;
    /* Перевіримо наявність рядка.*/
    IF SQL%NOTFOUND THEN
      INSERT INTO major_stats (major, total_credits, total_students)
        VALUES (v_StatsRecord.major, v_StatsRecord.total_credits,
              v_StatsRecord.total_students);
    END IF;
  END LOOP;
END UpdateMajorStats;
```

Синтаксис створення тригера є:

```
CREATE [OR REPLACE] TRIGGER ім'я_тригера
{BEFORE | AFTER } активізуюча_подія ON посилання_на_таблицю
[FOR EACH ROW [WHEN]]
тіло_тригера;
```

де **ім'я_тригера** – це ім'я тригера, **активізуюча_подія** зазначає момент активізації тригера (у разі з **UpdateMajorStats** після будь-якої операції DML), **посилання_на_таблицю** – таблиця, для якої створено тригер, а **тіло_тригера** – програмний текст тригера. Якщо присутня умова_виконання в умові WHERE (коли), то спочатку вона обчислюється, а тіло тригера виконується лише в тому разі, якщо ця умова є істинна.

3.12.2 Елементи тригера

Обов'язковими елементами тригера є його ім'я, котре активізує подію й тіло. Умова WHEN необов'язкова.

Імена тригерів

Простір імен тригерів відрізняється від просторів імен інших підпрограм. Простором імен (namespace) називається набір ідентифікаторів, дозволених для використання як імені об'єкта. Для процедур, модулів та таблиць застосовується один і той самий простір імен. Це означає, що в межах однієї схеми бази даних всі об'єкти, котрі використовують один і той самий простір імен, повинні мати унікальні імена. Наприклад, **процедурі** й **модулеві** забороняється давати однакові імена.

Для тригерів визначено власний простір імен, тобто тригер може мати те саме ім'я, що й будь-яка таблиця або процедура. Проте в межах однієї схеми конкретне ім'я може бути надано лише одному тригерові. Імена тригерів – це ідентифікатори бази даних і тому підпорядковуються тим самим правилам, що й інші ідентифікатори. Наприклад, для таблиці major_stats можна створити тригер **major_stats**, але процедуру з ім'ям **major_stats** створити не можна. Нижче наведено приклад сеансу роботи в SQL*Plus:

```
SQL> CREATE OR REPLACE TRIGGER major_stats
        BEFORE INSERT ON major_stats
BEGIN
        INSERT INTO temp_table (char_col)
        VALUES ('Trigger fired!');
END major_stats;
/
Trigger created.

SQL> CREATE OR REPLACE PROCEDURE major_stats AS
BEGIN
         INSERT INTO temp_table (char_col)
        VALUES ('Procedure called!');
END major_stats;
/

CREATE OR REPLACE PROCEDURE major_stats AS
*
ERROR at line 1;
ORA-00955: name is already used by an existing object
(ім'я вже використовується існуючим об'єктом.)
```

Хоча застосовувати для таблиці й тригера однакові імена й не заборонено, робити це не рекомендовано. Краще дати кожному тригерові унікальне ім'я, котре зазначатиме, які функції він виконує, так само, як і таблиці, для якої він створений.

Типи тригерів

Тип тригера визначається тим, яка саме подія його активізовує: INSERT (введення), UPDATE (оновлення) або DELETE (вилучання). Тригери можуть активізовуватися до (BEFORE) або після (AFTER) операції, а також до рядка або оператора. Можливі варіанти тригерів наведено в таблиці 3.13.

Таблиця 3.13 – Типи тригерів

Категорія	Значення	Коментарі
Оператор	INSERT, UPDATE або DELETE	Визначає, який оператор DML спричинює активізування тригера
Момент часу	BEFORE або AFTER	Визначає момент активізування тригера: до і після виконання оператора
Рівень	Рядок або оператор	Якщо тригер є рядковим, то він активізується одноразово для кожного з рядків, на які впливає оператор, який спричиняє спрацьовування тригера. Якщо тригер є операторним, то він активізується одноразово до й після оператора. Рядкові тригери містять умову FOR EACH ROW (для кожного рядка) в описі тригера

Значення, задані для оператора, моменту часу й рівня, визначають тип тригера. Всього існує 12 можливих типів. Нижче наведено приклади правильних тригерів:

- до виконання операції оновлювання на операторному рівні;
- після виконання операції впровадження на рівні рядків;
- до виконання операції на рівні рядків.

Для таблиці може бути створено до 12 тригерів – по одному для кожного типу. Проте, розпочинаючи з PL/SQL 2.1 (Oracle7 версії 7.1), для таблиці може створюватися по кілька тригерів кожного виду. Це надає можливість визначати для однієї таблиці скільки завгодно тригерів.

Окрім того, тригер може активізовуватися для кількох типів операторів, які спричиняють його спрацьовування. Наприклад, тригер UpdateMajorStats спрацьовує на оператори INSERT, UPDATE та DELETE. Подія, котра активізує, зазначає одну чи кілька операцій DML, які спричинюють спрацьовування тригера.

У PL/SQL 8.0 пропонується ще один вид тригерів – **INSTEAD OF** (замість), які можна створювати лише для уявлень (об'єктних чи реляційних); причому такі тригери зактивізовуватимуться замість операторів DML, які спричиняють їхнє спрацьовування, тригери INSTEAD OF повинні бути рядковими тригерами. Такі тригери є необхідні, оскільки в основі уявлень, для яких вони створюються, можуть лежати з'єднання (joins), а не всі з'єднання поновлюються. Тригер же дозволяє виконувати оновлення інформації в належний спосіб. Для прикладу розглянемо подання **room_summary.**

```
CREATE VIEW room_summary AS
  SELECT building, sum(number_seats) total_seats
    FROM rooms
        GROUP BY building;
```

Вилучити інформацію безпосередньо з цього подання неможливо:

```
SQL> DELETE FROM room_summary where building = 'Building 7';
DELETE FROM room_summary where building = 'Building 7';
                    *
ERROR at line 1:
ORA-01732: data manipulation operation not legal on this view
```

(Операція маніпулювання даними для цього уявлення заборонена.)

Проте можна створити тригер INSTFAD OF і за його допомогою виконати потрібну операцію – вирізати зазначені рядки таблиці **rooms:**

```
CREATE TRIGGER room_summary_delete
  INSTEAD OF DELETE ON room_summary
  FOR EACH ROW
BEGIN
-- Вилучимо всі рядки таблиці rooms, котрі відповідають рядку подання
-- room_summary
  DELETE FROM rooms
    WHERE building = :old.building;
END room_summary_delete;
```

За допомогою тригера room_summary_delete оператор DELETE виконується успішно і виконує саме те, що слід.

Обмеження, котрі накладаються на тригери

Тіло тригера є блоком PL/SQL. Будь-який оператор, виконання якого дозволено в блоці PL/SQL, можна виконати і в тілі тригера за умови дотримання таких обмежень:

- У тригері не можна задавати жоден з операторів керування транзакціями: COMMIT, ROLLBACK або SAVEPOINT. Спрацьовування тригера є частиною процесу виконання

активізуючого оператора, тобто частиною тієї транзакції, яка охоплює й активізує оператор. Якщо цей оператор завершується або відкочується, все те, що виконано тригером, також завершується або відкочується.

- У процедурах й функціях, які викликаються в тілі тригера, також не можна задавати оператори керування транзакціями.
- У тілі тригера не можна оголошувати змінні з типами LONG та LONG RAW. Окрім того, в псевдозаписах :new і :old не можна посилатися на стовпці типів LONG та LONG RAW таблиці, для якої визначено тригер.

З тіла тригера можна звертатися не до всіх таблиць. Залежно від типу тригера та обмежень (constraints),які накладаються на таблиці, таблиці можуть бути змінюваними (**mutating).**

3.13 Об'єктно-орієнтоване програмування

Об'єкти є одним з основних нових засобів Oracle8 і PL/SQL 8. У цьому підрозділі буде розглянуто питання створювання і використання об'єктів, в тому числі йтиметься про методи і конструктори, а також про посилання на об'єкти. Окрім того буде розглянуто питання зберігання об'єктів у базі даних й маніпулювання ними в операторах DML.

3.13.1 Основи об'єктно-орієнтованого програмування

Програмні засоби розробляються для імітування як реальних об'єктів, так і взаємозв'язків поміж ними. Використовуючи моделі об'єктів та способів їхньої взаємодії, додаток може контролювати еволюцію об'єктів і автоматизувати процеси, які відбуваються в перебігу такої еволюції.

За приклад розглянемо модель вищого навчального закладу (ВНЗ). Які елементи моделі є головними, або сутностями? У ВНЗ навчаються студенти, яких деканати розподіляють по навчальних групах. Деканат своєю чергою інформує викладачів про кількість студентів у групах і про склад цих груп. Викладачі взаємопов'язані зі студентами під час навчання і проведення іспитів. Робітники бібліотеки повинні знати, які книжки викладачі рекомендують студентам і забезпечувати студентів цими книжками.

Цю модель подано на рис. 3.12. Де кола відображають сутності, які діють в моделі університету, а стрілки – взаємодію сутностей поміж собою, приміром отримання книжок у бібліотеці.

Це є доволі інформативна модель, за аналізування якої можна здобути різні відомості про реальний світ. Наприклад, можна побачити, що інспектор є центральною фігурою в цій моделі, оскільки решта сутностей мають з ним взаємодіяти.

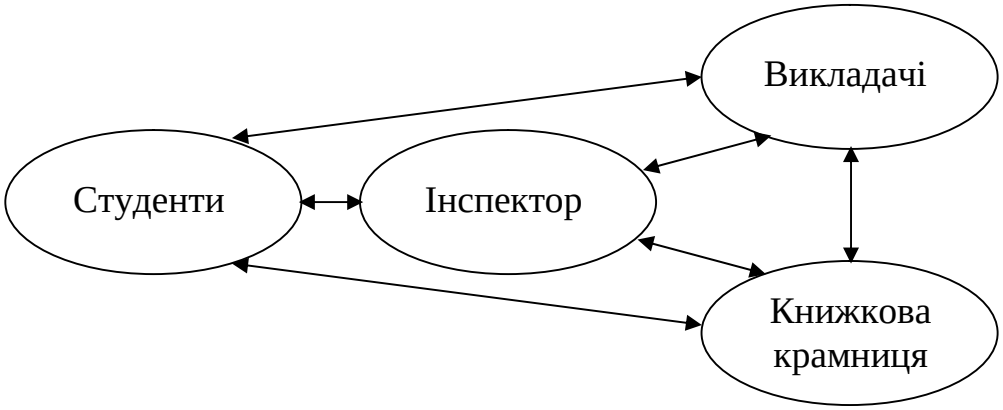

Рисунок 3.12 – Модель університету

За допомогою об'єктно-орієнтованого програмування на підставі цієї моделі створюється комп'ютерний додаток. Кожна сутність в системі відбиває конкретний об'єкт. Об'єкт показує властивості, або атрибути, реальної сутності й операції, які дозволено виконувати над ними.

Розглянемо об'єкт Student, який відповідає конкретному студентові (рис. 3.13). Студент має атрибути – властивості (**attributes)**: ім'я (**first_name**), прізвище (**last_name**), профілюючу дисципліну (**major**) і поточну кількість отриманих заліків (**current_credits**). Окрім того, на рисунку показано операції, які справляють вплив на ці атрибути: **ChangeMajor** (змінює профілюючу дисципліну) і **UpdateCredits** (поновлює відомості про заліки студента у групі). Ці операції мають назву **методи (methods).**

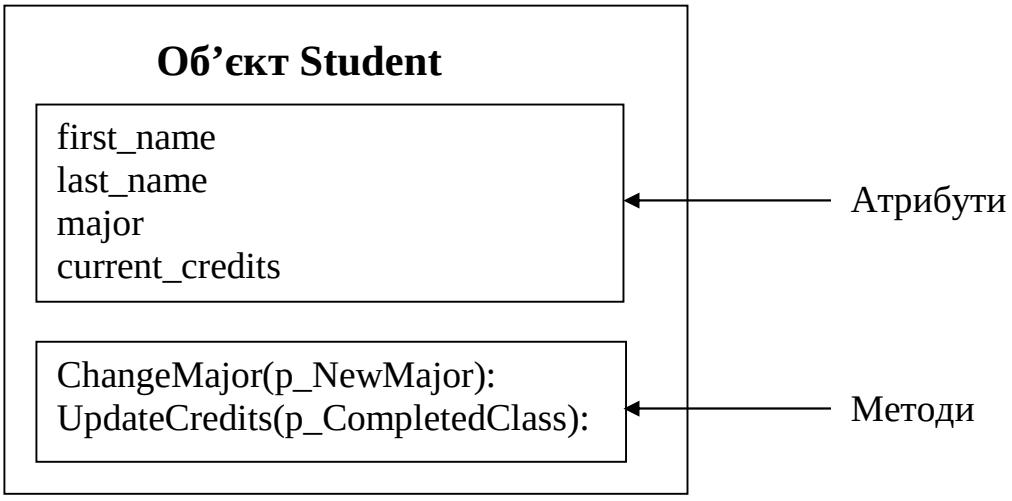

Рисунок 3.13 – Об'єкт Student

Об'єкти взаємодіють поміж собою, зумовлюючи використання конкретних методів. Наприклад, інспектор може використати метод **UpdateCredits** після закінчення одним із студентів курсу навчання.

Абстракція

Атрибути і методи об'єкта доволі точно зреалізовують абстрактне представлення даних та процедур. У ідеалі клієнт, котрий використовує певний об'єкт, керує атрибутами лише через методи. При цьому додаток клієнта може просто викликати методи, не знаючи про те, як вони працюють реально.

Об'єкти та екземпляри об'єктів

Слід зазначити відмінність, яка існує поміж власне об'єктним типом та **екземпляром (instance)** цього типу. В системі може бути лише один об'єктний тип з конкретним ім'ям, але багато примірників цього типу. Примірник об'єкта є схожий на змінну: кожен примірник має власну область пам'яті і, як наслідок, власну копію атрибутів об'єкта. Наприклад, на рис. 3.14 подано два примірники об'єктного типу **StudentObj.** Вони є аналогічні до двох різних записів PL/SQL, оголошених з одним і тим самим типом.

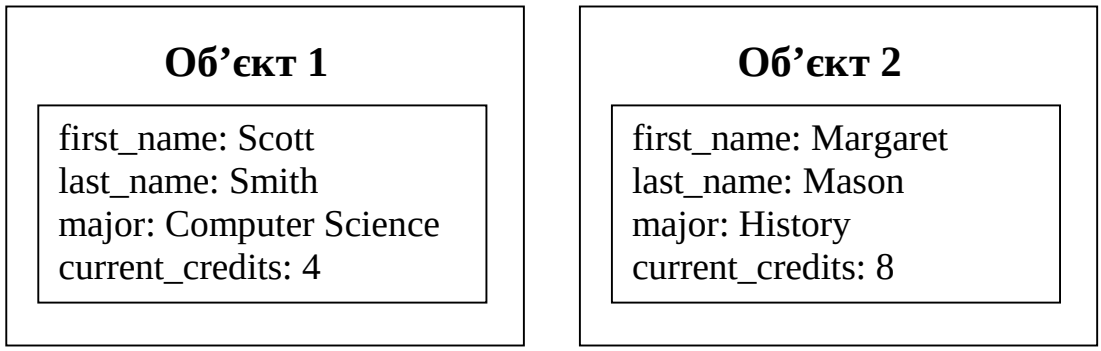

Рисунок 3.14 – Примірники типа StudentObj

Сьогодні застосовується безліч об'єктно-орієнтованих мов програмування, зокрема C++ та Java. Такі мови надають можливість описувати об'єкти й маніпулювати ними, проте мають істотний недолік – вони не забезпечують надійного й коректного зберігання та використання об'єктів. Саме для цього q призначено об'єктно-реляційні бази даних, подібні Oracle8. Систему Oracle8 створено для зберігання об'єктних даних і для роботи з ними. Керування об'єктними даними є аналогічне до керування реляційними даними і здійснюється за допомогою мови SQL, яка виступає в ролі засобу взаємодії з базами даних. У об'єктно-реляційній базі даних мова SQL (і PL/SQL) використовується для маніпулювання як реляційними, так і об'єктними даними. Окрім того, Orac1e8 забезпечує: ефективне керування транзакціями, надійне резервне копіювання й відновлення інформації, високопродуктивне опрацьовування запитів, блокування даних, паралельність роботи користувачів, а також розширюваність самої системи. Об'єднання об'єктів з реляційною моделлю дає добрі результати – ефективність і надійність реляційної бази даних поєднуються з гнучкістю й можливістю моделювання об'єктної структури.

3.13.2 Об'єктні типи

У Oracle8 об'єкти створюються за допомогою **об'єктних типів (object types)**, які описують атрибути й методи конкретного виду об'єктів. Нижче показано, як створювати й використовувати об'єктні типи. Проте для цього повинен бути приступним **засіб роботи з об'єктами (objects option)**. Після поєднання з базою даних Oracle8 з'являється повідомлення, схоже на таке:

Oracle Enterprise Edition Release 8.0.3.0.0. – Production
With the Partitioning and **Objects** options
PL/SQL Release 8.0.3.0.0 – Production

Якщо в повідомленні не зазначається наявність засобу роботи з об'єктами, це означає, що його не було встановлено, і – отже створювати й використовувати об'єктні типи в цьому разі є неможливо. У такій ситуації слід за допомогою програми-інсталятора Oracle підімкнути до системи Oracle засіб роботи з об'єктами.

Створювання об'єктних типів

Об'єктний тип є схожий на модуль і також має опис (специфікацію) й тіло. У описі типу містяться атрибути й попередні оголошення методів, а його тіло складається з реального програмного тексту методів. Синтаксис створювання специфікації типу наведено нижче.

Опис об'єктного типу створюється за допомогою оператора CREATE TYPE… AS OBJECT:

```
CREATE [OR REPLACE] TYPE [схема.] ім'я типу AS OBJECT (
ім'я атрибута тип даних [,ім'я. атрибута тип даних]…
| [{MAP | ORDER} MEMBER опис_функції]
| [MEMBER {опис процедури | опис функції}
[,MEMBER {опис процедури | опис функції}] …]
| [PRAGMA RESTRICT_REFERENCES (ім'я_методу, обмеження)
[,PRAGMA RESTRICT_REFERENCES (ім'я_методу, обмеження)]…]
);
```

де **ім'я_типу –** ім'я нового об'єктного типу, а **схема –** його власник. Спочатку через кому перелічуються атрибути типу:

ім'я_атрибута тип_даних

де **ім'я_атрибуту** – це ім'я атрибута, а **тип_даних –** вбудований тип Oracle, або раніше визначений користувачем тип даних, або посилання на об'єктний тип. Після атрибутів зазначаються методи разом з прагмами PRAGMA

RESTRICT_REFERENCES для цих методів. Наприклад, наступний оператор створює об'єктний тип для студентів:

```
CREATE OR REPLACE TYPE StudentObj AS OBJECT (
  ID              NUMBER(5),
  first_name      VARCHAR2(20),
  last_name       VARCHAR2(20),
  major           VARCHAR2(30),
  current_credits NUMBER(3),
);
```

При створюванні специфікації або тіла типу за допомогою програми SQL*PLUS треба зазначати в рядку, який слідує за визначенням типу, косу рису (/), щоби виконався оператор CREATE TYPE …AS OBJECT (як це робиться при виконанні блока PL/SQL.

Слід зробити декілька **зауважень щодо об'єктних типів**:

1 Оператор CREATE TYPE є оператором DDL, тому його не можна явно зазначати в блоці PL/SQL. Для завдання цього оператора можна скористатися модулем DBMS_SQL.
2 Для створення об'єктного типу слід мати системний привілей CREATE TYPE (що є частиною ролі RESOURCE).
3 Об'єктні типи створюються як об'єкти словника даних. Отже, вони створюються в поточній схемі, якщо лише в операторі CREATE TYPE ... AS OBJECT не зазначено іншої схеми.
4 Атрибути новостворюваного типу зазначаються подібно до полів запису PL/SQL або стовпців таблиці в операторі CREATE TABLE.
5 На відміну від полів запису, атрибути об'єктного типу не можна обмежувати як NOT NULL й ініціалізовувати значеннями за умовчанням.
6 На атрибути об'єкта, як і на запис PL/SQL, можна посилатися за допомогою уточнюючого запису через крапку.

При використанні типів даних до атрибутів об'єктів пред'являється низка вимог. Атрибути об'єктів можуть мати будь-який тип даних Oracle8, **за винятком:**

- типів LONG та LONG RAW. Проте атрибути можуть мати тип LOB;
- будь-яких типів даних, використовуваних для підтримки національних мов: NCHAR, NVARCHAR2 або NCLOB;
- типу даних ROWID;
- типів, приступних лише в PL/SQL, але не в базі даних. До таких типів належать BINARY_INTEGER, BOOLEAN, PLS_INTEGER, RECORD та REF CURSOR;
- типів, визначених з % TYPE або %ROWTYPE;
- типів, визначених в модулі PL/SQL.

Причиною таких вимог є те, що в системі Oracle Release 8 об'єктний тип є об'єктом словника даних. Тому дозволено використовувати лише ті засоби й типи, які застосовуються безпосередньо в базі даних, а типи PL/SQL та конструкції, подібні до %TYPE, в базі даних не застосовуються. У подальших версіях Oracle8 ці обмеження, можливо, буде скасовано і можна буде оголошувати об'єктні типи як локально в блоках PL/SQL, так і в словнику даних.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1 Дэн Хотка. ORACLE8i на примерах. – М.: Издательский дом «Вильямс». – 2001.

2 Oracle 8i/ Энциклопедия пользователя: Компания Advanced Information System. – СПб: ООО «ДиаСофт», 2001.

3 Jason Couchman, Ulrike Schwinn «Oracle8 Certified Professional DBA» Osborne/McGraw – Hill. 1999.

4 Кристофер Аллен. 101: ORACLE PL/SQL: Издательство «Лори», 2001.